

Міністерство освіти і науки України  
Державний заклад  
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

**Оселедько Богдан Олександрович**

## **АНАЛІЗ ТА РОЗРОБКА ДОДАТКУ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ**

**кваліфікаційна робота**

**здобувача вищої освіти другого (магістерського) рівня**

**освітньої програми «Мультимедійні системи»**

**за спеціальністю 121 Інженерія програмного забезпечення**

Особистий підпис \_\_\_\_\_ Богдан ОСЕЛЕДЬКО

Науковий керівник \_\_\_\_\_ Микола СЕМЕНОВ,  
кандидат педагогічних наук, доцент  
кафедри інформаційних технологій  
та систем

В.о. завідувача кафедри \_\_\_\_\_ Микола СЕМЕНОВ,  
кандидат педагогічних наук, доцент  
кафедри інформаційних технологій  
та систем

Полтава – 2024

## **АНОТАЦІЯ**

**Оселедько Б. О.**

**Тема:** Аналіз та розробка додатку візуалізації алгоритмів.

**Спеціальність:** 121 «Інженерія програмного забезпечення».

**Установа:** ЛНУ імені Тараса Шевченка, 2024р.

**Магістерська робота містить:** 63 с., 6 таб., 15 рис., 24 джерела, 3 додатка.

**Об’єкт дослідження** – візуалізатори алгоритмів.

**Предмет дослідження** – візуалізатори алгоритмів сортування.

**Мета роботи** - аналіз візуалізації алгоритмів сортування та розробка Web-модуля візуалізації алгоритмів сортування.

**Результати роботи** – розглянуто опис поточного стану в області візуалізаторів і систем візуалізації. Зокрема, розглянуто застосування візуалізаторів в навчальному процесі і виникаючі при цьому вимоги до візуалізаторів. Наводиться аналіз візуалізаторів алгоритмів з точки зору висунутих вимог. Розглядається оцінка складеності алгоритмів сортування. Проводиться аналіз алгоритмів сортування. Обґрунтовується вибір технології та середовища розробки додатку. Розроблено Web-модуль для візуалізації алгоритмів сортування.

**Ключові слова:** АЛГОРИТМІЧНА ПІДГОТОВКА, ВІЗУАЛІЗАЦІЯ, ВІЗУАЛІЗАТОР, АЛГОРИТМ СОРТУВАННЯ, JAVASCRIPT.

## ANNOTATION

**Oseledko Bohdan**

**Theme:** Analysis and development of an algorithm visualization application.

**Speciality:** 121 "Software Engineering".

**Institution:** Luhansk Taras Shevchenko National University (LTSNU), 2024 year.

**Master's work of:** 63 p., 15 im, 24 sources.

**The object of research** is algorithm visualizers.

**The subject** of the visualizers of sorting algorithms.

**An aim of research is** - analysis of visualization of sorting algorithms and development of a Web-module for visualization of sorting algorithms.

**Job performanes** - The description of the current state in the field of visualizers and visualization systems is considered. In particular, the application of visualizers in the educational process and the resulting requirements for visualizers are considered. An analysis of algorithm visualizers is given from the point of view of the stated requirements. The assessment of the complexity of sorting algorithms is considered. Analysis of sorting algorithms is carried out. The choice of technology and application development environment is justified. A Web-module for visualization of sorting algorithms has been developed.

**Keywords:** ALGORITHMIC PREPARATION, VISUALIZATION, VISUALIZER, SORTING ALGORITHM, JAVASCRIPT.

## ЗМІСТ

<b>ВСТУП.....</b>	<b>6</b>
<b>РОЗДІЛ 1. ВІЗУАЛІЗАТОРИ АЛГОРИТМІВ.....</b>	<b>11</b>
1.1. Застосування візуалізаторів в навчальному процесі.....	11
1.1.1. Реалізація дидактичного принципу наочності в алгоритмічній підготовці здобувачів вищої освіти.....	11
1.1.2. Візуалізатори в навчальному процесі .....	18
1.1.3. Варіанти застосування візуалізатора.....	21
1.1.4. Вимоги до візуалізаторів алгоритмів .....	22
1.2. Огляд візуалізаторів на прикладі алгоритмів сортування.....	23
1.2.1. Підходи до візуалізації алгоритмів сортування .....	23
1.2.2. Огляд візуалізаторів алгоритмів сортувань.....	24
1.2.3. Аналіз візуалізаторів алгоритмів сортувань.....	28
1.3. Висновки до розділу .....	29
<b>РОЗДІЛ 2. АНАЛІЗ АЛГОРИТМІВ СОРТУВАННЯ.....</b>	<b>30</b>
2.1. Оцінка складності алгоритмів сортування.....	30
2.2. Порівняння алгоритмів сортування .....	36
2.3. Висновки до розділу .....	41
<b>РОЗДІЛ 3. РЕАЛІЗАЦІЯ ДОДАТКУ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ СОРТУВАННЯ .....</b>	<b>42</b>
3.1. Вибір технології для побудови візуалізатора .....	42
3.2. Вибір середовища розробки візуалізатора алгоритмів .....	49
3.3. Каскадні таблиці стилів CSS .....	51
3.4. Опис інтерфейсу Web-модуля візуалізації алгоритмів сортування.....	53
3.5. Методичні вказівки по використанню Web-модуля візуалізації алгоритмів сортування.....	55
3.6. Висновки до розділу .....	58
<b>ВИСНОВКИ .....</b>	<b>59</b>
<b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....</b>	<b>61</b>
<b>ДОДАТОК А.....</b>	<b>64</b>

<b>ДОДАТОК Б.....</b>	<b>76</b>
<b>ДОДАТОК В.....</b>	<b>89</b>

## ВСТУП

Сучасна педагогічна наука неперервно розвивається та прогресує, і цей процес впливає на погляди на педагогічний процес. Методи та засоби навчання та виховання стають більш гуманними та ефективними. Однією з ключових особливостей змін в освіті України є перехід до спрямованого формування у суб'єктів навчання здатності творчо діяти, застосовувати знання та досвід на практиці. Міняються цілі навчання, що вимагає коригування змісту та методик навчання, спрямованих на формування світогляду, ціннісних орієнтацій, уміння самостійно вчитися та критично мислити, а також використовувати інформаційно-комунікаційні технології, зокрема персональні комп'ютери.

Стратегічні орієнтири інформатизації освіти в Україні визначені в законах та програмах, таких як Закон України "Про освіту", Державна національна програма "Освіта. Україна ХХІ століття", та Державна програма "Інформаційні та комунікаційні технології в освіті і науці". Ці документи визначають інформаційне суспільство як пріоритетний напрям державної політики та підкреслюють важливість впровадження інноваційних інформаційно-комунікаційних технологій в освітню сферу.

Розв'язання завдань інформатизації навчання, зокрема алгоритмізації та програмування, пов'язано з визначенням теоретичних засад організації процесу навчання з використанням інформаційно-комунікаційних технологій в різних аспектах, таких як психолого-педагогічний, дидактичний та методичний. Наукове обґрунтування, розробка та апробація відповідних засобів навчання алгоритмізації та програмування, а також способів ефективного поєднання інформаційно-комунікаційних технологій із різними методами та формами організації навчання стають невід'ємною частиною цього завдання.

Також важливим є вивчення методичних засад організації навчальної діяльності здобувачів вищої освіти з алгоритмізації та програмування, з

використанням інформаційно-комунікаційних технологій, щоб сприяти формуванню у них алгоритмічної компетентності.

Особливий акцент в навчанні алгоритмізації та програмування приділяється принципу наочності. Використання засобів наочності сприяє ефективному впливу на емоційну сферу здобувачів вищої освіти, підвищує доступність вивченого матеріалу та активізує розумову діяльність здобувачів вищої освіти.

Якщо для оптимізації та систематизації змісту освіти важливо здійснювати його оптимізацію, систематизацію та проблематизацію, то для вдосконалення методики навчання ключовими є індивідуалізація, диференціація та активізація. В даному контексті візуалізація визнається найважливішим напрямком вдосконалення дидактичних засобів. У 2003 році ЮНЕСКО визнало пріоритет візуальної подачі матеріалу в освіті.

Принцип наочності у навчанні є одним із основних положень дидактики, що передбачає побудову навчання на конкретних образах, що сприймаються здобувачами освіти безпосередньо. Вперше цей принцип був сформульований чеським педагогом Я.А.Коменським у XVII столі, а в подальшому використовувався такими педагогами, як Дж.Локк, Ж.Ж.Руссо, Й.Г.Песталоцці, К.Д.Ушинський. Сучасні педагоги розглядають наочність як джерело знань, що формує чуттєві уявлення та поняття, є ілюстрацією до вивчених положень і сприяє розвитку абстрактного мислення. Засоби наочності використовуються як перед вивченням нового матеріалу, так і під час усвідомлення понять, повторення та перевірки знань [23].

За відзначенням педагогів XX та XXI століть, таких як С.П.Баранов, В.І.Бондар, Н.П.Волкова, В.І.Євдокімов, Л.В.Занков, А.І.Зільберштейн, А.М.Маслов, Н.О.Менчинська, М.Г.Моро, А.Розуменко, О.Я.Савченко, Л.М.Скаткін, І.Ф.Харламов, В.В.Ягупов та інші, принцип наочності визнано найбільш визнаним у педагогіці, і його використання простежується з найдавніших часів. Питання розвитку та адаптації цього принципу до сучасних умов з використанням моделей та процесу моделювання у навчанні

було вивчено в другій половині ХХ століття, і дослідженням цієї теми займалися такі вчені, як В.Г.Болтянський, Д.Б.Ельконін, Л.М.Фрідман та інші. Однією з ключових особливостей використання принципу наочності у навчальному процесі в старших класах та ВНЗ займалися Г.Ващенко та С.І.Архангельський. Крім того, питання застосування комп'ютерних моделей у ВНЗ розглядалися в роботах Н.В.Апатової, Т.А.Бороненко, Ю.А.Вороніна, Л.В.Горчакова, І.В.Роберт, І.А.Цвелої та інших дослідників. Цікавими є також роботи Ю.Р.Валькмана та Н.Н.Манько, які висвітлюють аспекти когнітивної візуалізації дидактичних об'єктів для стимулювання навчальної діяльності.

Використання методу демонстраційних прикладів у навчанні програмуванню базується на концепції відомого методиста в цьому напрямку, Н.Вірта, який розглядає програмування як мистецтво конструювання та пропонує метод виділення найпростіших будівельних блоків з існуючих програм з їх систематичним описом. Однак варто враховувати, що програмування – це різнопланова діяльність, яка часто вимагає складної розумової праці. Помилково вважати, що можна звести її лише до використання готових рецептів. Наш метод навчання вимагає уважного відбору та ретельного розгляду типових прикладів. Очевидно, що вивчення прикладів несе різні переваги для кожного здобувача освіти, і його користь значною мірою залежить від кмітливості та інтуїції конкретного здобувача освіти.

Проблемам ролі візуалізації у підвищенні мотивації та активізації навчальної діяльності з алгоритмізації та програмування присвячені дослідження таких авторів, як М. С. Львов, Н. В. Морзе, О. В. Співаковський та інші.

При вивченні алгоритмів обробки інформації через різні структури даних [14,16], велике значення мають візуалізатори алгоритмів, які дозволяють навчально відображати їхню роботу у наочній формі. Це



відкриває можливості для використання сучасних технологій у навчанні програмуванню [13].

Візуалізатор – це програма, що динамічно демонструє застосування алгоритму до обраних даних на екрані комп'ютера. Використання візуалізаторів дозволяє докладно вивчити роботу алгоритмів в кроковому режимі, аналогічному режиму трасування програм.

Для деяких алгоритмів динамічна демонстрація їхньої роботи є більш природною, ніж простий набір статичних ілюстрацій. Візуалізація дозволяє наочно демонструвати як загальні принципи, так і відмінності в механізмах їх дії, зокрема для алгоритмів сортування.

Завдяки візуальній підтримці алгоритмічної підготовки, здобувачі освіти краще засвоюють навчальний матеріал, усвідомлюють його дію діяльнісним методом, що забезпечує їм можливість широкого практичного застосування отриманих знань.

Використання візуалізаторів при формуванні алгоритмічних компетенцій розвиває пізнавальні здібності здобувачів освіти та сприяє їхній здатності самостійно аналізувати та інтерпретувати результати, стимулюючи їх до дослідницької діяльності. Таким чином, здобувачі освіти формують нові професійно важливі навички та вміння, готуючись до успішної професійної діяльності.

Таким чином розробка Web-модуля візуалізації алгоритмів сортування є актуальною темою.

**Об'єкт дослідження** – візуалізатори алгоритмів.

**Предмет дослідження** – візуалізатори алгоритмів сортування.

**Мета роботи** - аналіз візуалізації алгоритмів сортування та розробка Web-модуля візуалізації алгоритмів сортування.

**Для досягнення визначеної цілі важливо вирішити наступні завдання:**

1. Розглянути застосування візуалізаторів в навчальному процесі.
2. Проаналізувати та порівняти алгоритми сортування.

3. Визначити зміст, структуру, функції і дидактичні можливості Web-модуля для реалізації візуалізації алгоритмів сортування.
4. Вивчити програмне забезпечення, яке використовується для створення додатку.
5. Розробити Web-модуль для візуалізації алгоритмів сортування.

**Методи дослідження:** методи теорії алгоритмів та мов програмування, методи комп'ютерної графіки.

Практичною цінністю роботи є розроблений Web-модуль для візуалізації алгоритмів сортування, який можливо використовувати в навчальному процесі.

У першому розділі досліджено становище в галузі візуалізаторів і систем візуалізації, зокрема, їх використання в навчальному процесі та виникнення пов'язаних з цим вимог. Також надано аналіз візуалізаторів алгоритмів з урахуванням висунутих вимог.

Другий розділ присвячено оцінці складності алгоритмів сортування та проведенню аналізу різних алгоритмів сортування.

У третьому розділі перебуває розроблювальна частина роботи. У ньому обґрунтовується вибір технології та середовища розробки, розглядається розробка Web-модуля для візуалізації алгоритмів сортування. Так само в цьому розділі містяться методичні вказівки по роботі з додатком.

## **РОЗДІЛ 1. ВІЗУАЛІЗАТОРИ АЛГОРИТМІВ**

### **1.1. Застосування візуалізаторів в навчальному процесі**

#### **1.1.1. Реалізація дидактичного принципу наочності в алгоритмічній підготовці здобувачів вищої освіти**

Необхідність використання конкретних прикладів у навчальному процесі була теоретично виправдана вперше в XVII столітті чеським педагогом Я.А.Коменським. Він підкреслив, що ефективне навчання базується на чуттєвому сприйнятті, визначивши, що "зовнішні чуття" слід активно залучати для полегшення сприйняття матеріалу [15].

Цей видатний педагог визначив "золоте правило", яке визнає все, що можна сприймати чуттями, як основоположний принцип навчання. Він пропонував, щоб об'єкти сприймалися різними чуттями, якщо це можливо, і наголошував на тому, що чим більше навчання базується на відчуттях, тим воно більш достовірне [15].

У XVIII - на початку XIX століття швейцарський педагог Й.Г.Песталоцці розробив теорію наочного навчання, вважаючи його абсолютною основою будь-якого пізнання. Він підкреслював, що чуттєве сприйняття є єдиним фундаментом людського пізнання і вважав наочне навчання продовженням природних інстинктів [21].

Хоча обидва педагоги розуміли наочне навчання як засіб отримання знань через сприйняття, Песталоцці розглядав його як важливий елемент розвитку логічного мислення та дав психологічне обґрунтування, підкреслюючи роль узагальнення вражень у формуванні понять і їх поступового розуміння [20].

В уявленні видатного педагога XIX століття, К.Д.Ушинського, наочне навчання є формою навчання, яка ґрунтується на конкретних образах, безпосередньо сприйнятих дитиною або під керівництвом вчителя, або завдяки самостійним спостереженням. Ушинський, аналізуючи наочне навчання як педагог, також виконує його психологічний розгляд, вказуючи,

що участь багатьох органів чуття у сприйманні збільшує міцність вражень у механічній, нервовій пам'яті і полегшує їх подальше відтворення.

Ушинський трактує наочність як дидактичний принцип, який пронизує як зміст, так і окремі методи та прийоми навчання, пов'язаний з іншими дидактичними принципами, що є "необхідними умовами навчання".

У "Педагогічному словнику" зауважується, що наочність у навчанні є дидактичним принципом, який базується на конкретних образах, сприйняття яких є безпосереднім для здобувачів освіти. Це розуміння наочного навчання мало панівний характер у педагогічній науці ХХ століття. Зокрема, А.І.Зільберштейн відзначає, що принцип наочності ґрунтується на теорії, що визнає зовнішній світ та його відображення в людському розумі як основоположні для отримання знань.

У 70-80-х роках минулого століття активно досліджували питання моделювання в процесі навчання такі вчені, як В.Г.Болтянський, В.В.Давидов, Д.Б.Ельконін, Л.М.Фрідман. Згідно з літературними джерелами, саме Д.Б.Ельконін вперше звернув увагу на це питання.

У ХХ столітті психологи приділяли багато уваги сприйняттю предметів і явищ оточуючого світу здобувачами середньої освіти та вищої освіти. Вони прийшли до висновку, що "наочність не відокремлює сприйняття і представлення від цілісної аналітико-синтетичної розумової діяльності".

У ХХІ столітті концепція та застосування наочності в дидактиці отримують нові відтінки, з'являються нові терміни, такі як "моделювання", "нова наочність" і інші. Сучасний погляд на це питання визначає нові терміни та розуміння їх специфіки на сучасному етапі.

В сучасній педагогіці трактування принципу наочності взаємодіє із розумінням наочного навчання Я.А.Коменського, Й.Г.Песталоцці, К.Д.Ушинського. Наприклад, у підручнику з педагогіки Волкової Н.П. (2001 р.) зазначається: "Принцип наочності передбачає навчання на основі живого сприймання конкретних предметів і явищ дійсності або їх зображень".

Наочне навчання передбачає, що під час усвідомлення матеріалу слід використовувати різні відчуття, включаючи зорове сприймання, як визначено Г. Ващенком. Відзначається, що сприйняті об'єкти залишають в нашій свідомості конкретні образи та уявлення, які, у свою чергу, сприяють розвитку вищих форм мислення. Особливо це важливо в дитинстві, коли виключно вербальний спосіб подачі інформації може сприяти формуванню поверхневого та неповного вербального типу мислення.

У технічних навчальних закладах використання наочності у вивченні матеріалу доповнюється використанням технічних засобів, які розширюють можливості чутливих органів людини. С.І. Архангельський підкреслює, що технічні засоби навчання збагачують наочність, дозволяючи подавати інформацію у більш активній формі, впливаючи на мисленнєву діяльність здобувачів вищої освіти та їхні емоційний стан.

Під час вивчення конкретного предмету наочність включає як чутливий, так і мислительний аспекти, сприяючи виявленню зовнішніх ознак та властивостей предмету. За словами С.І. Архангельського, однією з важливих функцій наочності є створення уявлень, які утворюють основу для формування понять. Він розглядає наочність як перехід від конкретного до абстрактного, від сприйняття до мислення, що визначається як «перехід від конкретного до абстрактного, від сущого до мислі, від ознак і уявлень до понять і визначень».

Головним завданням наочності в навчанні є забезпечення зв'язку між спостереженнями та уявленнями, що допомагає здобувачам вищої освіти глибше розуміти сутність вивченого об'єкта. С.І. Архангельський виділяє два типи наочності: "безпосередня наочність", що базується на спостереженнях дійсності, та "опосередкована наочність", яка відображає явище, подію чи предмет вивчення у наглядній формі, що відображає його сутність та взаємозв'язки. Таким чином, наочність в навчанні сприяє відкриттю зовнішніх ознак та властивостей об'єкта, що допомагає активізувати пізнавальну діяльність та механізм сприйняття інформації.

В.М. Вергасов стверджує, що «наочність впливає на формування початкової нейронної моделі образу, поняття або явища на етапі сприйняття. Таким чином, важливо, щоб наочність якнайбільше сприяла переходу інформації з зовнішнього середовища до пам'яті інтелекту».

