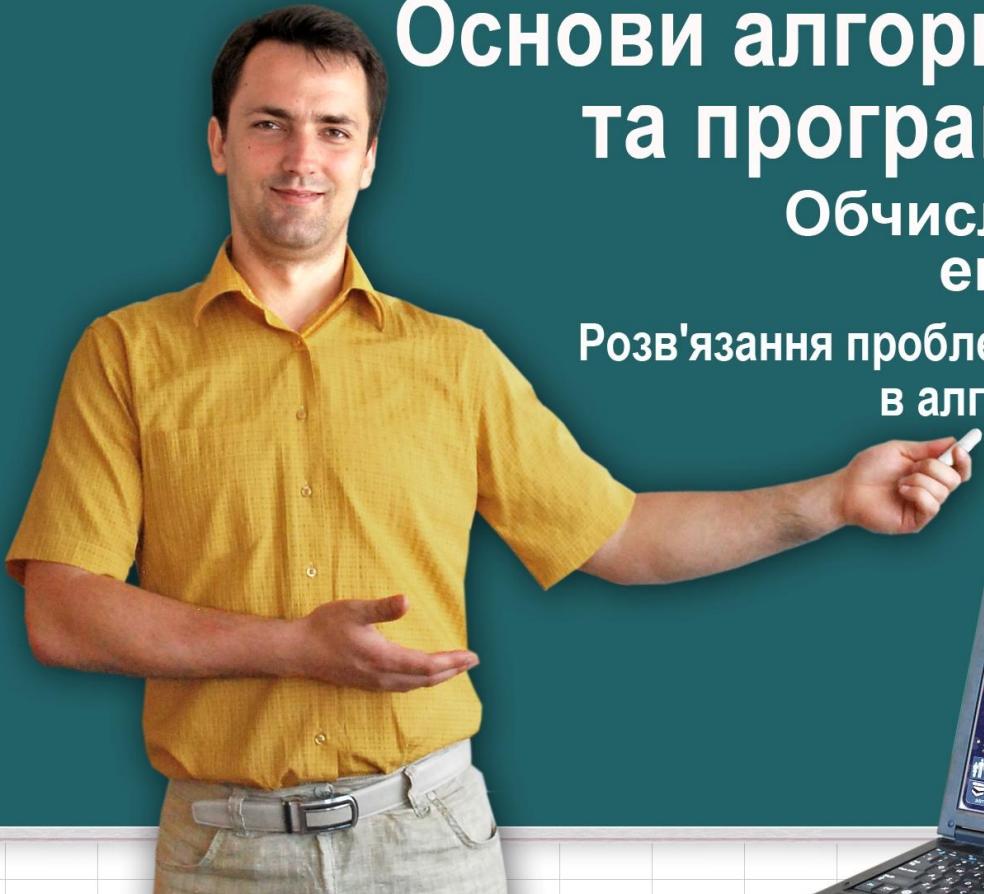


Співаковський О.В.
Осипова Н.В.
Львов М.С.
Бакуменко К.В.

Основи алгоритмізації та програмування

Обчислювальний експеримент

Розв'язання проблем ефективності
в алгоритмах пошуку
та сортування



Херсон 2011

**Міністерство освіти і науки, молоді та спорту України
Херсонський державний університет**

Співаковський О.В.

Осипова Н.В.

Львов М.С.

Бакуменко К.В.

ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ

ОБЧИСЛЮВАЛЬНИЙ ЕКСПЕРИМЕНТ

РОЗВ'ЯЗАННЯ ПРОБЛЕМ ЕФЕКТИВНОСТІ
В АЛГОРИТМАХ ПОШУКУ ТА СОРТУВАННЯ

Навчальний посібник

**Рекомендовано
Міністерством освіти і науки, молоді та спорту України
як навчальний посібник для студентів
вищих навчальних закладів**

Херсон – 2011

УДК 004.42
ББК 32.973.2 я 72

Гриф надано
Міністерством освіти і науки, молоді та спорту України
(лист № 1/11-4065 від 25.05.2011 р.)

Рецензенти:

C.А. Раков – начальник відділу наукового забезпечення українського центру оцінки якості освіти, доктор педагогічних наук, професор.

Г.М. Жолткевич – декан механіко-математичного факультету, завідувач кафедрою теоретичної та прикладної інформатики Харківського національного університету імені В.Н. Каразіна, доктор технічних наук, професор.

Н.В. Морзе – проректор з навчально-наукових питань інформатизації та телекомуникаційних систем в АПК Національного університету біоресурсів і природокористування України, доктор педагогічних наук, професор.

М.В. Працьовитий – директор фізико-математичного інституту, завідуючий кафедри вищої математики НПУ імені М.П. Драгоманова, доктор фізико-математичних наук, професор.

Співаковський О.В., Осипова Н.В., Львов М.С., Бакуменко