Навчальні функції наочності були вивчені традиційними дослідниками дидактики, такими як Л.В. Занков, Ф.І. Менчинська, М.І. Махмутов, В.А. В'ялих та інші.

Л. В. Занков розрізняє три основні функції, де наочність виступає як:

- джерело інформації;
- засіб ілюстрації інформації;
- опора для усвідомлення зв'язків між явищами, предметами та поняттями.

М.І. Махмутов додає ще одну функцію, де наочність розглядається як засіб формування проблемних ситуацій, що в свою чергу сприяє розвитку творчих навиків під час самостійної діяльності майбутнього фахівця. В.А. В'ялих приписує наочності функцію активізації пізнавальної діяльності здобувача вищої освіти та розглядає її як основу абстрактного мислення.

Сучасна концепція наочного навчання базується на узгодженні минулих досягнень з сучасними вимогами. Таким чином, традиційне розуміння принципу наочності вже не відповідає сучасній дійсності і вимагає подальшого розвитку та удосконалення. Застосування моделювання в навчанні вважається наступним етапом у використанні наочного матеріалу.

Сучасний прогрес інформаційних технологій і комп'ютерної техніки розширює можливості використання наочності на новому рівні, підвищуючи її інформаційну та пізнавальну складову.

Введення комп'ютеризації в освітній процес відкриває нові перспективи для розвитку мислення та надає нові можливості для активного навчання. Оскільки візуально-образне мислення грає важливу роль у житті людини, використання його в навчальному процесі виявляється дуже ефективним. Комп'ютерна графіка може застосовуватися на всіх етапах

навчання: при поясненні нового матеріалу, закріпленні, повторенні та контролі. Отже, важливо перейти від розгляду наочності як допоміжного засобу для навчання алгоритмізації до повноцінного використання візуального мислення під час алгоритмічної підготовки здобувачів вищої освіти.

За визначенням В.П.Зінченка, візуальне мислення представляє собою людську діяльність, що породжує нові образи та створює нові візуальні форми з певним смисловим навантаженням, які роблять значення видимим.

В сучасному розумінні принципу наочності, визначеному В.П.Зінченком, визначається логіка пізнання, що переходить від чуттєво-наочного до абстрактно-логічного, від наочності чуттєво-конкретної (такої як об'єкти в природі, малюнки, макети тощо) до наочності абстрактної і символічної (схеми, таблиці, діаграми, графіки). Принцип наочності пов'язаний із роботою органів чуття, таких як зорових, слухових, тактильних і інших. Однак виявилось, що принцип наочності має більше значень.

Тепер обговорюється роль наочності як засобу переходу від чуттєвого матеріалу до його абстрактного тлумачення і від абстрактного до глибшого розуміння чуттєвого. Чуттєвий матеріал, який включає природну наочність (натуральні предмети й об'єкти), образотворчу наочність (малюнки, фотографії та ін.) та реальні моделі, представляє собою зміст наочності. На етапі переходу до абстрактних понять необхідні інші засоби наочності, такі як схеми, таблиці, графіки та символи. Цей вид наочності відомий як абстрактно-символічний і допомагає осмислити сутність і динаміку явищ і процесів, які вивчаються.

Використання наочності слід підпорядковувати конкретній меті, розвитку самостійності й активності здобувачів освіти, з урахуванням їх вікових особливостей. Вона повинна мати змістовний, естетичний оформлення та відповідати психологічним законам сприймання.

Для досягнення цієї мети слід дотримуватися таких правил реалізації принципу наочності:

1. Запам'ятовування предметів у натурі, на картинках або моделях ефективніше та швидше, ніж запам'ятовування поданого словесно, усно або письмово.
2. Дитина мислить формами, фарбами, звуками, образами взагалі, тому наочне навчання, яке будується на конкретних образах, є доцільним.
3. Що тільки можна, діти мають сприймати відчуттями (зором, слухом, нюхом тощо), особливо в початковому навчанні.
4. Наочність – це не мета, а лише засіб досягнення поставленої мети.
5. Поняття доходять до свідомості здобувачів освіти легше, коли вони підкріплені конкретними фактами, прикладами та образами. Для цього необхідно застосовувати всі види наочності.
6. Слід використовувати наочність як самостійне джерело інформації для створення проблемних ситуацій.
7. Спостереження здобувачів освіти повинні бути систематизованими і перебувати у співвідношенні причин і наслідків незалежно від часу їх набуття.
8. Застосовуючи наочні засоби, слід розглядати їх із здобувачами освіти спочатку загально, потім – головне й другорядне, і, нарешті, знову загально.
9. Надмірна кількість наочних посібників розсіює увагу здобувачів освіти і заважає осмислити головне.
10. Використовуючи наочність, слід актуалізувати чуттєвий досвід здобувачів освіти, конкретизувати та ілюструвати ті поняття, які формуються.
11. Намагатися виготовляти наочні посібники разом із здобувачами освіти.
12. Старанно готувати наочність до занять.



13. Науково обґрунтовано застосовувати сучасні засоби наочності, такі як поліекранна проєкція, навчальне телебачення, відеозапис, кодослайди, комп'ютери, проєктори та інші технічні засоби, володіючи методикою їх використання.
14. У кабінетній системі навчання можливості застосування наочності кращі, що вимагає ретельного планування та дозування наочності.
- 15.3 віком здобувачів освіти предметна наочність повинна поступово поступатися місцем символічній, і викладач має приділяти увагу адекватності розуміння суті явищ і їх наочного подання.
16. Надмірне захоплення наочністю може створити штучні перешкоди для глибокого оволодіння знаннями, гальмуючи розвиток абстрактного мислення та розуміння загальних закономірностей.

Дидактичні засоби є необхідним елементом правильно організованого навчального процесу. Незважаючи на те, що їх вплив на кінцеві результати навчання не є вирішальним, дидактичні засоби сприяють підвищенню ефективності методів навчання, збагачуючи їх. Вони, відповідно до В. Оконя, допомагають реалізації принципу наочності, покращуючи умови безпосереднього сприйняття дійсності і надаючи матеріал у формі вражень та спостережень, що становлять основу для опосередкованого пізнання, розумової діяльності та практичної роботи.

Один з ефективних методів, сприяючи розвитку алгоритмічної та технологічної компетентності, полягає у використанні комп'ютерних моделей інформаційних процесів та об'єктів в процесі вивчення алгоритмізації. Дослідники вказують на різні програмні засоби, такі як виконавці алгоритмів, комп'ютерні навчальні середовища, електронні тренажери, імітатори, моделі інформаційних процесів і обчислювальних систем, які допомагають у цьому процесі. Особливу роль відіграють візуалізатори алгоритмів.

Важливі властивості візуальних засобів, які стали відомі завдяки дослідженням в області візуалізації дидактичних об'єктів, мають сприяти трансформації основних елементів освітнього процесу. Досягнення педагогічної мети, зокрема формування моделюючого середовища, включаючи технології, засоби і способи візуалізації дидактичних об'єктів, визначається як необхідна умова для позитивної активізації навчальної діяльності. Моделювання, розглядає як активний спосіб формування когнітивно-візуальних образів вивчаємого об'єкта і їх використання для раціоналізації пізнання, дозволяє створювати семантичний простір для дослідження та забезпечує можливість експериментування на моделі, включаючи логічне узагальнення, рефлексію та інші форми розумової діяльності [18].

Науковці в ІТ Інституті Херсонського державного університету зосереджуються на важливих питаннях виявлення потенціалу когнітивної візуалізації в процесі алгоритмічної підготовки здобувачів вищої освіти. Під керівництвом М.С. Львова та О.В. Співаковського тут розроблений програмно-методичний комплекс під назвою "Відеоінтерпретатор алгоритмів пошуку та сортування" [17].

### **1.1.2. Візуалізатори в навчальному процесі**

Візуалізатори алгоритмів широко застосовуються для навчання інформатики [11, 22]. Дослідження [30] показали, що застосування візуалізаторів допомагає здобувачам освіти глибше і швидше зрозуміти з'ясовний матеріал. При цьому так само виявлена велика зацікавленість здобувачів освіти у лекціях із застосуванням візуалізаторів, ніж без таких.

Процес навчання при використанні для вивчення алгоритмів книг відбувається одним із таких способів:

1. Статичне сприйняття тексту з динамічною прокруткою в голові.
2. Реалізація алгоритму на вибраній мові програмування, або копіювання алгоритму з книги з тим, щоб динамічно, крок за кроком, відстежити дію алгоритму на тестових прикладах.

Перший спосіб є досить складним для більшості здобувачів освіти, оскільки вимагає програмістської уяви. Він доступний лише досвідченим програмістам, а не здобувачам освіти старших класів школи або молодших курсів інституту, які тільки приступають до вивчення алгоритмів. Другий спосіб є більш прямолінійним і доступним, однак акцентує увагу не на суті алгоритму, а на його програмній реалізації і тонкощах використовуваної мови програмування.

Таким чином, традиційне статичне викладання матеріалу по алгоритмізації та програмування є неефективним з точки зору вивчення алгоритмів. Крім того, якщо розглянути наведені способи навчання з точки зору використання на лекціях з алгоритмізації та програмування, то очевидно, що вони також не ефективні. Якщо перший спосіб вимагає певних зусиль, то другий і зовсім неможливо реалізовувати на лекційних заняттях.

Подальші дослідження в галузі викладання алгоритмізації та програмування призвели до виникнення в середині вісімдесятих років [22] нового підходу до викладання - з'явилися візуалізатори алгоритмів дискретної математики.

Візуалізатор - це програма, яка ілюструє виконання алгоритма при певних вхідних даних. Прикладами візуалізаторів можуть служити, наприклад, програма, що займається побудовою графіка функції, або програма, візуально моделююча який-небудь фізичний процес. Стосовно до дискретної математики й програмування, візуалізатори зазвичай моделюють деякі алгоритми, даючи можливість навчаючому за допомогою інтуїтивно зрозумілого інтерфейсу проходити алгоритм крок за кроком від початку до кінця, а при необхідності, і навпаки.

Перераховуючи видатні характеристики візуалізаторів, можна відзначити наступне:

1. Простота використання, яка обумовлена зрозумілістю інтерфейсу. Таким чином, для роботи з візуалізатором, зазвичай, не потрібна спеціальна підготовка.

2. Чіткість і простота представлення візуалізованого процесу.
3. Компактність візуалізаторів, що спрощує їх передачу в мережі Інтернет, особливо важливо при дистанційному навчанні.

Візуалізатори у розглянутій області вирішують наступні завдання, що виникають під час навчання:

1. Графічне та текстове пояснення дій алгоритму на конкретних наборах вхідних даних. При цьому розуміння алгоритму не є обов'язковим, оскільки сам візуалізатор повинен пояснити дію алгоритму.
2. Надання користувачеві інструменту, що реалізує даний алгоритм. У результаті здобувач освіти звільняється від необхідності виконувати кроки алгоритму, оскільки їх автоматично виконує візуалізатор.

Функції візуалізатора, як правило, включають:

1. Крокове виконання алгоритму.
2. Перегляд дії алгоритму при різних наборах вхідних даних, включаючи ті, що вводяться користувачем.
3. Перегляд дії алгоритму в динаміці.
4. Перезапуск алгоритму на поточному наборі вхідних даних.

Динамічна візуалізація наочно демонструє таку характеристику алгоритму, як трудомісткість (особливо при поетапній демонстрації). Для деяких алгоритмів (наприклад, машини Тюрінга) динамічний варіант демонстрації представляється більш натуральним, ніж будь-який набір статичних ілюстрацій.

Основні завдання візуалізатора алгоритму включають:

1. Відображення вхідних і вихідних даних у наочній формі - представлення абстрактного поняття в картинках для полегшення сприйняття, як для початківців, так і для більш досвідчених здобувачів освіти.
2. Відображення внутрішніх службових змінних.
3. Відображення процесу впливу алгоритму на вхідні дані та внутрішні змінні:

- a) зміна даних;
- b) копіювання і переміщення даних;
- c) процес прийняття рішень;
- d) коментарі до кожної дії алгоритму;
- e) відображення роботи алгоритму по кроках;
- f) можливість введення даних для апробації алгоритму на різних наборах вхідних даних.

### **1.1.3. Варіанти застосування візуалізатора**

При навчанні візуалізатори можуть бути використані декількома різними способами. Опишемо деякі з них.

На початку візуалізатори алгоритмів використовувалися як супроводжуючий матеріал на лекціях. При цьому, викладач заздалегідь готує візуалізаційний матеріал і показує його на лекціях, пояснюючи роботу алгоритму. Візуалізатори так само можуть бути використані для більшого залучення здобувачів освіти в процес навчання. Одним з таких способів є відображення деякого стану алгоритму і пропозиція здобувачам освіти визначити, яку дію виконує алгоритм. Після отримання відповідей, вони перевіряються шляхом виконання кроків вперед. Візуалізатор так само дозволяє детально розібрати чому було здійснено та обрано саме цю дію, а не другу. У такому режимі візуалізатори так само можуть бути використані для перевірки знань.

Інший спосіб використання візуалізаторів - початкове знайомство з алгоритмом. Здобувач освіти спочатку працює з візуалізатором, складаючи для себе загальну схему алгоритму. Пізніше викладач дає повний опис алгоритму, після чого здобувач освіти може повернутися до роботи з візуалізатором.

Ще один спосіб використання візуалізаторів полягає в тому, що здобувачі освіти самі створюють візуалізатори, досконально вивчаючи при цьому візуалізуючі алгоритми.

Отримані таким чином візуалізатори можуть бути опубліковані в мережі Internet і використані наведеними вище способами.

Важливою областю використання візуалізаторів є дистанційне і самостійне навчання. У цьому випадку велике значення має можливість завдання користувачем вхідного набору даних.

Таки чином, здобувач освіти може не запитувати викладача, що буде при обробці деякого набору даних, а ввести його в візуалізатор і подивитися самостійно.

#### **1.1.4. Вимоги до візуалізаторів алгоритмів**

Для ефективного використання в навчальному процесі візуалізатори алгоритмів повинні відповідати наступним вимогам:

1. Керованість: можливість встановлювати власні вхідні дані та спостерігати за роботою алгоритму на них.
2. Інтерактивність: при використанні візуалізатора повинна бути можливість виконувати кроки як уперед, так і назад, а також використовувати режим автоматичної візуалізації.
3. Історія: візуалізатор повинен дозволяти здійснювати будь-яку кількість кроків назад, відображаючи стан алгоритму на конкретний момент.
4. Відображення ходу виконання алгоритму.
5. Можливість коментування виконання програми.
6. Простота використання: візуалізатор повинен бути зрозумілим для користувачів без спеціальної підготовки.
7. Доступність: можливість отримувати доступ до візуалізаторів не тільки під час занять.
8. Платформонезалежність: візуалізатор повинен працювати незалежно від платформи та операційної системи.
9. Незалежність від мережі: для роботи візуалізатора необов'язковий доступ до мережі.
10. Зручність створення візуалізаторів алгоритмів: простота використання системи візуалізаторів для написання нових візуалізаторів.

11. Керованість, інтерактивність і історія дозволяють глибше вивчити роботу алгоритму, ніж візуалізація на попередньо вибраних наборах даних.

Простота використання є важливою для самостійного навчання, оскільки користувачі рідко звертають увагу на інструкції. Можливості відображення кроку алгоритму і коментування програми є ключовими для пояснення простих і складних алгоритмів відповідно. Доступність, платформонезалежність і незалежність від мережі дозволяють використовувати візуалізатори як на лекціях і практичних заняттях, так і для самостійного навчання.

## **1.2. Огляд візуалізаторів на прикладі алгоритмів сортування**

Розглянемо візуалізатори алгоритмів, зосередившись на прикладі алгоритмів сортування, оскільки вони є ключовим елементом при вивченні алгоритмізації та програмування і є важливою складовою багатьох інших алгоритмів.

### **1.2.1. Підходи до візуалізації алгоритмів сортування**

Давайте розглянемо декілька підходів до візуалізації алгоритмів сортування:

1. *Числовий підхід*: відсортований масив представляється набором чисел, які входять в його склад. Під час порівняння та обміну елементів вони виділяються, а у деяких випадках може використовуватися анімація обміну елементів [26].
2. *Гістограмний підхід*: відсортований масив візуалізується у вигляді гістограми, де обмін елементів відображається як фізичне переміщення відповідних чисел у вигляді стовпців [27].
3. *Історичний підхід*: весь процес сортування представляється як єдине зображення, при цьому значення елементів зазвичай позначаються кольорами [26].

На рис. 1.1. наведені приклади зображень, які генерують візуалізатори, що використовують описані підходи.

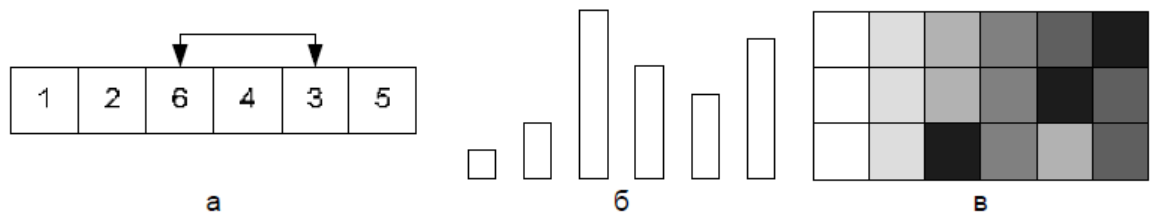


Рис. 1.1. Підходи до візуалізації алгоритмів сортування «числовий» (а), «гістограммний» (б) та «історичний» (в)

Зазвичай відображається стандартний набір алгоритмів сортування, до якого входять обмінна, бульбашкова, сортування простим вибором, сортування простою вставкою, пірамідальна, злиття та швидке сортування.

### 1.2.2. Огляд візуалізаторів алгоритмів сортувань

У наведеному нижче огляді в заголовку підрозділів наводиться назва університету, в якому виконаний візуалізатор.

#### **Brown University (BALSA)**

Система візуалізації BALSA є одним з перших дослідів створення візуалізаторів алгоритмів. Принципи побудови цієї системи описані в роботі [26].

Ця система дозволяє проводити візуалізацію одночасно на основі кількох підходів, наприклад, «гістограммного» і «числового». При цьому відповідні зображення оновлюються синхронно зі зміною даних. Також є можливість використовувати «історичний» підхід, при якому на одному зображенні показується процес сортування в цілому.

Візуалізатори, в основному, демонструють зміни в сортованих даних. При цьому відображення алгоритму та коментарів відсутня, а процес візуалізації коментується викладачем. Сама система дозволяє відображати коментарі і кроки алгоритму, але це практично не використовується.

Візуалізація виконується за допомогою заздалегідь написаних програм на мові сценаріїв. Можливість ручного введення даних не передбачена. Візуалізатори виконуються у вигляді модулів, що вбудовуються в систему BALSA, і не є платформонезалежними. Це також обмежує їх доступність.

#### **State University of New York, College at Brockport**



Колекція візуалізаторів сортувань розташована за адресою [32]. Представлений «стандартний» набір візуалізаторів сортувань, створених на загальному ядрі. При цьому для алгоритму пірамідального сортування використано спеціальне рішення.

При візуалізації застосовується «гістограммний» підхід. Візуалізація виконується на заданому при створенні візуалізатора наборі даних. Існує можливість здійснювати кроки назад за алгоритмом, але перевірка, виконана автором, показала, що вона не працює на деяких наборах вхідних даних.

Присутні коментарі до поточних дій, але алгоритми представлені тільки на супровідних сторінках. Візуалізатори виконані у вигляді Java-апплетів, що визначає їх доступність і платформонезалежність. Для візуалізації зв'язку з сервером не потрібно. Таким чином, ці візуалізатори є автономними.

### **Princeton University**

Візуалізатори виконані у вигляді єдиного Java-апплета, доступного за адресою [31]. На додаток до «стандартного» набору візуалізаторів сортування візуалізуються алгоритми побудови опуклих оболонок. Для візуалізації застосовується як «числовий», так і «гістограммний» підходи з можливістю вибору. Вихідний набір даних може бути задано тільки випадково.

При візуалізації алгоритми не відображаються, так як використана візуалізація даних. З тієї ж причини вироблені дії не коментуються. Третій недолік - неможливість здійснювати шаги назад за алгоритмом.

### **Hope College**

Візуалізатори сортування, виконані в єдиній оболонці, доступні за адресою [28]. Для візуалізації використовується «гістограммний» підхід. Введення вхідних даних неможливе. Трасування алгоритму у зворотньому напрямку не передбачено.

При візуалізації відображається код програми і підсвічується поточний оператор, але вироблені дії не коментуються. Візуалізатор виконаний у вигляді Java-аплета і є платформонезалежним і автономним.

### **University of Joensuu (Jeliot)**

У прикладах до системи візуалізації Jeliot [29] наведені тільки візуалізатори бульбашкової та швидкої сортувань.

При виконанні алгоритмів використовується низькорівнева візуалізація, що відображає не тільки виконання операторів, але й розрахунки виразів. При цьому підсвічується поточний оператор. Введення вихідних даних виконується через спеціальні класи вводу-виводу. Трасування алгоритму у зворотному напрямку неможливе. Для перегляду візуалізаторів, побудованих на базі системи Jeliot, потрібна присутність самої системи візуалізації, виконаної в вигляді Java-додатки, що постачаються для різних платформ. Таким чином, система не є в повній мірі платформонезалежною.

### **СПбДУ ІТМО (Казаков М.А.)**

В роботі [12] описаний візуалізатор пірамідального сортування, доступний за адресою [25].

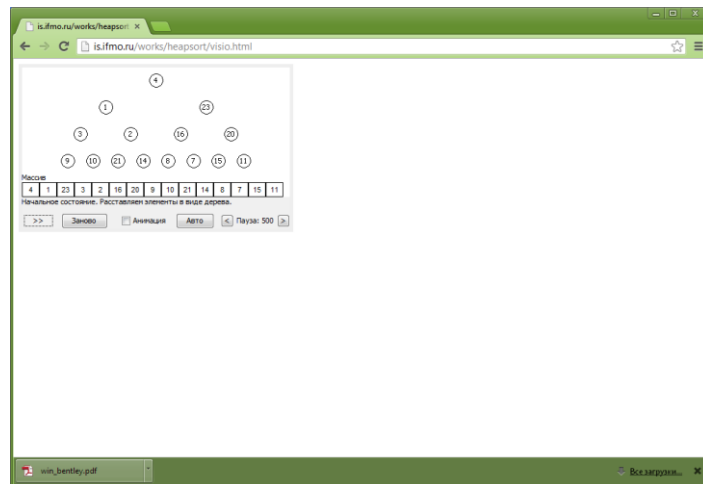


Рис. 1.2. Візуалізатор пірамідального сортування

Даний візуалізатор заснований на автоматном підході, викладеному в роботі [24]. При візуалізації на екрані відображаються стан масиву, піраміда та коментар до виконуваної дії. Введення вихідних даних непередбачено.

Алгоритм трасується тільки в прямому напрямку. Візуалізатор виконаний у вигляді Java-аплета і є платформонезалежним.

**НДІ ІТ Херсонський державний університет (М.С.Львов та О.В.Співаковський)**

Програмно-методичний комплекс "Відеоінтерпретатор алгоритмів пошуку та сортування" розроблено для використання під час вивчення курсу "Основи алгоритмізації і програмування" здобувачами освіти вищих навчальних закладів. Цей комплекс є ефективним інструментом для освоєння алгоритмів, мов програмування, налагодження програм, а також для вдосконалення навичок логічного розробки алгоритмів та програм [17].

Задачі, які вирішує програмно-методичний комплекс, охоплюють різноманітні алгоритми обробки масивів даних, зокрема сортування та пошук унікальних елементів (максимумів, мінімумів і т.д.). Використання цього комплексу дозволяє вивчати теми, пов'язані з мовою програмування, вивченням вспоміжних алгоритмів, рекурсією і іншими аспектами. Робочою мовою програмування в цьому комплексі є Паскаль. Основною перевагою є можливість візуалізації процесу виконання алгоритмів в динаміці, що сприяє кращому розумінню основних понять алгоритмізації та програмування.

Логічним розвитком цього додатку стало створення WEB-орієнтованого Інтегрованого середовища курсу "Основи алгоритмізації та програмування" у 2007-2009 роках. Це середовище розроблено для використання в навчальному процесі при вивченні тем, пов'язаних з алгоритмами обробки масивів, задачами вибору, пошуку та впорядкування даних.

Інтегроване середовище курсу «Основи алгоритмізації та програмування» складеться з наступних модулів: електронний посібник, бібліотека лекцій, бібліотека задач, середовище демонстрації програм (рис. 1.3), система поточного та підсумкового контролю знань, що містить алгоритмічні тести та електронний журнал.

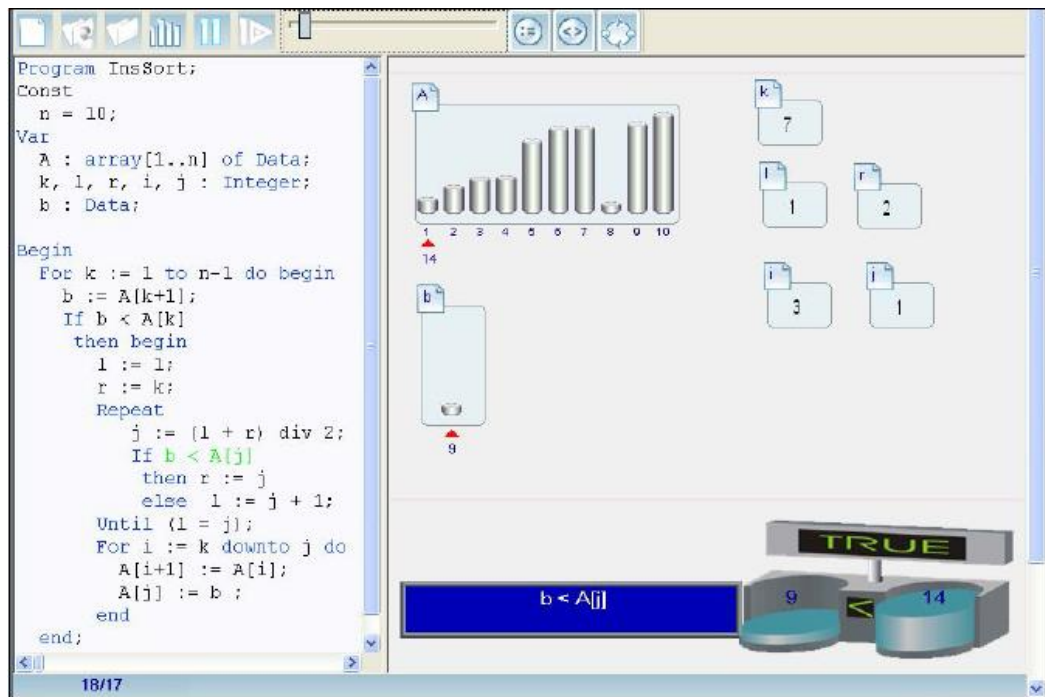


Рис. 1.3. Візуалізація алгоритму сортування вставками у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування»

Модуль "Середовище демонстрації" призначено для використання під час лекцій та проведення практичних та лабораторних робіт з метою наочної демонстрації виконання алгоритмів і організації обчислювального експерименту для аналізу їх ефективності. Однією з основних переваг модуля "Середовище демонстрації" є можливість візуалізації як класичних алгоритмів, які входять до колекції системи, так і алгоритмів, які розробив користувач.

### 1.2.3. Аналіз візуалізаторів алгоритмів сортувань

Розглянемо описані візуалізатори з точки зору вимог, викладених у розділі 1.1.4. Порівняльна характеристика розглянутих візуалізаторів приведена в таблиці 1.1. Її стовпці відповідають властивостям, зазначеним в розділі 1.1.3. При цьому використовуються такі позначення:

- «+» - Властивість виконується;
- «-» - Властивість не виконується;
- «±» - властивість виконується частково.

**Порівняльна характеристика візуалізаторів алгоритмів сортувань**

<b>Візуалізатори</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>
Brown University	-	+	±	+	±	±	-	-	-	-
SUNY Brockport	-	±	+	±	-	+	-	+	+	+
Princeton University	±	-	+	-	-	-	±	+	+	+
Hope College	-	-	+	-	+	-	+	+	+	+
Jeliot	+	-	-	-	+	-	+	+	±	+
СПбГУ ИТМО (Казаков М.А.)	-	-	+	-	-	+	+	+	+	+
НДІ ІТ ХДУ (М.С.Львов та О.В.Співаковський)	±	+	±	+	+	+	+	+	±	+

З наведеної таблиці випливає, що не один з розглянутих візуалізаторів не задовольняє всім висунутим вимогам. Тому необхідно розробити нову систему візуалізації, для якої всі зазначені властивості виконуються.

**1.3. Висновки до розділу**

В данному розділі розглянуто застосування візуалізаторів в навчальному процесі, зокрема сформульовано вимоги до візуалізаторів, які застосовуються при навчанні. Далі розглянуті поточний стан загальнодоступних візуалізаторів на прикладі візуалізаторів сортувань і вказані їх недоліки.

## РОЗДІЛ 2. АНАЛІЗ АЛГОРИТМІВ СОРТУВАННЯ

### 2.1. Оцінка складності алгоритмів сортування

Нехай  $A$  представляє алгоритм для вирішення конкретного класу задач, і  $N$  визначає розмірність окремої задачі у цьому класі, яка може представляти розмір масиву, кількість вершин у графі і так далі. Позначимо  $f_A(N)$  функцію, яка визначає верхню межу максимальної кількості основних операцій (додавання, множення і так далі), які повинен виконати алгоритм  $A$ , розв'язуючи задачу розмірності  $N$ . Якщо ця функція зростає не швидше, ніж деякий поліном від  $N$ , ми будемо називати алгоритм  $A$  поліноміальним. У протилежному випадку, якщо зростання цієї функції вище, ніж поліноміальне,  $A$  вважатиметься експоненціальним алгоритмом.

Виявляється, що між цими двома класами алгоритмів існує значна відмінність: у випадку великих розмірностей задач, які, як правило, є найцікавішими на практиці, поліноміальні алгоритми можуть бути успішно виконані на сучасних комп'ютерах, тоді як експоненціальні алгоритми, як правило, непридатні для практичного використання при таких розмірах задач. Зазвичай експоненціальні алгоритми вимагають повного перебору всіх можливих варіантів розв'язку, і через практичну нереалізованість такого підходу розробляються інші стратегії для їх вирішення.

Наприклад, навіть у випадку, коли існує експоненціальний алгоритм для знаходження оптимального розв'язку деякої задачі, на практиці частіше використовуються ефективніші поліноміальні алгоритми для знаходження необов'язково оптимального, але прийняттого розв'язку (наближеного до оптимального). Однак, навіть якщо задачу можна вирішити за допомогою поліноміального алгоритму, його можна побудувати лише після глибокого розуміння суті поставленої задачі.

Поліноміальні алгоритми також можуть істотно відрізнятися в залежності від ступеня полінома, що апроксимує залежність  $f_A(N)$ . Оцінювання часової складності алгоритму проводиться із застосуванням

відношення  $O$  («велике  $O$ »): кажуть, що  $f_A(N)$  зростає як  $g(N)$  для великих  $N$  і записується це як  $f_A(N) = O(g(N))$ . Якщо існує позитивна константа  $Const > 0$ , така, що  $\lim_{N \rightarrow \infty} f_A(N)/g(N) = Const$ , то оцінка  $O(g(N))$  називається асимптотичною часовою складністю алгоритму.

Оцінка  $O(g(N))$  для функції  $f_A(N)$  застосовується, коли точна величина  $f_A(N)$  невідома, а відомо лише порядок зростання часу, затрачуваного на розв'язання задачі розмірністю  $N$  за допомогою алгоритму  $A$ . Абсолютні значення  $f_A(N)$  залежать від конкретної реалізації, тоді як  $O(g(N))$  є характеристикою цього алгоритму. Наприклад, якщо часова асимптотична складність алгоритму дорівнює  $O(N^2)$  (такий алгоритм називається квадратичним), то при збільшенні  $N$  час розв'язання задачі збільшується як квадрат  $N$ . Факт експоненціальної складності алгоритму в термінах введеної символіки можна записати як  $f_A(N) = O(k^N)$ , де  $k$  — як правило ціле число більше за одиницю.

Інший вид оцінки пов'язаний з введенням «малого  $o$ »: кажуть, що  $f_A(N)$  зростає не швидше від  $g(N)$  для великих  $N$ , що записується  $f_A(N) = o(g(N))$ , якщо  $\lim_{N \rightarrow \infty} f_A(N)/g(N) = 0$ . Наприклад, очевидно, що  $x^2 = o(x^5)$ ,  $\sin(x) = o(x)$ . Інший приклад: алгоритм  $A$  є поліноміальним, якщо  $f_A(N) = o(P_k(N))$ , де  $P_k(N)$  — деякий поліном від  $N$  степеня  $k$ . Так, алгоритм, асимптотична складність якого дорівнює  $o(N \log N)$ , належить до поліноміальних.

Для оцінки складності більшості реальних алгоритмів зазвичай вистачає використання логарифмічних, степеневих та показникових функцій, а також їх комбінацій у вигляді сум, добутків та підстановок. Всі ці функції мають монотонне зростання і можуть бути виражені простими аналітичними виразами.

**Алгоритм сортування** — це процедура, призначена для впорядкування лінійного списку (масиву) елементів.

## Постановка задачі.

**Вхід алгоритму:** послідовність з  $n$  чисел  $a_1, a_2, a_3, \dots, a_n$

**Вихід алгоритму:** перестановка  $(a_{\pi(1)}, a_{\pi(2)}, \dots, a_{\pi(n)})$  вхідної послідовності таким чином, що  $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$  ( $\pi$  — перестановка послідовності чисел  $1 \dots n$ ).

**Вхідна послідовність найчастіше представляється у вигляді масиву** із  $n$  елементів, хоча може мати інше подання, таке як зв'язаний список.

## Структури даних

На практиці елементи, які підлягають впорядкуванню, рідко обмежуються просто числами. Зазвичай кожен такий елемент є записом, в якому присутній ключ для впорядкування, а також інша додаткова інформація. У практиці алгоритм сортування реалізується так, щоб разом з ключами переміщувати й супутню інформацію. Якщо інформація в кожному записі є об'ємною, то для мінімізації копіювання великої об'ємної інформації впорядкування може відбуватися не безпосередньо в масиві елементів, а в масиві вказівників на ці елементи.

Сам метод сортування не залежить від того, чи включає в себе впорядкування лише чисел, чи також супутню інформацію. Таким чином, при описі алгоритмів для спрощення припускається, що елементи є числами.

## Характеристики алгоритмів

Мабуть, не існує жодної іншої проблеми, яка породила б стільки різноманітних рішень, як задача сортування. Чи існує якийсь "універсальний", найкращий алгоритм? Загалом кажучи, такого не існує. Проте, можна обрати метод, який оптимально працює, враховуючи приблизні характеристики вхідних даних.

Щоб обґрунтовано вибрати такий метод, розглянемо параметри, за якими буде проводитися оцінка алгоритмів.

1. *Час сортування*- це основний параметр, що характеризує швидкодію алгоритму.



2. *Пам'ять*: багато алгоритмів вимагають виділення додаткової пам'яті для тимчасового зберігання даних. При оцінці використаної пам'яті не враховуватиметься місце, яке займає початковий масив, і незалежні від вхідної послідовності витрати, наприклад, на зберігання коду програми.
3. *Стабільність*: стабільне сортування не змінює взаємне розташування рівних елементів. Така властивість може бути корисною, якщо елементи складаються з декількох полів, і сортування відбувається за одним з них, наприклад, за *x*.



Рис. 2.1. Приклад роботи стабільного сортування

Взаємне розташування рівних елементів з ключем 1 і додатковими полями "a", "b", "c" залишилося тим самим: елемент з полем "a", потім - з "b", потім - з "c".

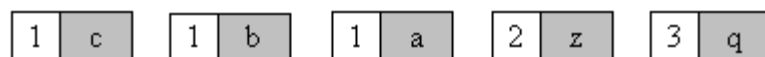


Рис. 2.2. Приклад роботи нестабільного сортування

Взаємне розташування рівних елементів з ключем 1 і додатковими полями "a", "b", "c" змінилося.

4. *Природність поведінки* - здатність методу ефективно опрацьовувати вже впорядковані чи частково впорядковані дані. Алгоритм володіє природністю поведінки, якщо він враховує цю особливість вхідної послідовності і функціонує ефективніше.

Ще однією ключовою характеристикою алгоритму є його область застосування. Це розділяється на дві основні категорії:

1. **Внутрішнє сортування:** цей тип алгоритмів працює з даними, які повністю містяться в оперативній пам'яті та доступні для довільного доступу.

2. **Зовнішнє сортування:** такі алгоритми впорядковують інформацію, яка знаходиться на зовнішніх носіях. Це ставить деякі обмеження, такі як:

- доступ до носія здійснюється послідовно: можна читати або записувати лише елемент, який слідує за поточним в кожний момент часу.
- обсяг даних не дозволяє їх розмістити в оперативній пам'яті.

Крім того, доступ до даних на носії є набагато повільнішим, ніж операції з оперативною пам'яттю. Зовнішнє сортування поділяється на два основні підкласи:

- **Внутрішнє сортування:** робиться з масивами, які повністю поміщаються в оперативній пам'яті з довільним доступом до будь-якого елемента. Дані зазвичай сортуються на місці, без додаткових витрат.
- **Зовнішнє сортування:** робиться з пристроями великого об'єму, які запам'ятовують, але з доступом не довільним, а послідовним (наприклад, сортування файлів). Тобто в будь-який момент ми можемо 'бачити' тільки один елемент, а витрати на перемотування є значно великими в порівнянні з пам'яттю. Це приводить до використання спеціальних методів сортування, які зазвичай використовують додатковий дисковий простір.

Для багатьох алгоритмів середній та найгірший час сортування  $n$ -елементного масиву зазвичай є  $O(n^2)$ , оскільки передбачаються перестановки елементів, розташованих поруч (різниця між їхніми індексами не перевищує певного заданого числа). Ці алгоритми часто є стабільними, хоча не завжди ефективними для обробки великих масивів.

Інший клас алгоритмів забезпечує сортування за час  $O(n \log n)$ , використовуючи можливість обміну елементів, розташованих на будь-якій відстані один від одного.

## Теорема про найкращий час сортування

Якщо алгоритм сортування в своїй роботі спирається тільки на операції порівняння двох об'єктів ( $\leq$ ) і не враховує жодної додаткової інформації про елементи, то він не може впорядкувати масив елементів швидше ніж за  $O(n \log n)$  в найгіршому випадку.

### Доведення

На кожному кроці алгоритм виконує одне порівняння, і його результат може приймати один із двох варіантів:

1.  $A \leq B$
2.  $A > B$

Залежно від результату порівняння алгоритм приймає подальші дії. Отже, уся робота алгоритму може бути представлена у вигляді бінарного дерева, де в листках розташовані можливі перестановки вхідного масиву.

Отже, кількість листків у дереві дорівнює  $n!$  листів, значить висота дерева є  $\log(n!)$ . Час роботи в найгіршому випадку пропорційний висоті дерева:

$$O(\log[(n!)]) = O\left(\log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right)\right) = O(n \log n)$$

Ці ефективні алгоритми застосовуються у практичних завданнях, але більшість з них не володіє стабільністю. Стабільні алгоритми, що працюють за час  $O(n \log n)$  потребують  $O(n)$  додаткової пам'яті.

Відомий стабільний алгоритм сортування, що не вимагає додаткової пам'яті працює за час  $O(n \log^2 n)$ .

Інший тип алгоритмів використовує додаткову інформацію про елементи, яку вони сортують (наприклад, унікальність чисел у певному діапазоні). Це дозволяє їм працювати більш ефективно за час  $O(n)$ .

**Відомі алгоритми сортування можна класифікувати за часом виконання:**

За час  $O(n^2)$

- сортування вибором
- сортування вставкою
- сортування обміном (бульбашки)

За час  $O(n \log n)$

- сортування купою
- швидке сортування
- сортування злиттям

За час  $O(n)$  з використанням додаткової інформації про елементи

- сортування підрахунком
- сортування за розрядами
- сортування комірками

За час  $O(n \log^2 n)$

- модифіковане сортування злиттям
- сортування Шелла

## 2.2. Порівняння алгоритмів сортування

Спробуємо порівняти ефективність методів сортування. Хай  $n$  як і раніше позначає число сортованих елементів, а  $C$  і  $M$  — відповідно кількість необхідних порівнянь ключів і пересилок елементів. Для всіх трьох простих методів сортування можна дати замкнуті аналітичні формули. Вони приведені в табл. 2.1. Заголовки стовпців Min, Max, Середин. визначають відповідно мінімуми, максимуми і очікувані середні значення для всіх  $n!$  перестановок  $n$  елементів.

Таблиця 2.1

**Порівняння простих методів сортування**

	Min	Середин	Max
сортування вставкою	$C = n - 1$ $M = 2(n - 1)$	$(n^2 + n - 2)/4$ $(n^2 - 9n - 10)/4$	$(n^2 - n)/2 - 1$ $(n^2 + 3n - 4)/2$
сортування вибором	$C = (n^2 - n)/2$ $M = 3(n - 1)$	$(n^2 - n)/2$ $n(\ln n + 0,57)$	$(n^2 - n)/2$ $n^2/4 + 3(n - 1)$

	Min	Середин	Max
сортування обміном (бульбашки)	$C = (n^2 - n)/2$ $M = 0$	$(n^2 - n)/2$ $(n^2 - n) * 0,75$	$(n^2 - n)/2$ $(n^2 - n) * 1,5$

Для вдосконалених методів немає досить простих і точних формул. Все, що можна сказати, — це що вартість обчислень дорівнює  $c \cdot n^{1.2}$  в разі сортування Шелла і  $c \cdot n \log n$  у випадках пірамідального і швидкого сортувань.

Ці формули дають лише приблизну оцінку ефективності як функції від  $n$ ; вони допускають класифікацію алгоритмів сортування на простих ( $n^2$ ) і вдосконалених, або «логарифмічні» ( $n \log n$ ). Проте для практичних цілей корисно мати деякі експериментальні дані, які можуть пролити світло на коефіцієнти  $c_i$ , що дозволяють проводити подальшу оцінку різних методів. Крім того, в цих формулах не враховуються витрати на інші операції, відмінні від порівнянь ключів і пересилок елементів, такі, як управління циклами і так далі. Зрозуміло, ці чинники якоюсь мірою залежать від конкретних систем, але проте деякий приклад експериментально отриманих даних є інформативним

У таблиці 2.2. приведений час (у мілісекундах), який витратила система Паскаль на обчислювальній машині CDC 6400 на виконання сортування описаними тут методами. У трьох стовпцях вказаний час, що було потрібно для сортування вже розсортованого масиву, випадкової перестановки і масиву із зворотним ладом елементів.

Таблиця 2.2

#### Час виконання програм сортування

	Впорядкований масив	Випадковий масив	Впорядкований в зворотному порядку масив
Просте включення	12 23	366 1444	704 2836
Бінарне включення	56 125	373 1327	662 2490

	Впорядкований масив	Випадковий масив	Впорядкований в зворотному порядку масив
Простий вибір	489 1907	509 1956	695 2675
Метод бульбашки	540 2165	1026 4054	1492 5931
Метод бульбашки з обмеженням	5 8	1104 4270	1645 6542
Шейкер-сортування	5 9	961 3642	1619 6520
Сортування Шелла	58 116	127 349	157 492
Пірамідальне сортування	116 253	110 241	104 226
Швидке сортування	31 69	60 146	37 79
Сортування злиттям	99 234	102 242	99 232

Ліве число в кожній колонці дане для масиву з 256 елементів, а праве — для 512 елементів. Ці дані демонструють явну відзнаку методів  $n^2$  від методів  $n \log n$ . Примітні наступні моменти:

1. Сортування бінарними включеннями не виявляє переваги в порівнянні зі сортуванням простими включеннями, особливо коли вже існує відповідний порядок.

2. Сортування методом бульбашки вважається найгіршим серед усіх порівнюваних методів. Навіть поліпшена його версія, яка використовує шейкер-сортування, залишається менш ефективною, ніж сортування простими включеннями і простим вибором.

3. Швидке сортування перевершує пірамідальне сортування відносно 2 до 3. Вона сортує масив з елементами, розташованими в зворотному ладі практично так само, як вже розсортований.

Слід додати, що ці дані були отримані при сортуванні елементів, що складаються тільки з ключа без супутньої інформації. Це — не дуже реалістичне допущення; у таблиці 2.3. показано, як впливає збільшення

розміру елементів на швидкість роботи програм. У вибраному прикладі супутні дані займають в 7 разів більше пам'яті, чим ключ.

Таблиця 2.3

**Час виконання програм сортування (ключі з інформацією)**

	<b>Впорядкований масив</b>	<b>Випадковий масив</b>	<b>Впорядкований в зворотньому порядку масив</b>
Просте включення	12 46	366 1129	704 2150
Бінарне включення	56 76	373 1105	662 2070
Простий вибір	489 547	509 607	695 1430
Метод бульбашки	540 610	1026 3212	1492 5599
Метод бульбашки з обмеженням	5 5	1104 3237	1645 5762
Шейкер-сортування	5 5	961 3071	1619 5757
Сортування Шелла	58 186	127 373	157 435
Пірамідальне сортування	116 264	110 246	104 227
Швидке сортування	31 55	60 137	37 75
Сортування злиттям	99 196	102 195	99 187

Ліве число в кожній колонці показує час, потрібний для сортування записів без супутніх даних, правий, — відображає сортування з супутніми даними;  $n$  — 256. Звернете увагу на наступні деталі:

1. Сортування простим вибором дає істотний виграш і виявляється кращим з простих методів.

2. Сортування методом бульбашки як і раніше є найгіршим методом (вона ще більше здала свої позиції), і лише її «удосконалення», зване шейкер-сортування, ще ледве гірше в разі масиву із зворотним порядком,

3. Швидке сортування навіть укріпило свою позицію як найшвидший метод і виявилось дійсно кращим алгоритмом сортування.

### Переваги та недоліки алгоритмів

Алгоритм	Переваги	Недоліки
Сортування Вибором	Дуже простий. Швидко сортує невеликі списки	Повільно працює з великими списками
Сортування вставками	Дуже простий. Швидко сортує невеликі списки	Дуже повільно працює з великими списками
Бульбашкове сортування	Швидко працює для майже відсортованих списків	Повільно працює в решті випадків
Швидке сортування	Швидко сортує великі списки	Працює некоректно при великій кількості однакових значень
Метод Шелла	Сортує дробові числа	Вимагає простору пам'яті для зберігання тимчасових значень
Сортування злиттям	Швидко сортує великі списки	Працює повільніше, ніж швидке сортування
Сортування підрахунком	Дуже швидко працює, якщо розкид вхідних значень не великий	Повільно працює у випадку якщо розкид складає $>\log(N)$
Сортування Шейкером	Сортує дані на жорсткому диску	Працює повільніше, ніж швидке сортування

### Порівняння часу сортувань

Змальований нижче графік ілюструє різницю в ефективності вивчених алгоритмів.

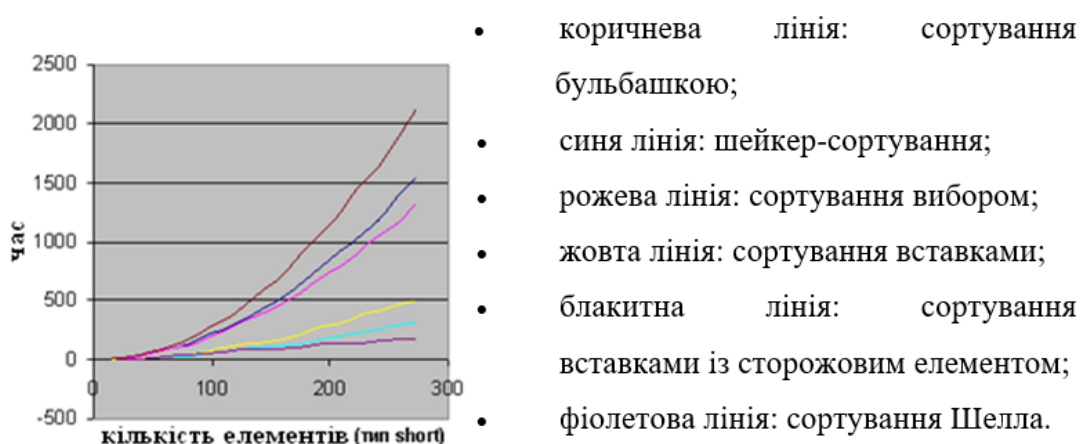


Рис. 2.3. Різницю ефективності алгоритмів



### **2.3. Висновки до розділу**

В данному розділі розглядається оцінка складеності алгоритмів сортування. Проводиться аналіз алгоритмів сортування.

Згідно вимог, прості алгоритми сортування (такі, як сортування вибором і сортування включенням) не є дуже ефективними.

Алгоритм сортування обмінами, хоча і завершує свою роботу (оскільки він використовує лише цикли з параметром і в тілі циклів параметри примусово не змінюються) і не використовує допоміжної пам'яті, але займає багато часу. Навіть, якщо внутрішній цикл не містить жодної перестановки, то дії будуть повторюватись до тих пір, поки не завершиться зовнішній цикл.

Алгоритм сортування вибором ефективніше сортування обмінами за критерієм  $M(n)$ , тобто за кількістю пересилань, але також є не дуже ефективним. З цих причин було розроблено деякі нові алгоритми сортування, що отримали назву швидких алгоритмів сортування. Це такі алгоритми, як сортування деревом, пірамідальне сортування, швидке сортування Хоара та метод цифрового сортування.

## **РОЗДІЛ 3. РЕАЛІЗАЦІЯ ДОДАТКУ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ СОРТУВАННЯ**

### **3.1. Вибір технології для побудови візуалізатора**

Візуалізатор алгоритму - це програма, написана в рамках певної технології. Як зазначалося вище, при використанні візуалізаторів на лекціях, а також при заочному навчанні можливе використання будь-яких мов програмування і відповідно, технологій. Обговорення переваг і недоліків різних підходів до побудови додатків виходять за рамки даної роботи. При побудові додатків, придатних для використання в дистанційному навчанні потрібне використання інтернет-орієнтованих технологій і мов.

При виборі технологічної платформи для створення візуалізаторів сформулюємо вимоги, яким повинна задовольняти ця платформа:

1. Наявність готових бібліотек з побудови користувальницького інтерфейсу.
2. Можливість вбудовувати додатки в Web-сторінки.
3. Крос-платформеність.
4. Можливість роботи без використання браузера.
5. Широке поширення і підтримка.

Існують різні технології для реалізації додатків в Інтернет. Прикладами можуть служити Adobe Flash [33], Microsoft Silverlight [34], JavaScript [35]. Безперечними достоїнствами цих технологій є їх широке розповсюдження і спрощене написання Інтернет-додатків. Їх недоліком є їх вузька сфера застосування - вони працюють тільки в рамках браузерів. Незважаючи на недавній вихід у світ технології Adobe AIR, що дозволяє запускати Flash-додатки без браузерів, сама технологія істотно поступається по гнучкості і поширеності технології Java. Технологія Java Applets, що є частиною технологічної платформи Java, надає максимальні переваги в порівнянні з іншими мовами і технологіями.

Наведемо порівняльну таблицю потенційних рішень при виборі технології побудови візуалізаторов (табл. 3.1.), В якій цифрами 1 - 5 позначені вимоги, зазначені вище.

Таблиця 3.1

### Результати порівняння платформ для розробки візуалізаторов

Платформа	1	2	3	4	5
Flash	+	+	+	-	+
Adobe AIR	+	+	+	+	-
Silverlight	+	+	-	-	-
Delphi	+	-	-	+	-
Java Applets	+	+	+	+	+
JavaScript	±	+	+	±	±

У дипломному проєкті для організації візуалізації алгоритмів сортування було використано скрипти на мові JavaScript (рис.3.1.).

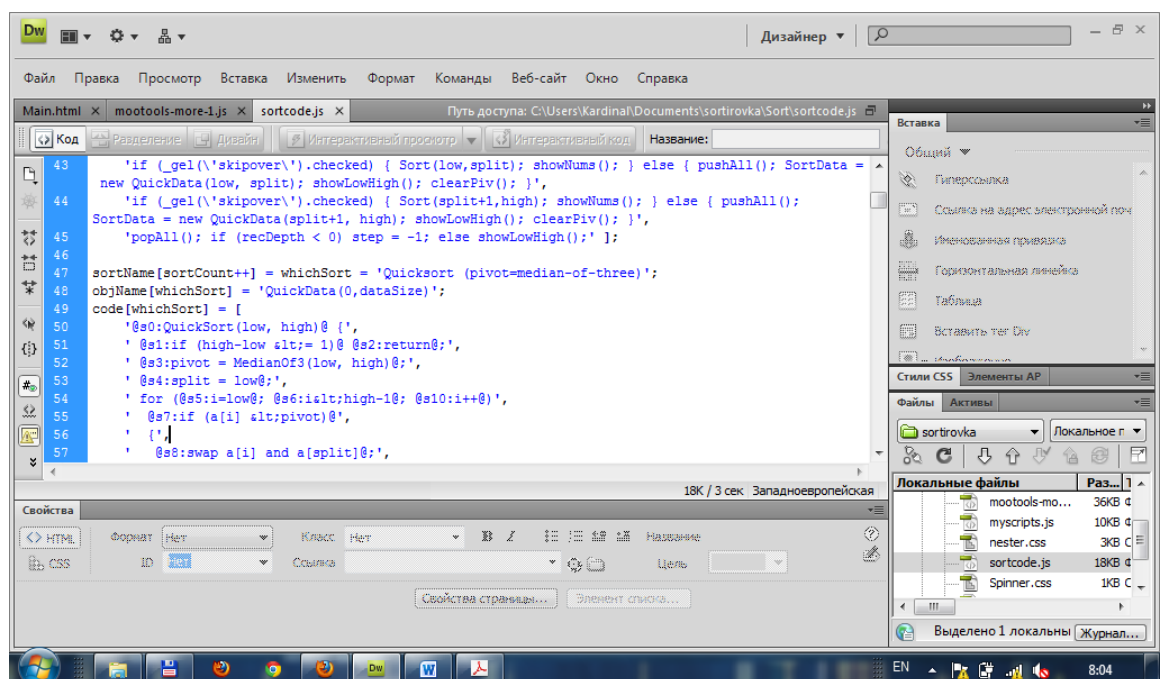


Рис. 3.1. Скрипти на мові JavaScript

Мову програмування JavaScript розроблено компанією Netscape для створення інтерактивних HTML-документів. Вона є об'єктно-орієнтованою мовою розробки вбудовуваних застосунків, які можуть виконуватися як на стороні клієнта, так і на стороні сервера. Синтаксис мови дуже схожий на синтаксис мови Java, і через це його часто порівнюють з java-подібним. Клієнтські застосунки виконуються браузером для перегляду web-

документів на машині користувача, серверні застосування виконуються на сервері. Обидва типи додатків, які розробляються, використовують загальний компонент мови, відомий як ядро. Це ядро включає визначення стандартних об'єктів і конструкцій, таких як змінні, функції, основні об'єкти і інструмент взаємодії з Java аплетами. Додаткові компоненти мови, які містять визначення об'єктів, специфічних для кожного типу додатків, додаються до цього ядра.

У клієнтських застосунках, вони вбудовуються безпосередньо в HTML-сторінки і інтерпретуються браузером, відображаючи частини документа у його вікні. Серверні додатки, які заздалегідь компілюються в проміжний байт-код, служать для підвищення продуктивності.

Основні області використання мови JavaScript при створенні інтерактивних HTML-сторінок включають:

- Динамічне створення документів за допомогою сценаріїв.
- Оперативна перевірка достовірності полів форми перед їх відправленням на сервер.
- Створення динамічних HTML-сторінок з CSS і об'єктною моделлю документа.
- Взаємодія з користувачем при вирішенні локальних завдань, що вирішуються додатком JavaScript, вбудованим у HTML-сторінку.

Мова JavaScript проста у використанні. Операції, які вона виконує, описані для зрозумілих об'єктів, таких як елементи робочої області, браузера і контейнери HTML. Об'єктно-орієнтованість мови виражається у використанні об'єктів з властивостями і методами над цими об'єктами.

На вершині ієрархії об'єктів знаходиться об'єкт window, що представляє вікно браузера і є батьківським для всіх інших об'єктів. Інші об'єкти у ієрархії можуть мати свої підлеглі об'єкти, і ця структура представлена в ієрархії об'єктів клієнта браузера.

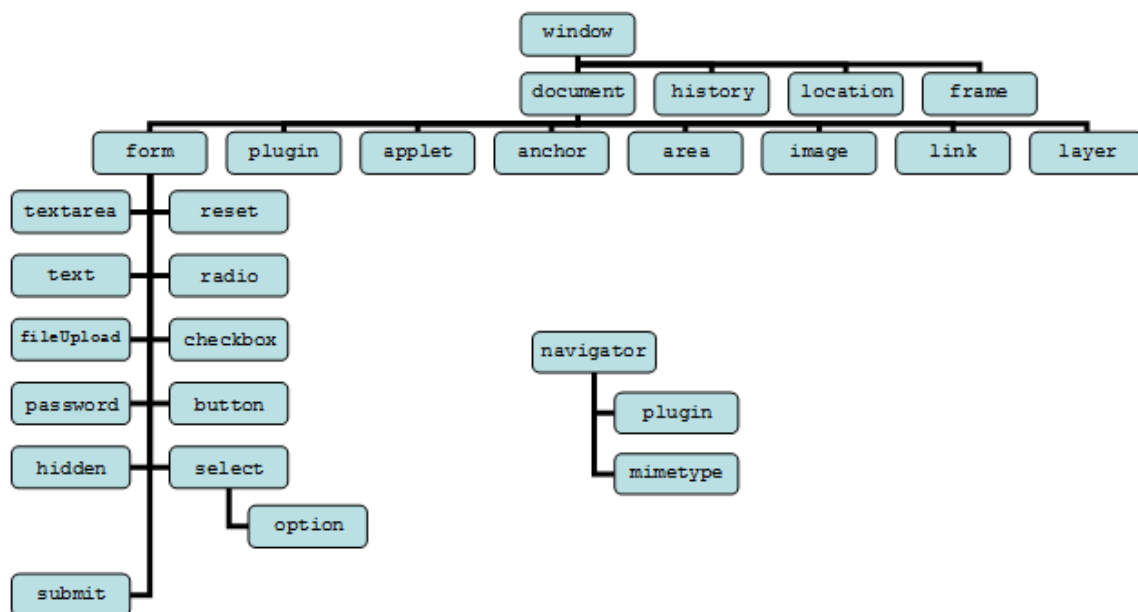


Рис.3.2. Ієрархія об'єктів Javascript на стороні клієнта

Поміж цих категорій об'єктів користувач може також створювати свої власні, але зазвичай програми використовують існуючу систему класів, не створюючи нових.

Окремо розташований об'єкт navigator має два підлеглі об'єкти. Цей об'єкт відноситься безпосередньо до браузера, і його властивості дозволяють визначити характеристики програми перегляду. Кожна сторінка, крім об'єкта navigator, обов'язково включає чотири інші об'єкти:

- window – об'єкт верхнього рівня, властивості якого застосовуються до всього вікна, в якому відображається документ.
- document – властивості якого визначають вміст самого документа, такі як зв'язки, колір фону, форми і інші.
- location – властивості якого пов'язані з url-адресою документа, що відображується.
- history – представляє адреси html-сторінок, що раніше завантажувалися.

Крім зазначених об'єктів, сторінка може включати додаткові об'єкти, які залежать від її вмісту і є дочірніми об'єктами об'єкту document. Наприклад, якщо на сторінці розташована форма, то всі її елементи стають дочірніми об'єктами цієї форми. Для точного вказівки імені об'єкту

використовується точкова нотація з повною вказівкою всього ланцюжка спадкоємства об'єкту. Це можливо, оскільки об'єкт верхнього рівня має властивість, значенням якої є об'єкт нижнього рівня. Посилання на об'єкт встановлюється за допомогою імені, яке визначається параметром NAME тега HTML.

Для створення візуалізації алгоритмів сортувань нам необхідні наступні компоненти:

- Прапорці;
- Списки;
- Картинки;
- Кнопки;
- Об'єкт textarea;
- Об'єкт text.

### **Прапорці**

Контрольний елемент "прапорець" використовується у випадку, коли потрібно вибрати один або кілька варіантів із запропонованих. Кожен варіант вибору представлений прапорцем, який можна встановити або скинути. Тип контрольного елемента "прапорець" визначається значенням параметра type в тезі `<input>`. Обов'язковим є також параметр value, який передається для обробки, якщо кнопка вибору натиснута.

### **Списки**

Якщо кількість елементів значно велика, використання прапорців або перемикачів може призвести до збільшення розміру форми. У такому випадку більш компактне представлення варіантів вибору може бути досягнуте за допомогою тега `<select>`. Важливо відзначити, що цей тег має різні параметри. Обов'язковим є параметр name. Для визначення кількості одночасно видимих елементів використовується параметр `size=n`. При значенні `n`, рівному 1, відображається спадне меню або список вибору, а при `n > 1` виводиться список з `n` одночасно видимими значеннями. У випадку, якщо параметр `size` не вказаний, йому автоматично присвоюється значення 1.

Параметр multiple вказує на те, що з меню або списку можна вибрати декілька елементів. Елементи меню задаються у середині тега <select> за допомогою тега <option>. Загальний вигляд тега такий:

```
<option selected value=строка>
```

Параметр selected означає, що даний елемент списку вважається вибраним за умовчанням. Параметр value містить значення, що передається, коли даний елемент вибраний із списку або меню.

### **Картинки**

Кнопки-картинки — це ті ж кнопки, але лише з можливістю відправки даних на сервер. Власне, такі кнопки в Javascript складають два різновиди контейнера INPUT: image і submit. У Javascript об'єкт, пов'язаний з даними кнопками, називається Submit.

```
<FORM>
```

Активна кнопка:

```
<INPUT Type=image Src=images.gif onclick="return false;">
```

```
</form>
```

Як ми вже відзначали, даний об'єкт володіє тими ж властивостями, методами і подіями, що і об'єкт Button. Але реакція в різних браузерях при обробці подій може бути різною. Так, в події onclick в Internet Explorer можна відмінити передачу даних на сервер, видавши як значення повернення false. Netscape Navigator на таку поведінку обробника події взагалі не реагує і відмінити передачу можна лише в атрибуті onsubmit контейнера FORM:

```
<FORM onsubmit="return false">
```

Активна кнопка:

```
<INPUT Type=image Src=images.gif border=0>
```

```
</form>
```

### **Кнопки**

Використання кнопок в Web взагалі немислимо без вживання Javascript. Кнопка вводиться у форму головним чином для того, щоб можна було обробити подію click:

```
<FORM>
<INPUT      Type=button      Value="Окно      запобігання"
onclick="window.alert("Открилі окно");">
</form>
```

Текст, що відображується на кнопці, визначається атрибутом VALUE контейнера INPUT. З цим атрибутом пов'язана властивість value об'єкту Button. Согласно специфікації, змінювати значення даного атрибуту не можна.

### **Об'єкт text**

Об'єкт text - це поле введення, визначуване в теге `<Input type="text">` і надаюче користувачеві можливість вводити текстові дані. Об'єкт text є властивістю об'єкту form і повинен розміщуватися в контейнері `<form> . . . </form>`. Об'єкти text містять дані, які можна і читати, і динамічно змінювати в Js-прграмах.

```
<input [type="text"]
name="textName"
value="textValue"
size=integer
[onBlur="handlerText"]
[onChange="handlerText"]
[onFocus="handlerText"]
[onSelect="handlerText"]>
```

### **Об'єкт textarea**

Об'єкт textarea відповідає області тексту, визначеній у формі. Об'єкти textarea є властивостями об'єкту form і мають бути поміщені в контейнер `<form> . . . </form>`. Елементи цього типа використовуються для введення декількох рядків тексту у вільному форматі. Також його часто використовують для виведення прикладів тексту наприклад js-програмі, сформірованнго тексту пропонованого для розміщення наприклад банера і ін.



Для звернення до методів і властивостей об'єкту `textarea` застосовуються типові для елементів форми вирази:

- `textareaName.propertyName`
- `textareaName.methodName(parameters)`
- `formName.elements[i].propertyName`
- `formName.elements[i].methodName(parameters)`

де `textareaname` - це значення атрибуту `name` тега `<textarea>`, а `formname` - ім'я форми, в котрій визначений об'єкт `textarea`. Вміст об'єктів `textarea` в js-програмах може динамічно змінюватися шляхом надання нового значення їх властивості `value`.

### **3.2. Вибір середовища розробки візуалізатора алгоритмів**

Провівши огляд програмних засобів, призначених для редагування і верстки HTML кодів, було з'ясовано, що кожен редактор має як свої переваги, так і недоліки. Зваживши всі за і проти, мій вибір зупинився на HTML редакторі Dreamweaver MX.

Dreamweaver – це сучасне програмне середовище для створення веб-сайтів. Інші програми дуже сильно уступають Dreamweaver по зручності, наочності і простоті в освоєнні, тому що, фірма-розробник Macromedia, постійно займається його удосконалюванням.

Система Dreamweaver представляє собою візуальний редактор гіпертекстових документів, в який інтегровано різні програмні інструменти та модулі, що охоплюють весь цикл операцій з розробки та підтримки віртуальних проектів. Як потужний професійний інструмент, Dreamweaver володіє всіма необхідними засобами для генерації сторінок HTML різної складності та розміру.

Програма надає можливість використання в режимі візуального проектування (WYSIWYG), що в перекладі з англійської означає "що бачите, те й отримуєте". Вона відрізняється чистою роботою з вихідним текстом веб-документів та вбудованими інструментами для підтримки обширних мережних проектів.

Це значить, що зображення сторінки HTML у вікні документа не сильно відрізняється від її представлення в найбільш популярних програмах перегляду — броузерах Microsoft Internet Explorer і Netscape Navigator.

Програма послідовно підтримує візуальне проектування, яке визначається як спосіб створення гіпертекстових документів, де робота з текстом і об'єктами зображень переважає над прямим кодуванням. У ідеалі, користувач повинен бути повністю вільним від необхідності вручного введення HTML-коду, а саме проектування має замінити програмування. Взаємодія з кодами залишається можливою, але максимально мінімізованою.

Програма не лише володіє потужним арсеналом інструментів для візуального проектування, але також може відображати веб-сторінки практично так само, як спеціалізовані програми перегляду, такі як Microsoft Internet Explorer, Netscape Navigator, Opera та інші.

В оболонку Dreamweaver інтегрований повнофункціональний редактор HTML, що володіє всіма необхідними інструментами для роботи з дескрипторами гіпертекстової розмітки.

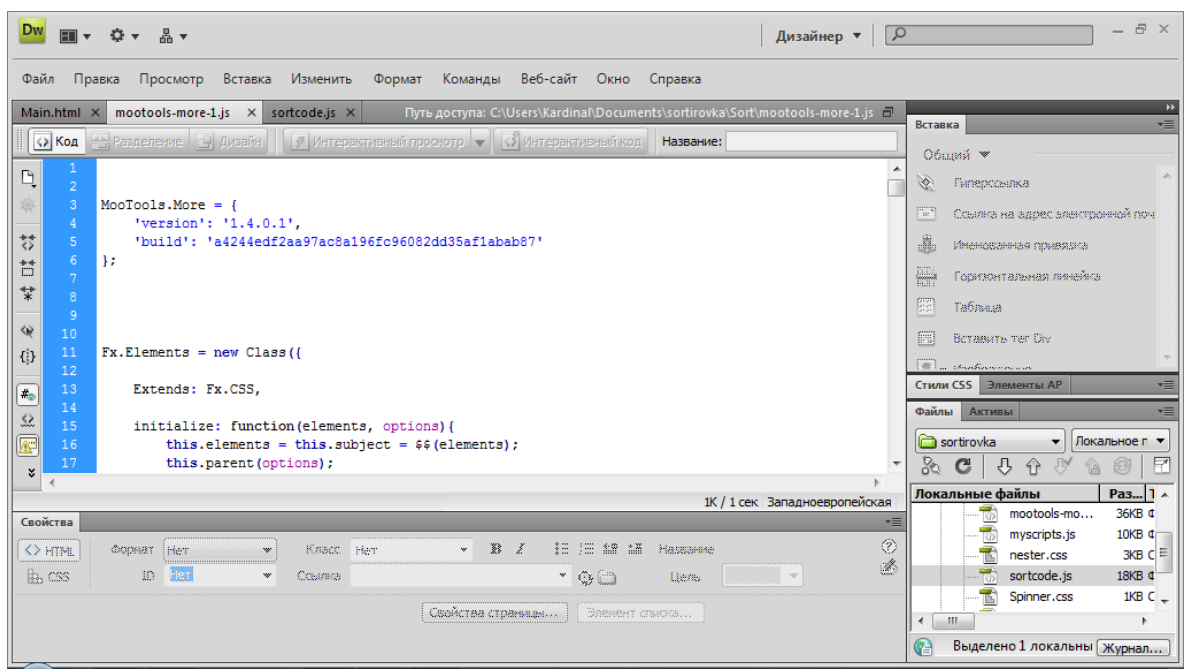


Рис. 3.3. HTML редактор Dreamweaver MX

Програма Dreamweaver базується на принципах відкритої архітектури. Це означає, що існує повністю відкритий інтерфейс для програмного

забезпечення (Application Programming Interface, API), який дозволяє стороннім розробникам вносити кардинальні зміни до функціональності програми та її інтерфейсу. Це включає можливість додавання нових інструментів, створення палітри чи меню, програмування нових об'єктів чи мультимедійних роликів та інше. Однак розробка розширеного мережевого проекту, як правило, не обмежується лише створенням компонентів гіпертекстових документів та їх структури.

В оболонку Dreamweaver інтегрований спеціальний засіб, призначений для обслуговування розповсюдження у мережі документів і сайтів. Він має всі необхідні інструменти для дистанційного відновлення версій документів. Цей засіб підтримує розподілену роботу декількох виконавців над одним мережним проектом, має механізм синхронізації версій документів і захисту від несанкціонованого доступу.

### **3.3. Каскадні таблиці стилів CSS**

Каскадні таблиці стилів, або просто CSS (Cascading Style Sheets), представляють собою набір правил, які визначають форматування різних частин HTML-коду і зберігаються окремо від нього. Кожне таке правило, що визначає форматування окремого фрагмента або групи фрагментів коду, називається стилем.

Таблиці стилів описуються мовою CSS і можуть бути збережені в спеціальних файлах з розширенням .css. Хоча їх можна вбудовувати безпосередньо в веб-сторінку, таблицю стилів, розміщену в окремому файлі, можна використовувати на кількох веб-сторінках. Зокрема, ця таблиця стилів може знаходитися навіть на іншому веб-сайті.

Web-сторінка може посилатися одночасно на кілька таблиць стилів. Наприклад:

```
<LINK REL="stylesheet" HREF="styles1.css">
```

```
<LINK REL="stylesheet" HREF="styles2.css">
```

за допомогою таблиці стилів можна змінити зовнішній вигляд будь-якого тегу HTML. Для досягнення цього, достатньо просто визначити його заново в таблиці стилів, використовуючи такий підхід:

```
H1 | { color: #FF0000;  
font-size: smaller }
```

За допомогою таблиць стилів можна формувати не лише текст. Любому елементу сторінки - будь то зображення, таблиця чи горизонтальна лінія - може бути присвоєно стильовий клас.

Стандарт CSS визначає три методи визначення стилів для елемента сторінки:

- Зовнішня (або прив'язана) таблиця стилів, де стилі знаходяться в окремому файлі з розширенням css і приєднуються до Web-сторінки за допомогою тегу <LINK>.
- Внутрішня (або впроваджена) таблиця стилів, яка має той же формат, що і зовнішня, але розташовується в секції заголовка тієї ж Web-сторінки і міститься усередині тегу <STYLE>.
- Внутрішні (також вбудовані або впроваджені) стилі, де визначення стилю міститься безпосередньо в потрібному тезі і використовує спеціальний атрибут STYLE.

Вся робота зі стилями виконується в панелі CSS Styles Dreamweaver: пункт меню Window, пункт CSS Styles.

Для створення нового стилю, оберіть пункт New CSS Style контекстного або додаткового меню.

### **Недоліки CSS**

Нажаль не всі веб-браузери коректно підтримують каскадні таблиці стилів, тому що це новий стандарт (1997 р.). У деяких випадках необхідно переконвертувати сторінку у вікні браузера.

У диплому проєкті були використані каскадні таблиці стилів для оформлення сторінки візуалізації алгоритмів.

### **3.4. Опис інтерфейсу Web-модуля візуалізації алгоритмів сортування**

Дуже важливу роль в ефективності роботи програми грає правильно розроблений інтерфейс. Саме від цього буде залежати виконання декількох з основних вимог - зручність і простота в освоєнні. Складний, перевантажений інтерфейс може зменшити ефективність роботи. Головне правило, від якого варто відштовхуватися - інтерфейс не повинен заважати роботі користувача й не повинен відволікати його увагу від роботи, одночасно не втрачаючи при цьому у функціональності.

Виходячи із цього, було вирішено використовувати типові для windows-програм візуальні компоненти. Це дасть можливість більшості користувачів швидко розібратися й включитися в роботу. Інтерфейс прямо залежить від функцій, що виконуються програмою.

Список функцій програми.

Можливість сортування числових та символьних масивів наступними методами:

1. Швидке сортування (Quicksort (pivot=last));
2. Швидке сортування (Quicksort (pivot=median-of-three));
3. Метод бульбашки (Bubble Sort);
4. Метод бульбашки (Bubble Sort (smart));
5. Гном сортування (Gnome Sort);
6. Сортування Шейкером (Shaker Sort);
7. Комбінований (CombSort11);
8. Сортування вставками (Insertion Sort);
9. Сортування Шелла (Shell Sort);
10. Бінарною купою (Heap Sort);
11. Сортування злиттям (Merge Sort);
12. Сортування за розрядами (Radix Sort).

Основу інтерфейсу повинні становити компоненти представляючі користувачеві саму необхідну інформацію й доступ до найважливіших функцій системи.

Впливаючи із цього, головне вікно програми повинне містити наступні основні компоненти:

1. Список алгоритмів сортування для вибору;
2. Методи сортування;
3. Розмірність масиву;
4. Затримка в ms;
5. Покрокове виконання;
6. Автозапуск;
7. Перезапуск;
8. Колекція даних;
9. Відображення алгоритму сортування;
10. Відображення даних сортування.

Схема інтерфейсу програми-клієнта показана на рис. 3.4.

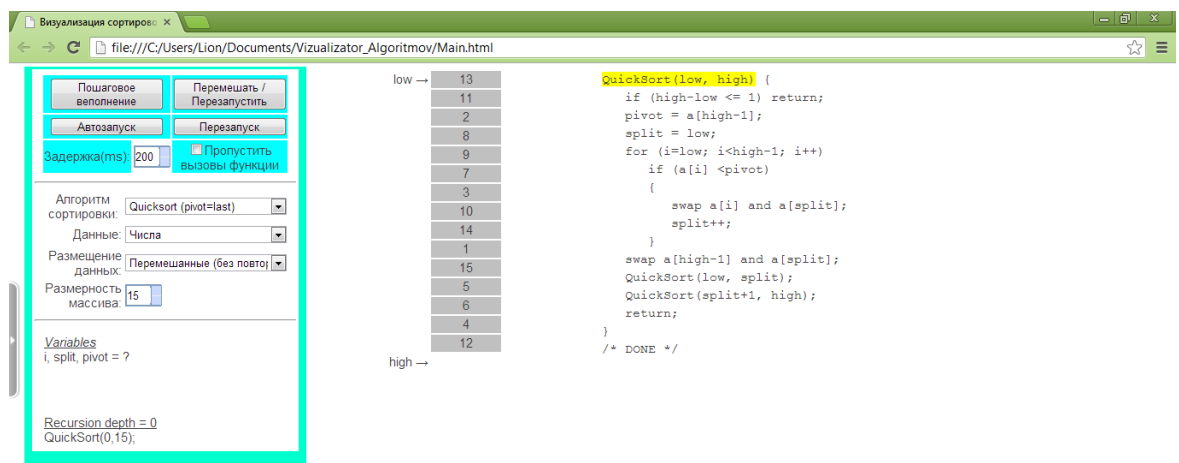


Рис. 3.4. Загальна концепція інтерфейсу візуалізації алгоритмів сортування

Слід зазначити, що вся концепція інтерфейсу виконана в простому стилі й у цілому типова для програм такого роду.

### 3.5. Методичні вказівки по використанню Web-модуля візуалізації алгоритмів сортування

Після запуску Main.html з'являється сторінка на якій є можливість проводити сортування масивів даних та відображати їх демонстрацію.

Користувач може визначити, які алгоритми слід виконати під час демонстрації, і обрати для них відповідні дані. (рис. 3.5.-3.6).

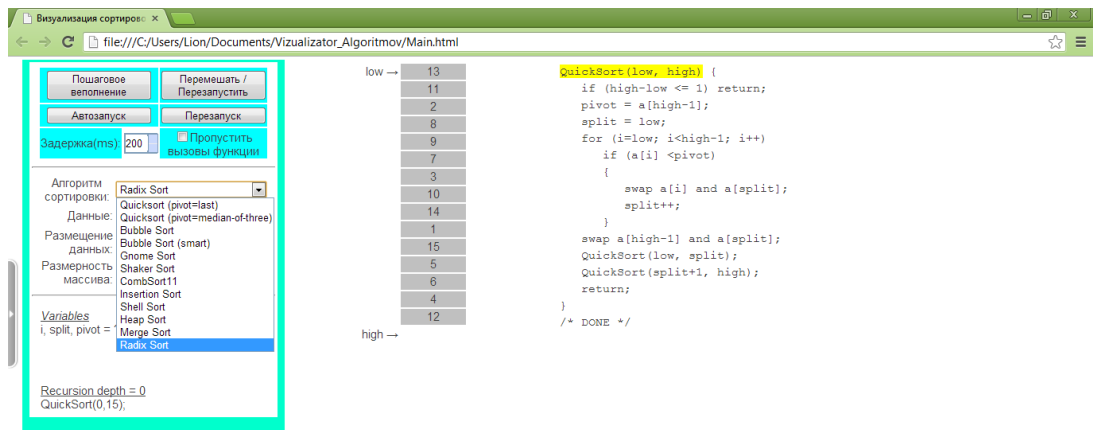


Рис. 3.5. Вибір алгоритму сортування

Потрібна колекція даних, з якої користувач може вибрати масиви, для оцінки ефективності різних алгоритмів на ідентичних даних та формулювання висновків про ефективність виконання того самого алгоритму на різних наборах даних. Завантажений алгоритм відображається у правій частині вікна.

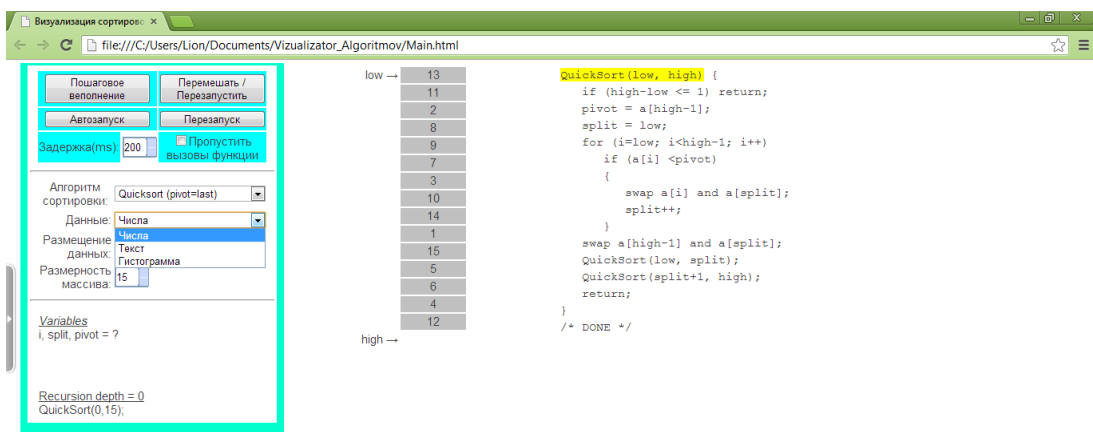


Рис. 3.6. Вибір даних що підлягають сортуванню

Колекція даних може бути числовою, символьною або гістограма (рис. 3.6.-3.8).

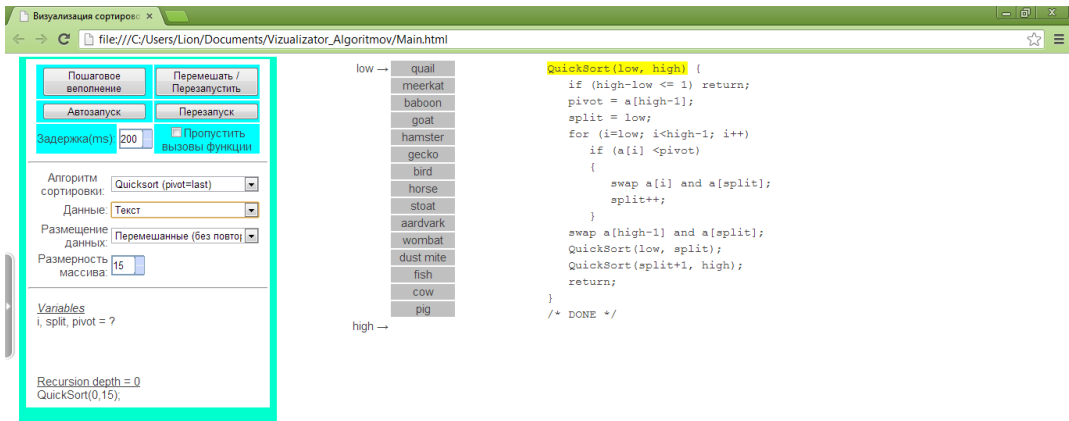


Рис. 3.7. Вибір символного масиву даних

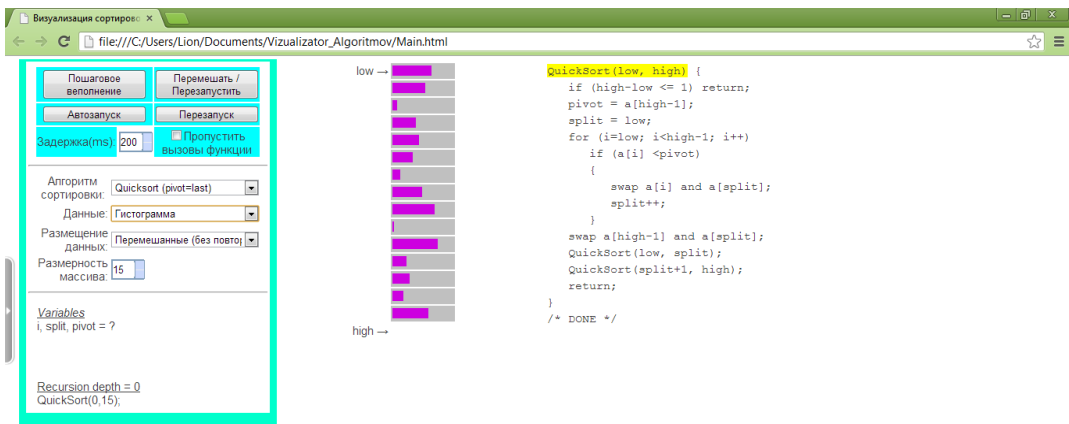


Рис. 3.8. Вибір гістограмного масиву даних

Наступний крок — заповнення контейнерів даними. Для цього необхідно в розкриваю чому списку «Розміщення даних» (рис. 3.9), що формує дані для візуалізації виконання алгоритму у середовищі демонстрації одним із способів:

- за зростанням;
- за спаданням;
- випадковим чином (з перемішуванням);
- випадковим чином (без перемішування);
- константами.



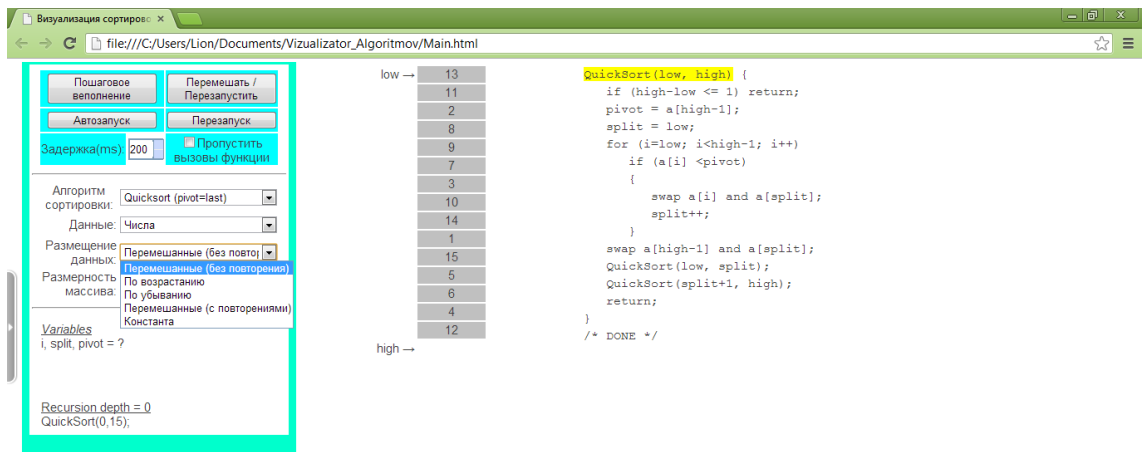


Рис. 3.9. Формування даних для візуалізації виконання алгоритму  
Також треба установити розмір масиву (максимум 20).

Користувач може вибрати режим роботи: "Неперервний" або "Покроковий". При виборі «Неперервний тип роботи» за допомогою кнопки «Автозапуск» можна виконувати візуалізацію алгоритму автоматично.

Якщо обрати перемикач «Пропустити визов функцій», то будуть пропускатися функції внутрі алгоритму.

При встановленні режиму "Покрокового виконання", використовуючи кнопку "Покрокове виконання", можна крок за кроком візуалізувати виконання кожного окремого етапу алгоритму (рис. 3.10.).

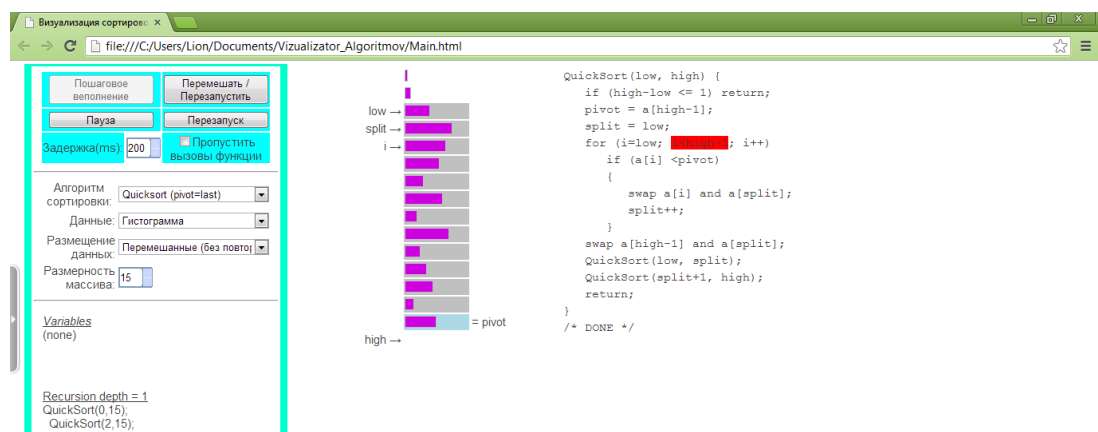


Рис. 3.10. Покрокове виконання алгоритму

### 3.6. Висновки до розділу

В розділі сформульовані функціональні вимоги до розроблювальної програми, серед яких основними є наступні:

1. Можливість сортування числових та символьних масивів наступними методами:
  - Швидке сортування (Quicksort (pivot=last));
  - Швидке сортування (Quicksort (pivot=median-of-three));
  - Метод бульбашки (Bubble Sort);
  - Метод бульбашки (Bubble Sort (smart));
  - Гном сортування (Gnome Sort);
  - Сортування Шейкером (Shaker Sort);
  - Комбінований (CombSort11);
  - Сортування вставками (Insertion Sort);
  - Сортування Шелла (Shell Sort);
  - Бінарною купою (Heap Sort);
  - Сортування злиттям (Merge Sort);
  - Сортування за розрядами (Radix Sort).
2. Можливість демонструвати наочний матеріал;
3. Легкість в освоєнні й роботі з програмою.

Був складений загальний алгоритм роботи програми та розроблен інтерфейс, на основі передбачуваної функціональності був обраний програмний засіб для реалізації поставленої мети в роботі.

Мовою програмування було обрано JavaScript і спеціалізоване середовище Dreamweaver MX.

## ВИСНОВКИ

У магістерській роботі було розглянуто аналіз та розробку додатку візуалізації алгоритмів.

Зручність використання має велике значення при самостійному навчанні, оскільки користувачі рідко звертаються до інструкцій. Можливості відображення етапів алгоритму та коментування кроків програми є ключовими. Перше важливе для пояснення простих алгоритмів і введення в інформатику, а друге — для зрозуміння складних алгоритмів. Доступність, платформонезалежність і незалежність від мережі дозволяють використовувати візуалізатор як на лекціях та практичних заняттях, так і для самостійного навчання.

Використання візуалізаторів для формування алгоритмічних навичок сприяє розвитку пізнавальних здібностей здобувачів вищої освіти, їхньої здатності аналізувати та інтерпретувати результати та підштовхує до дослідницької діяльності. Таким чином, формуються нові якісні професійно значущі вміння та навички, що допомагають підготувати майбутніх випускників до успішної професійної діяльності.

Для досягнення поставленої мети були вирішені наступні завдання:

- Розглянуто застосування візуалізаторів в навчальному процесі.
- Проведено аналіз та порівняння алгоритмів сортування.
- Визначено зміст, структура, функції і дидактичні можливості додатку для реалізації візуалізації алгоритмів сортування.
- На основі проведеного аналізу інструментальних засобів та теоретичного матеріалу по алгоритмам сортування інформації розроблено і реалізовано діючий Web-модуль для візуалізації алгоритмів сортування, який візуалізує поширені методи сортування. Як програмне середовище було обрано мову програмування JavaScript і візуальне середовище Dreamweaver MX. До програми були розроблені методичні рекомендації з використання.

Практичну цінність своєї роботи бачу в тім, що мною був отриманий багатий досвід в реалізації алгоритмів сортування та удосконалені знання по роботі з JavaScript в спеціалізованому середовищі Dreamweaver MX, в якому розроблено Web-модуля для візуалізації алгоритмів сортування, який можливо використовувати в навчальному процесі.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Казаков М. А., Шалыто А. А. Автоматный подход к реализации анимации в визуализаторах алгоритмов // Компьютерные инструменты в образовании. 2005. № 3.
2. Казаков М.А. Визуализаторы алгоритмов как элемент технологии преподавания дискретной математики и программирования./ М.А. Казаков, С.Е. Столяр // Тезисы докладов международной научно-методической конференции "Телематика-2000". СПб.: СПбГИТМО (ТУ), 2000.
3. Кнут Д. Искусство программирования. Том 1. Основные алгоритмы. / Д. Кнут - М.: Вильямс, 2000.
4. Коменский Я.А. Избранные педагогические сочинения: в 2 т. / Я.А.Коменский; под ред. А.И.Пискунова. – М: Педагогика, 1982. – Т. 1. – 656 с.
5. Кормен Т. Алгоритмы. Построение и анализ. / Т. Кормен, Ч. Лейзерсон, Р. Ривест - М.: МЦНМО, 1999.
6. Львов М.С. ПМК «Відеоінтерпретатор алгоритмів пошуку та сортування» /М.С.Львов, О.В.Співаковський // Інформатизація освіти України: стан, проблеми, перспективи: зб. наук. праць / ред. О.В.Співаковський. – ХДУ: Херсон, 2003.– С. 100-102.
7. Манько Н.Н. Когнитивная визуализация дидактических объектов: Монография. –Уфа:Изд-во БГПУ, 2007. – 180 с.ISSN 5-87978-364-2.
8. Основи охорони праці: Підручник. / За ред. К.Н Ткачука і М.О. Халімовського. - К.: Основа, 2006.- 448 с.
9. Павелко В. Теоретичний аспект наочного навчання на різних етапах його розвитку /В. Павелко // Науково-методичний журнал “Нова педагогічна думка”, № 4, 2009.
10. Песталоцци Й.Г. Избранные педагогические сочинения: в 3 т. / под ред. М.Ф.Шабасовой. – М.: Акад. пед. наук РСФСР, 1963. – Т. 2: 1791-1804. – 1963. – 563 с.

- 11.Столяр С. Е., Осипова Т. Г., Крылов И. П., Столяр С. С. Об инструментальных средствах для курса информатики // II Всероссийская конференция “Компьютеры в образовании”. СПб.: 1994.
- 12.Український педагогічний словник / [укл. С.Гончаренко]. – К.: Либідь, 1997. – 373 с.
- 13.Шалыто А. А. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
- 14.Визуализатор пирамидальной сортировки / <http://is.ifmo.ru/works/heapsort/>
- 15.Brown M., Sedgewick R. A system for Algorithm Animation / Computer Graphics, Proceedings of the 11th annual conference on Computer graphics and interactive techniques, July 1984.
- 16.Demetrescu C., Finocchi I., Stasko J. Specifying Algorithm Visualizations: Interesting Events or State Mapping? // Proceedings of the International Dagstuhl Seminar on Software Visualization, Schloss Dagstuhl, May 2001, appears in Software Visualization State-of-the-Art Survey, LNCS 2269, Stephan Diehl (ed.), Springer Verlag, 2002.
- 17.Hope College animator page / <http://www.cs.hope.edu/~algaanim/animator/Animator.html>.
- 18.Jeliot Home Page / <http://cs.joensuu.fi/jeliot/>.
- 19.Lawrence A., Badre A., Stassko J. Empirically evaluating the use of animations to teach algorithms // Technical report, Graphics Visualization and Usability Center, Georgia Institute of Technology, 1993.
- 20.Princeton University animators collection [http://www.cs.princeton.edu/~ah/alg\\_anim/version1/Animator.html](http://www.cs.princeton.edu/~ah/alg_anim/version1/Animator.html).
- 21.State University of New York College at Brockport sorting animators collection / <http://www.cs.brockport.edu/cs/Javasort.html>.

- 22. Adobe Flash. <http://www.adobe.com/flashplatform/>
- 23. Microsoft Silverlight. <http://www.silverlight.net/>
- 24. Флэнаган Д. JavaScript. Подробное руководство. СПб.: Символ-Плюс, 2008. – 992 с.

## ДОДАТОК А

### Лістинг Main.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Визуализация сортировок</title>
    <script type="text/javascript" src="sort/mootools-core-1.js"></script>
    <script type="text/javascript" src="sort/mootools-more-1.js"></script>
    <script type="text/javascript" src="sort/myscripts.js"></script>
    <script type="text/javascript" src="sort/Spinner.js"></script>
    <script type="text/javascript" src="sort/sortcode.js"></script>
    <link rel="stylesheet" type="text/css" href="sort/nester.css">
    <link rel="stylesheet" type="text/css" href="sort/Spinner.css">

</head>
<body>
<script type="text/javascript">
var pending = 0;
var swapped1;
var swapped2;
var auto = false;
var autoDelay = 200;
var autoStep;
var arraySize = 20;
var dataSize = 15;
var scaleData;
var randType = 1;
var recDepth;
var stack = [];
var lbls = [];
var undefText;
var varText;
var dataType = 'numbers';
var SortData;
var animals = ['aardvark','armadillo','baboon','beaver','bird',
               'cat','cow','dog','dust mite','emu',
               'fish','gazelle','gecko','gnu','goat',
               'goose','hamster','hedgehog','horse','lemur',
               'meerkat','octopus','pig','puma','quail',
               'skunk','stoat','weasel','wombat','zebra' ];

var a = [];
window.addEventListener('domready', function() {
    initSlides();
    hideAll();
    initAll();
    $('delay').value = autoDelay;
    $('dataSize').value = dataSize;
    standardPage("<a href=home.html> Java[Script]</a>: Sorting Demonstrations");
});
```



```

function _elAddText(thing, t) { _gel(thing).innerHTML += t; }

function _elColor(thing, c) { _gel(thing).style.backgroundColor = c; }

function textObj(t,c,a) {
  this.text = t;
  this.color = (typeof(c) == 'undefined' ? '' : c);
  this.align = (typeof(a) == 'undefined' ? '' : a);
}
function numSet(k, o) { elSet('n', k, o); }

function numColor(k, c) { numSet(k, new textObj("", c)); }

function scratchColor(k, c) { _elColor('piv'+k, c); }

function elSet(what, k, o) {
  if (typeof(o) == 'string') o = new textObj(o);
  if (o.text != "") _elText(what+k, o.text);
  if (o.color != "") _elColor(what+k, o.color);
  if (o.align != "") _gel(what + k).style.textAlign = o.align;
}

function spaces(n) {
  var s = "";
  for (var j=0; j<n; j++)
    s += '&nbsp;';
  return s;
}

function swap(i,j) {
  var t = a[i];
  a[i] = a[j];
  a[j] = t;

  numColor(swapped1 = i, 'pink');
  numColor(swapped2 = j, 'pink');
  pending = setTimeout('resetColor();', 1000);
}

function resetColor() {
  numColor(swapped1, 'silver');
  numColor(swapped2, 'silver');
  showNums();
  clearTimeout(pending);
  pending = 0;
}

function initSort(which) {
  var tbl = '<table class="code">';
  var j,k,s;
  if (auto) autoRun();
  whichSort = which;

```

```

dataType = _gel('dataType').options[_gel('dataType').selectedIndex].value;

for (j=0; j<code[whichSort].length; j++) {
    s = code[whichSort][j];
    while (s.indexOf('@') >= 0) {
        s = s.replace(/@/, '<span id="');
        s = s.replace(/:/, '">');
        s = s.replace(/@/, '</span>');
    }
    for (k=0; s.charAt(0) == ' '; k++)
        s = s.slice(1);
    s = spaces(3*k) + s;
    tbl += '<tr><td>' + s + '</tr></td>';
}
tbl += '<tr><td><span id="s-1">/* DONE */</span></tr></td>';
_elText('code', tbl + '</table>');
SortData = false;
newSort();
showLbIs();
}

function newSort() {
    if (auto) autoRun();
    if (SortData) _elColor("s" + SortData.step, 'white');
    recDepth = 0;
    SortData = eval('new ' + objName[whichSort]);
    _elColor("s0", 'yellow');
    showLowHigh();
    clearPiv();
    _elText('rec', "");
}

function clearPiv() {
    for (var j=0; j<arraySize+1; j++)
        elSet('piv', j, new textObj(' ', 'white'));
}

function initAll() {
    changeMethod(_gel('method'));
    changeData(_gel('dataType'));
    changeRand(_gel('randType'));
    setSize();
    setDelay();
}

function setSize() {
    dataSize = _gel('dataSize').value * 1;
    fixMax();
    MixUp();
}

```

```

}

function setDelay() {
  autoDelay = _gel('delay').value * 1;
}

function changeMethod(e) {
  initSort(e.options[e.selectedIndex].value);
  showNums();
}

function changeData(e) {
  dataType = e.options[e.selectedIndex].value;
  showNums();
  showLbIs();
  showScratch();
  showBucket();
}

function changeRand(e) {
  clearTimeout(autoStep);
  clearTimeout(pending);
  randType = 1*e.options[e.selectedIndex].value;
  fixMax();
  MixUp();
}

function fixMax() {
  scaleData = ((randType <= 3) ? animals.length / dataSize : 1);
}

function MixUp() {
  var j, k, t;
  switch (randType) {
    case 1:// 1 = permutation of 1..dataSize
      for (j=0; j<dataSize; j++)
        a[j] = j+1;
      for (j=dataSize-1; j>0; j--) {
        k = Math.floor(Math.random()*j);
        t = a[k];
        a[k] = a[j];
        a[j] = t;
      }
      break;

    case 2:// 2 = increasing (sorted)
      for (j=0; j<dataSize; j++)
        a[j] = j+1;
      break;

    case 3:// 3 = decreasing (reversed)

```

```

        for (j=0; j<dataSize; j++)
            a[j] = dataSize-j;
    break;

    case 4:// 4 = random sample of 1..maxSize (repeats allowed)
        for (j=0; j<dataSize; j++)
            a[j] = Math.ceil(Math.random()*animals.length);
    break;

    case 5:// 5 = all the same
        t = Math.ceil(Math.random()*animals.length);
        for (j=0; j<dataSize; j++)
            a[j] = t;
    break;

    default:
        alert('Hmm ... Mixup called with randType = ' + randType);

}
newSort();
_elColor("s0", 'yellow');
showNums();
showLbIs();
showLowHigh();
}

function addLabel(varName) {
    var t;
    with (SortData) {t = eval(varName)};
    if (typeof(t) != 'undefined')
        lIs[t]= varName + (lIs[t]==" ? ' &rarr; ' : ', '+lIs[t]);
    else
        showVar(varName);
}

function showVar(varName) {
    var t;
    with (SortData) {t = eval(varName) };

    if (typeof(t) != 'undefined') {
        if (varName == 'temp') t = dataRep(t).text;
        varText = varName + ' = ' + t + (varText != " ? '<br>' : ") + varText;
    } else
        undefText = varName + (undefText != " ? ', ' : '= ?') + undefText;
}

function showLIs() {
    undefText = varText = "";
    for (var j=0; j<arraySize+1; j++) {
        lIs[j] = "";
    }
}

```

```

}

switch (whichSort) {
    case 'Quicksort (pivot=median-of-three)':
        if (SortData.subFlag) {

            addLabel('middle');
        }
        addLabel('low');
        addLabel('high');
        break;
    case 'Quicksort (pivot=last)':
        if (SortData.pivot)
            elSet('piv', SortData.high-1, '= pivot');
        else
            showVar('pivot');
            addLabel('low');
            addLabel('high');
            addLabel('split');
        addLabel('i');
        break;

    case 'Bubble Sort (smart)':
        showVar('swaps');
        case 'Bubble Sort':
        case 'Gnome Sort':
            addLabel('i');
        addLabel('j');
        break;
        case 'Shaker Sort':
            addLabel('i');
        addLabel('j');
        addLabel('k');
        addLabel('min');
        addLabel('max');
        break;
        case 'CombSort11':
            addLabel('i+gap');
            undefText = "";
        addLabel('i');
            showVar('swaps');
            showVar('gap');
        break;
        case 'Insertion Sort':
            addLabel('i');
            addLabel('j');
            showVar('temp');
        break;
        case 'Shell Sort':
            addLabel('j+inc');
            undefText = "";
        addLabel('i');

```

```

        addLabel('j');
        showVar('inc');
    break;
    case 'Heap Sort':
        if (SortData.subFlag) {
            addLabel('parent');
        }
        addLabel('child');
        showVar('temp');
        showVar('len');
    } else {
        addLabel('i');
    }
    break;
    case 'Merge Sort':
        addLabel('low');
        addLabel('high');
        showVar('m2');
        showVar('m1');
        addLabel('i');
        addLabel('split');

    break;
    case 'Radix Sort':
        showVar('allZero');
        showVar('i');
        addLabel('j');
        addLabel('n');
        showVar('tmp');
        showVar('m');

        break;
    default:
        alert('Bad sort type: ' + whichSort);
}

for (var j=0; j<arraySize+1; j++)
    elSet('lbl', j, (lbls[j]=='' ? '' : lbls[j]));
varText += (undefText != '' && varText != '' ? '<br/>' : '') + undefText ;
    _elText('vars', varText == '' ? '(none)' : varText);
}

function showScratch() {
    if (whichSort == 'Merge Sort') {
        if (SortData.scratchFlag) {
            for (var j=0; j<SortData.scratch.length; j++) {
                var t = dataRep(SortData.scratch[j]);
                t.color = (j == SortData.m1 ? 'lightblue' : 'silver');
                elSet('piv', j+SortData.low, t);
            }

            if (SortData.m2 && SortData.m2 < SortData.high)

```

```

        numColor(SortData.m2,'lightblue');
    } else {
        for (var j=0; j<SortData.scratch.length; j++)
            elSet('piv', j+SortData.low, new textObj(' ', 'white'));
    }
}
}

function showBucket() {
    var t;
    if (whichSort == 'Radix Sort') {
        for (var j=0; j<SortData.bucket.length; j++) {
            if (j < SortData.bucketsize) {
                t = dataRep(SortData.bucket[j]);
                t.color = (j == SortData.m ? 'lightblue' : 'silver');
            } else {
                t = new textObj(' ', 'white');
            }
            elSet('piv', j, t);
        }
    }
}

function dataRep(i) {
    var t = new textObj();
    switch (dataType) {
        case 'numbers':
            t.align = 'center';
            t.text = "+ i; break;
        case 'words':
            t.align = 'center';
            t.text = animals[Math.floor((i-1)*scaleData)]; break;
        case 'bars':
            t.align = 'left';
            t.text = ''; break;
    }
    if (whichSort=='Radix Sort' && dataType != 'numbers') {
        t.align = 'left';
        t.text = ('0' + i + ':').slice(-3) + t.text;
    }
    return t;
}

function showNums() {
    var j;
    for (j=0; j<dataSize; j++)
        elSet('n', j, dataRep(a[j]));
    for (; j<arraySize; j++)
        elSet('n', j, ' ');
}

```

```

function showLowHigh() {
    for (var j=0; j<arraySize; j++)
        numColor(j, (j>=SortData.low && j<SortData.high ? 'silver' : 'white'));
    if (whichSort.slice(0,9) == 'Quicksort' || whichSort == 'Merge Sort') {
        var t = '<u>Recursion depth = ' + recDepth + '</u><br/>';
        for (var j=0; j<recDepth; j++)
            t += spaces(2*j) + stack[j] + '<br/>';
        t += spaces(2*j) + SortData;
        _elText('rec', t);
    }
}

function pushAll() {
    stack[recDepth++] = SortData;
}

function popAll() {
    recDepth--;
    if (recDepth>=0) SortData = stack[recDepth];
}

function autoRun() {
    clearTimeout(autoStep);
    auto = !auto;
    if (auto) {
        _gel('nextStepButton').disabled = true;
        _elText('autoRunButton', 'Пайза');
        SortSteps();
    } else {
        _gel('nextStepButton').disabled = false;
        _elText('autoRunButton', 'Автозапуск');
    }
}

function SortSteps() {
    if (pending != 0) resetColor();
    with(SortData) {
        if (step<0) {if (auto) autoRun(); return; }
        _elColor("s" + step, 'white');
        eval(commands[whichSort][step++]);
    }
    showLbIs();
    _elColor("s" + SortData.step, 'red');
    setDelay();
    if (auto) autoStep = setTimeout('SortSteps()', autoDelay);
}

</script>

```



```

<table>
<tbody><tr valign="top"><td height="404" bgcolor="#00FFCC">
<table>
<tbody><tr><td width="300">
<table>
<tbody><tr align="center"><td colspan="2" bgcolor="#00FFFF">
<button
id="nextStepButton"
onclick="javascript:SortSteps();"
style="width:35mm">Пошаговое выполнение</button>
</td><td bgcolor="#00FFFF">
<button id="scrambleButton" onclick="javascript:setSize();" style="width:35mm">Перемешать
/ Перезапустить</button>
</td></tr>
<tr align="center"><td colspan="2" bgcolor="#00FFFF">
<button
id="autoRunButton"
onclick="javascript:autoRun();"
style="width:35mm">Автозапуск</button>
</td><td bgcolor="#00FFFF">
<button id="resetButton" onclick="javascript:newSort();" style="width:35mm">Перезапуск
</button>
</td></tr>
<tr align="center">
<td
bgcolor="#00FFFF"><label
style="cursor:
w-resize;"
id="delayLabel">Задержка(ms):</label></td>
<td><div style="position: relative; padding-right: 10px; padding-left: 0px; width: 30px;"
class="DG-spinner-container"><input id="delay" class="DG-spinner DG-spinner-input"
maxlength="3" value="200" style="width: 30px; border: 0px none;"
properties="minValue:0,maxValue:500,increment:10,label:delayLabel" type="text"><div
style="position: absolute; top: 0px; height: 100%; right: 0px; border-left-width: 1px; border-left-
style: solid;" class="DG-spinner-arrows-container"><div style="position: absolute; background-
repeat: no-repeat; background-position: center center; height: 50%; top: 0px;" class="DG-
spinner-arrow-up"></div><div style="position: absolute; background-repeat: no-repeat;
background-position: center center; height: 50%; bottom: 0px;" class="DG-spinner-arrow-
down"></div><div style="position: absolute; top: 50%;" class="DG-spinner-arrow-
separator"></div></div></div></td>
<!--
Delay: <span id="speed"></span> ms<br/>
<button onclick="javascript:if (autoDelay>0) autoDelay -= 10;showSpeed();"
style="width:10mm">&mdash;</button>
<button onclick="javascript:autoDelay += 10;showSpeed();" style="width:10mm">+</button>
-->
<td bgcolor="#00FFFF">
<a title="Если установлен, вызовы других функций выполняются сразу (все сразу), а не
шаг за шагом."><input id="skipover" value="no" type="checkbox"><label
for="skipover">Пропустить<br>вызовы функции</label></a>
</td></tr></tbody></table>
<hr>
<table>
<tbody><tr><td align="center">Алгоритм сортировки:</td>
<td><select id="method" onchange="changeMethod(this);" style="width:50mm">
<script>
for (var j=0; j<sortCount; j++)
dw('<OPTION ' + (j==0 ? 'SELECTED ' : '') + 'value="' + sortName[j] + '>' +
sortName[j]);

```

```

</script></select>
</td></tr><tr><td align="right">Данные:</td>
<td><select      name="dataType"      id="dataType"      onchange="changeData(this);"
style="width:50mm">
<option selected="selected" value="numbers">Числа</option>
<option value="words">Текст</option>
<option value="bars">Гистограмма</option>
</select>
</td></tr><tr><td align="right">Размещение данных:</td>
<td><select      name="randType"      id="randType"      onchange="changeRand(this);"
style="width:50mm">
<option selected="selected" value="1">Перемешанные (без повторения)</option>
<option value="2">По возрастанию</option>
<option value="3">По убыванию</option>
<option value="4">Перемешанные (с повторениями)</option>
<option value="5">Константа</option>
</select>
</td></tr>
<tr>
<td align="right"><label style="cursor: w-resize;" id="sizeLabel">Размерность
массива:</label></td>
<td><div style="position: relative; padding-right: 10px; padding-left: 0px; width: 30px;"
class="DG-spinner-container"><input id="dataSize" class="DG-spinner DG-spinner-input"
maxlength="2" value="15" style="width: 30px; border: 0px none;"
properties="minValue:10,maxValue:20,label:sizeLabel" type="text"><div style="position:
absolute; top: 0px; height: 100%; right: 0px; border-left-width: 1px; border-left-style: solid;"
class="DG-spinner-arrows-container"><div style="position: absolute; background-repeat: no-
repeat; background-position: center center; height: 50%; top: 0px;" class="DG-spinner-arrow-
up"></div><div style="position: absolute; background-repeat: no-repeat; background-position:
center center; height: 50%; bottom: 0px;" class="DG-spinner-arrow-down"></div><div
style="position: absolute; top: 50%;" class="DG-spinner-arrow-
separator"></div></div></div></td>
</tr></tbody></table>
<hr>
</td></tr>
<tr><td><table>
<tbody><tr valign="top"><td height="90"><u><i>Variables</i></u><br>
<span id="vars">j, i = ?</span></td></tr>
<tr><td id="rec"></td></tr>
</tbody></table>
</td>
</tr></tbody></table></td>
<td width="5"></td>
<td><table>
<script>
for (var i=0; i<arraySize+1; i++) {
    dw('<tr valign=center height=20>');
    dw('<td width=120 id="lbl' + i + '" align=right></td>');
    dw('<td width=80 id="n' + i + '" align=center></td>');
    dw('<td width=80 id="piv' + i + '" align=left></td>');
    dw('</tr>');
}

```



## ДОДАТОК Б

### Лістинг sortcode.js

```
var code = [];
var commands = [];
var sortName = [];
var objName = [];
var sortCount = 0;
var whichSort;

sortName[sortCount++] = whichSort = 'Quicksort (pivot=last)';
objName[whichSort] = 'QuickData(0,dataSize)';
code[whichSort] = [
    '@s0:QuickSort(low, high)@ {',
    ' @s1:if (high-low &lt;= 1)@ @s2:return@;',
    ' @s3:pivot = a[high-1]@;',
    ' @s4:split = low@;',
    ' for (@s5:i=low@; @s6:i&lt;high-1@; @s10:i++@)',
    ' @s7:if (a[i] &lt;pivot)@',
    ' {',
    ' @s8:swap a[i] and a[split]@;',
    ' @s9:split++@;',
    ' }',
    ' @s11:swap a[high-1] and a[split]@;',
    ' @s12:QuickSort(low, split)@;',
    ' @s13:QuickSort(split+1, high)@;',
    ' @s14:return@;',
    '}' ];
commands[whichSort] = [
    'showNums(); showLowHigh();',
    'if (high-low > 1) step++;',
    'popAll(); if (recDepth < 0) step = -1; else showLowHigh();',
    'pivot = a[high-1]; _elColor(\n'+(high-1), \lightblue\);',
    'split = low;',
    'i = low;',
    'if (i>=high-1) step = 11;',
    'if (a[i]>=pivot) step = 10;',
    'swap(i,split);',
    'split++;',
    'i++; step = 6;',
    'swap(high-1,split); showLowHigh(); _elColor(\n'+split, \lightblue\);',
    'if (_gel(\skipover\).checked) { Sort(low,split); showNums(); } else { pushAll(); SortData =',
    'new QuickData(low, split); showLowHigh(); clearPiv(); }',
    'if (_gel(\skipover\).checked) { Sort(split+1,high); showNums(); } else { pushAll(); SortData =',
    'new QuickData(split+1, high); showLowHigh(); clearPiv(); }',
    'popAll(); if (recDepth < 0) step = -1; else showLowHigh();' ];

sortName[sortCount++] = whichSort = 'Quicksort (pivot=median-of-three)';
objName[whichSort] = 'QuickData(0,dataSize)';
code[whichSort] = [
    '@s0:QuickSort(low, high)@ {',
    ' @s1:if (high-low &lt;= 1)@ @s2:return@;',
```

```

    ' @s3:pivot = MedianOf3(low, high)@;',
    ' @s4:split = low@;',
    ' for (@s5:i=low@; @s6:i<high-1@; @s10:i++@)',
    ' @s7:if (a[i] < pivot)@',
    ' {',
    ' @s8:swap a[i] and a[split]@;',
    ' @s9:split++@;',
    ' }',
    ' @s11:swap a[high-1] and a[split]@;',
    ' @s12:QuickSort(low, split)@;',
    ' @s13:QuickSort(split+1, high)@;',
    ' @s14:return@;',
    '}',
    '@s15:MedianOf3(low, high)@ {',
    '@s16:middle = (low + high) / 2;@',
    '@s17:if (a[low] > a[middle])@',
    '@s18:swap a[low] and a[middle];@',
    '@s19:if (a[low] > a[high-1])@',
    '@s20:swap a[low] and a[high-1];@',
    '@s21:if (a[middle] < a[high-1])@',
    '@s22:swap a[middle] and a[high-1];@',
    // '@s23:swap a[middle] and a[high - 1];@',
    '@s23:return a[high - 1];@',
    '}' ];
commands[whichSort] = [
    'showNums(); showLowHigh();',
    'if (high-low > 1) step++;',
    'popAll(); if (recDepth < 0) step = -1; else showLowHigh();',
    'if (_gel(\'skipover\').checked) { MedianOf3(low, high); showNums(); pivot = a[high-1];',
    '_elColor(\'n\'+(high-1), \'lightblue\'); } else { pushAll(); SortData = new MedianData(low,high);',
    'showLowHigh(); }',
    'split = low;',
    'i = low;',
    'if (i>=high-1) step = 11;',
    'if (a[i]>=pivot) step = 10;',
    'swap(i,split);',
    'split++;',
    'i++; step = 6;',
    'swap(high-1,split); showLowHigh(); _elColor(\'n\' + split, \'lightblue\');',
    'if (_gel(\'skipover\').checked) { Sort(low,split); showNums(); } else { pushAll(); SortData =',
    'new QuickData(low, split); showLowHigh(); clearPiv(); }',
    'if (_gel(\'skipover\').checked) { Sort(split+1,high); showNums(); } else { pushAll(); SortData =',
    'new QuickData(split+1, high); showLowHigh(); clearPiv(); }',
    'popAll(); if (recDepth < 0) step = -1; else showLowHigh();',
    'showNums(); showLowHigh();',
    'middle = Math.floor((low + high) / 2);',
    'if (a[low] <= a[middle]) step++;',
    'swap(low,middle);',
    'if (a[low] <= a[high-1]) step++;',
    'swap(low,high-1);',
    'if (a[middle] >= a[high-1]) step++;',
    // 'swap(middle,high-1);',

```

```

        'swap(middle,high-1);',
        'popAll(); SortData.pivot = a[high-1]; showLowHigh(); _elColor('\n'+(high-1), 'lightblue\');'
    ];
    function SWAP(i,j) {
        var t = a[i];
        a[i] = a[j];
        a[j] = t;
    }
    function Sort(low, high) {
        if (high-low <= 1) return;
        var pivot = a[high-1];
        var split = low;
        for (i=low; i<high-1; i++)
            if (a[i] < pivot)
            {
                SWAP(i,split);
                split++;
            }
        a[high-1] = a[split];
        a[split] = pivot;
        Sort(low, split);
        Sort(split+1, high);
    }
    function MedianOf3(low, high) {
        middle = (low + high) / 2;
        if (a[low] > a[middle]) SWAP(low,middle);
        if (a[low] > a[high-1]) SWAP(low,high-1);
        if (a[middle] < a[high-1]) SWAP(middle,high-1);
    }
    function QuickData(low, high) {
        this.subFlag = false;
        this.low = low;
        this.high = high;
        this.step = 0;
        this.pivot = this.split = this.i = undefined;
    }
    QuickData.prototype.toString = function() {
        return 'QuickSort('+this.low+', '+this.high+')';
    }
    function MedianData(low, high) {
        this.subFlag = true;
        this.low = low;
        this.high = high;
        this.step = 15;
        this.middle = undefined;
    }
    MedianData.prototype.toString = function() {
        return 'MedianOf3('+this.low+', '+this.high+')';
    }
    sortName[sortCount++] = whichSort = 'Bubble Sort';
    objName[whichSort] = 'BubbleData(dataSize)';
    code[whichSort] = [

```

```

    '@s0:BubbleSort(size)@ {' ,
    ' for (@s1:i=size-1 @; @s2:i > 0@; @s8:i--@)',
    ' {' ,
    ' for (@s3:j=0@; @s4:j < i@; @s7:j++@)',
    ' {' ,
    ' @s5:if (a[j] > a[j+1])@',
    ' @s6:swap a[j] and a[j+1]@;',
    ' }',
    ' }',
    '}' ];

commands[whichSort] = [
    'showNums();',
    'i = size - 1;',
    'if (i <= 0) step = -1;',
    'j = 0;',
    'if (j>= i) step = 8;',
    'if (a[j] <= a[j+1]) step++;',
    'swap(j,j+1);',
    'j++; step = 4;',
    'i--; step = 2;' ];

sortName[sortCount++] = whichSort = 'Bubble Sort (smart)';
objName[whichSort] = 'BubbleData(dataSize)';
code[whichSort] = [
    '@s0:BubbleSort(size)@ {' ,
    ' for (@s1:i=size-1 @; @s2:i > 0@; @s12:i--@)',
    ' {' ,
    ' @s3:swaps = false@;',
    ' for (@s4:j=0@; @s5:j < i@; @s9:j++@)',
    ' {' ,
    ' @s6:if (a[j] > a[j+1])@',
    ' {' ,
    ' @s7:swap a[j] and a[j+1]@;',
    ' @s8:swaps = true@;',
    ' }',
    ' }',
    ' @s10:if (!swaps)@ @s11:break@;',
    ' }',
    '}' ];

commands[whichSort] = [
    'showNums();',
    'i = size - 1;',
    'if (i <= 0) step = -1;',
    'swaps = false;',
    'j = 0;',
    'if (j>= i) step = 10;',
    'if (a[j] <= a[j+1]) step = 9;',
    'swap(j,j+1);',
    'swaps = true;',
    'j++; step = 5;',
    'if (swaps) step++;',
    'step = -1;',

```

```

        'i--; step = 2;' ];
function BubbleData(size) {
    this.i = this.j = this.swaps = undefined;
    this.size = size;
    this.step = 0;
    this.low = 0;
    this.high = size;
}

sortName[sortCount++] = whichSort = 'Gnome Sort';
objName[whichSort] = 'GnomeData(dataSize)';
code[whichSort] = [
    '@s0:GnomeSort(size)@ {',
    ' @s1:i = 0@;',
    ' @s2:j = 1@;',
    ' @s3:while (i < size-1)@',
    ' {',
    ' @s4:if (a[i] > a[i+1])@',
    ' @s5:i = j++@;',
    ' else',
    ' {',
    ' @s6:swap a[i] and a[i+1]@;',
    ' @s7:i--@;',
    ' @s8:if (i < 0)@',
    ' @s9:i = 0@;',
    ' }',
    '}',
    '}' ];
commands[whichSort] = [
    'showNums();',
    'i = 0;',
    'j = 1;',
    'if (i == size-1) step = -1;',
    'if (a[i] > a[i+1]) step = 6;',
    'i = j++; step = 3;',
    'swap(i,i+1);',
    'i--;',
    'if (i>=0) step = 3;',
    'i = 0; step = 3;' ];

function GnomeData(size) {
    this.i = this.j = undefined;
    this.size = size;
    this.step = 0;
    this.low = 0;
    this.high = size;
}

sortName[sortCount++] = whichSort = 'Shaker Sort';
objName[whichSort] = 'ShakerData(dataSize)';
code[whichSort] = [
    '@s0:ShakerSort(size)@ {',
    ' for (@s1:i=0, k=size-1@; @s2:i < k@; @s15:i++, k--@)',

```



```

    '{',
    ' @s3:min = max = i@;',
    ' for (@s4:j = i + 1@; @s5:j <= k@; @s10:j++@)',
    '{',
    ' @s6:if (a[j] < a[min])@',
    ' @s7:min = j@;',
    ' @s8:if (a[j] > a[max])@',
    ' @s9:max = j@;',
    '}',
    ' @s11:swap a[min] and a[i]@;',
    ' @s12:if (max == i)@',
    ' @s13:swap a[min] and a[k]@;',
    ' else',
    ' @s14:swap a[max] and a[k]@;',
    '}',
    '}' ];
commands[whichSort] = [
    'showNums();',
    'i = 0; k = size - 1;',
    'if (i >= k) step = -1;',
    'min = max = i;',
    'j = i + 1;',
    'if (j > k) step = 11;',
    'if (a[j] >= a[min]) step++;',
    'min = j;',
    'if (a[j] <= a[max]) step++;',
    'max = j;',
    'j++; step = 5;',
    'swap(min,i);',
    'if (max != i) step++;',
    'swap(min,k); step++;',
    'swap(max,k);',
    'i++; k--; step = 2;'];

function ShakerData(size) {
    this.i = this.j = this.k = this.min = this.max = undefined;
    this.size = size;
    this.step = 0;
    this.low = 0;
    this.high = size;
}

sortName[sortCount++] = whichSort = 'CombSort11';
objName[whichSort] = 'CombData(dataSize)';
code[whichSort] = [
    '@s0:CombSort11(size)@ {',
    ' @s1:gap = size@;',
    ' do',
    '{',
    ' @s2:if (gap > 1)@',
    '{',
    ' @s3:gap = gap / 1.3@;',

```

```

        ' @s4:if (gap == 10 || gap == 9)@',
        ' @s5:gap = 11@;',
        ' }',
        ' @s6:swaps = false@;',
        ' for (@s7:i=0@; @s8:i + gap < size@; @s12:i++@)',
        ' {',
        ' @s9:if (a[i] > a[i+gap])@',
        ' {',
        ' @s10:swap a[i] and a[i+gap]@;',
        ' @s11:swaps = true@;',
        ' }',
        ' }',
        ' } @s13:while (gap > 1 || swaps)@;',
        '}' ];

commands[whichSort] = [
    'showNums();',
    'gap = size;',
    'if (gap <= 1) step = 6;',
    'gap = Math.floor(gap / 1.3);',
    'if (gap != 10 && gap != 9) step++;',
    'gap = 11;',
    'swaps = false;',
    'i=0;',
    'if (i + gap >= size) step = 13;',
    'if (a[i] <= a[i+gap]) step = 12;',
    'swap(i,i+gap);',
    'swaps = true;',
    'i++; step = 8;',
    'if (gap <=1 && !swaps) step = -1; else step = 2' ];

function CombData(size) {
    this.i = this.gap = this.swaps = undefined;
    this.size = size;
    this.step = 0;
    this.low = 0;
    this.high = size;
}

sortName[sortCount++] = whichSort = 'Insertion Sort';
objName[whichSort] = 'InsertData(dataSize)';
code[whichSort] = [
    '@s0:InsertionSort(size)@ {',
    ' for (@s1:i = 1@; @s2:i < size@; @s10:i++@)',
    ' {',
    ' @s3:if (a[i] < a[i-1])@',
    ' {',
    ' @s4:temp = a[i]@;',
    ' for (@s5:j = i@; @s6:j>0 && a[j-1]>temp@; @s8:j--@)',
    ' @s7:a[j] = a[j-1]@;',
    ' @s9:a[j]=temp@;',
    ' }',
    ' }',
    '}' ];

```

```

commands[whichSort] = [
    'showNums();',
    'i = 1;',
    'if (i >= size) step= -1;',
    'if (a[i] >= a[i-1]) step = 10;',
    'temp = a[i];',
    'j = i;',
    'if (j<1 || a[j-1]<=temp) step = 9;',
    'a[j] = a[j-1]; showNums();',
    'j--; step = 6;',
    'a[j] = temp; showNums();',
    'i++; step = 2;' ];

function InsertData(size) {
    this.i = this.j = this.temp = undefined;
    this.size = size;
    this.step = 0;
    this.low = 0;
    this.high = size;
}
sortName[sortCount++] = whichSort = 'Shell Sort';
objName[whichSort] = 'ShellData(dataSize)';
code[whichSort] = [
    '@s0:ShellSort(size)@ {' ,
        ' for (@s1:inc = size/2@; @s2:inc>0@; @s12:inc /= 2@)',
        '{',
        ' for (@s3:i = inc@; @s4:i < size@; @s11:i++@)',
        '{',
        '   @s5:j = i - inc@;',
        '   @s6:while (j >= 0)@',
        '   {' ,
        '     @s7:if (a[j] > a[j+inc])@',
        '     {' ,
        '       @s8:swap a[j] and a[j+inc]@;',
        '       @s9:j -= inc@;',
        '     } else {' ,
        '       @s10:j = -1@;',
        '     }',
        '   }',
        ' }',
        ' }',
        ' }',
        ' }';
commands[whichSort] = [
    'showNums();',
    'inc = Math.floor(size / 2);',
    'if (inc <= 0) step = -1;',
    'i = inc;',
    'if (i >= size) step = 12;',
    'j = i - inc;',
    'if (j < 0) step = 11;',
    'if (a[j] <= a[j+inc]) step = 10;',

```

```

        'swap(j,j+inc);',
        'j -= inc; step = 6;',
        'j = -1; step = 6;',
        'i++; step = 4;',
        'inc = Math.floor(inc/2); step = 2;' ];
function ShellData(size) {
    this.i = this.j = this.inc = undefined;
    this.size = size;
    this.step = 0;
    this.low = 0;
    this.high = size;
}
sortName[sortCount++] = whichSort = 'Heap Sort';
objName[whichSort] = 'HeapData(dataSize)';
code[whichSort] = [
    '@s0:HeapSort(size)@ {' ,
    ' for (@s1:i = size/2@; @s2:i >= 0@; @s4:i--@)',
    '   @s3:ReHeap(size, i)@;',
    ' for (@s5:i = size-1 @; @s6:i > 0@; @s9:i--@)',
    '   {' ,
    '     @s7:swap a[i] and a[0]@;',
    '     @s8:ReHeap(i, 0)@;',
    '   }',
    '}',
    '@s10:ReHeap(len, parent)@ {' ,
    '   @s11:temp = a[parent]@;',
    '   @s12:child = 2*parent + 1@;',
    '   @s13:while (child < len)@',
    '   {' ,
    '     @s14:if (child<len-1 && a[child]<a[child+1])@',
    '     @s15:child++@;',
    '     @s16:if (temp >= a[child])@ @s17:break@;',
    '     @s18:a[parent] = a[child]@;',
    '     @s19:parent = child@;',
    '     @s20:child = 2*parent + 1@;',
    '   }',
    '   @s21:a[parent] = temp@;',
    '   @s22:return@;',
    '}' ]
commands[whichSort] = [
    'showNums();',
    'i = Math.floor(size/2);',
    'if (i < 0) step = 5;',
    'if (_gel(\'skipover\').checked) { ReHeap(size,i); showNums(); } else { pushAll(); SortData = new ReHeapData(size, i); showLowHigh(); }',
    'i--; step = 2;',
    'i = size-1;',
    'if (i <= 0) step = -1;',
    'swap(i,0);',
    'if (_gel(\'skipover\').checked) { ReHeap(i,0); showNums(); } else { pushAll(); SortData = new ReHeapData(i, 0); showLowHigh(); }',
    'i--; step = 6;',

```

```

        'showNums()';
        'temp = a[parent];',
        'child = 2*parent + 1;',
        'if (child >= len) step = 21;',
        'if (child >= len - 1 || a[child] >= a[child + 1]) step = 16;',
        'child++;',
        'if (temp < a[child]) step = 18;',
        'step = 21;',
        'a[parent] = a[child]; showNums()',
        'parent = child;',
        'child = 2*parent + 1; step = 13;',
        'a[parent] = temp; showNums()',
        'popAll(); showLowHigh();' ];
function ReHeap(len, parent) {
    var temp = a[parent];
    var child = 2*parent + 1;
    while (child < len)
    {
        if (child < len - 1 && a[child] < a[child + 1]) child++;
        if (temp >= a[child]) break;
        a[parent] = a[child];
        parent = child;
        child = 2*parent + 1;
    }
    a[parent] = temp;
}
function HeapData(size) {
    this.subFlag = false;
    this.i = undefined;
    this.size = size;
    this.step = 0;
    this.low = 0;
    this.high = size;
}
function ReHeapData(a,b) {
    this.subFlag = true;
    this.temp = this.child = undefined;
    this.len = a;
    this.parent = b;
    this.size = undefined;
    this.step = 10;
    this.low = b;
    this.high = a;
}

HeapData.prototype.toString = function() {
    return 'HeapSort('+this.size+')';
}
ReHeapData.prototype.toString = function() {
    return 'ReHeap('+this.high+', '+this.low+')';
}
sortName[sortCount++] = whichSort = 'Merge Sort';

```

```

objName[whichSort] = 'MergeData(0,dataSize)';
code[whichSort] = [
    '@s0:MergeSort(low, high)@ {',
    ' @s1:if (high-low <= 1)@ @s2:return@;',
    ' @s3:split = (low+high) / 2@;',
    ' @s4:MergeSort(low, split)@;',
    ' @s5:MergeSort(split, high)@;',
    ' @s6:copy a[low ... split-1] to scratch array@;',
    ' @s7:m1 = 0@;',
    ' @s8:m2 = split@;',
    ' @s9:i = low@;',
    ' @s10:while (i < m2 && m2 < high)@',
    ' @s11:if (scratch[m1] <= a[m2])@',
    ' @s12:a[i++] = scratch[m1++]@;',
    ' else',
    ' @s13:a[i++] = a[m2++]@;',
    ' @s14:while (i < m2)@',
    ' @s15:a[i++] = scratch[m1++]@;',
    '}' ];
commands[whichSort] = [
    'showNums();',
    'if (high-low > 1) step++;',
    'popAll(); clearPiv(); if (recDepth < 0) step = -1; else showLowHigh();',
    'split = Math.floor((low+high) / 2);',
    'if (_gel(\'skipover\').checked) { Merge(low,split); showNums(); } else { pushAll(); SortData =',
    'new MergeData(low, split); showLowHigh(); }',
    'if (_gel(\'skipover\').checked) { Merge(split, high); showNums(); } else { pushAll(); SortData',
    '= new MergeData(split, high); showLowHigh(); }',
    'for (var j=low; j<split; j++) { scratch[j-low] = a[j]; } scratchFlag = true; showScratch();',
    'm1 = 0; showScratch();',
    'm2 = split; showScratch();',
    'i = low;',
    'if (i>=m2 || m2>=high) step = 14;',
    'if (scratch[m1]>a[m2]) step++;',
    'a[i++] = scratch[m1++]; showNums(); showLowHigh(); showScratch(); step = 10;',
    'a[i++] = a[m2++]; showNums(); showLowHigh(); showScratch(); step = 10;',
    'if (i>=m2) { popAll(); clearPiv(); if (recDepth < 0) step = -1; else showLowHigh(); }',
    'a[i++] = scratch[m1++]; showNums(); showLowHigh(); showScratch(); step = 14;' ];

function Merge(low, high) {
    if (high-low <= 1) return;
    var split = Math.floor((low+high) / 2);
    Merge(low, split);
    Merge(split, high);
    var scratch = [high-low];
    var m1 = 0;
    var m2 = split;
    var i;
    for (i = low; i<split; i++) scratch[i-low] = a[i];
    // alert("scratch=" + scratch + "\n a=" + a + "\n i,low,high=" + i + "," + low + "," + high);
    i = low;
    while (i < m2 && m2 < high)

```

```

        if (scratch[m1] <= a[m2])
            a[i++] = scratch[m1++];
        else
            a[i++] = a[m2++];
        while (i < m2)
            a[i++] = scratch[m1++];
    }

function MergeData(a,b) {
    this.i = this.split = this.m1 = this.m2 = undefined;
    this.scratch = [b-a];
    this.scratchFlag = false;
//    this.size = size;
    this.step = 0;
    this.low = a;
    this.high = b;
}

MergeData.prototype.toString = function() {
    return 'MergeSort(' + this.low + ',' + this.high + ')';
}

sortName[sortCount++] = whichSort = 'Radix Sort';
objName[whichSort] = 'RadixData(dataSize)';
code[whichSort] = [
    '@s0:Radix(size)@ {' ,
    ' @s1:allZero = false@;',
    ' for (@s2:i=0@; @s3:!allZero@; @s17:i++@)',
    ' {' ,
    ' @s4:allZero = true@;',
    ' for (@s5:j=m=n=0@; @s6:j<size@; @s13:j++@)',
    ' {' ,
    ' @s7:tmp = <a title="this means \'divide a[j] by 2^i\'">a[j] &gt;&gt; i@</a>;',
    ' @s8:if (tmp > 1)@ @s9:allZero = false@;',
    ' @s10:if ((tmp & 1) == 1)@',
    ' @s11:bucket[m++] = a[j]@;',
    ' else',
    ' @s12:a[n++] = a[j]@;',
    ' }',
    ' @s14:m = 0@;',
    ' while (@s15:n<size@)',
    ' @s16:a[n++] = bucket[m++]@;',
    ' }',
    ' }' ];
commands[whichSort] = [
    'showNums();',
    'allZero = false;',
    'i=0;',
    'bucketsize=0; showBucket(); if (allZero) step = -1;',
    'allZero = true;',
    'j=m=n=0;',
    'if (j>=size) step = 14;',

```

```

        'tmp = a[j] >> i;',
        'if (tmp <= 1) step++;',
        'allZero = false;',
        'if ((tmp & 1) != 1) step++;',
        'bucket[m++] = a[j]; bucketSize++; showBucket(); step = 13;',
        'a[n++] = a[j]; showNums();',
        'j++; step = 6;',
        'm = 0; showBucket();',
        'if (n>=size) step++;',
        'a[n++] = bucket[m++]; showBucket(); showNums(); step = 15;',
        'i++; step = 3;'];

function RadixData(size) {
    this.i = this.j = this.m = this.n = this.tmp = this.allZero = undefined;
    this.bucket = [size];
    this.bucketSize = 0;
    this.step = 0;
    this.low = 0;
    this.high = this.size = size;
}
RadixData.prototype.toString = function() {
    return 'RadixData: allZero=' + this.allZero + '\ni=' + this.i + '\nj=' + this.j;
}

```



## ДОДАТОК В

### Лістинг nester.css

```
.inline {
    display:inline;
}
span.arrow-visible {
    background: url('images/arrow-visible.gif') no-repeat left center;
    width: 30px;
    display: block;
    float: left;
    clear: left;
}
span.arrow-hidden {
    background: url('images/arrow-hidden.gif') no-repeat left center;
    width: 30px;
    display: block;
    float: left;
    clear: left;
}
div.header {
    padding: 2px 10px;
    margin-bottom: 3px;
    margin-top: 6px;
    font-weight: bold;
    background-color: #7521C7;
    color: ghostwhite;
}
div.collapseText {
    background: white;
    // color: black;
    padding: 8px;
    border: 3px solid #D3D3D3;
    font-family: sans-serif;
}
table.fraction {
    background-color:inherit;
    border-spacing: 0px 0px;
    // display: inline-table;
    display: inline-block;
    vertical-align: middle;
    margin: 1px 4px;
    font-weight: inherit;
}
td.numerator {
    background-color:inherit;
    text-align: center;
    border-bottom: thin solid;
    font-size: 10px;
    font-weight: inherit;
}
td.denominator {
```

```

        background-color:inherit;
        text-align: center;
        font-size: 10px;
    }

.transparent {
    background: transparent;
    filter: alpha(opacity=0);
    -moz-opacity: .0;
    opacity: .0;
}
pre,.code{
    font-family:"Courier New", Courier, monospace;
}
.note {
    font-size: 12px;
    font-style: oblique;
}
a:link {
    color: #638EDE;
    text-decoration: none;
}
a:visited {
    color: #9090b0;
    text-decoration: none;
}
a:hover {
    color: #ffffff; /*#605887;
    background-color: #7521C7;
    //text-decoration: underline;
}
a:active {
    color: #638EDE;
}
body,p,table,ul,ol {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 15px;
    font-weight: normal;
    color: #3f3f3f;
    margin-left: 8px;
    margin-top: 2px;
    margin-right: 8px;
    margin-bottom: 2px;
    background-color: #FFFFFF;
}
ul.disc {list-style-type: disc}
ul.circle {list-style-type: circle}
ul.square {list-style-type: square}
ul.none {list-style-type: none}
.center { text-align:center; }
.right { text-align:right; }

```

```
th {
  background-color: #7521C7;
  color: #FFFFFF;
}
h1,h2,h3,h4,h5 {
  color: #2f1b57;
  font-variant: small-caps;
  font-weight: bold;
}
h1 {font-size: 18px;}
h2 {font-size: 16px;}
h3 {font-size: 15px;}
h4 {font-size: 14px;}
h5 {font-size: 12px;}
.content IMG {
  margin: 8px;
}
.content A IMG {
  border-color: blue;
  border-width: 1px;
}
```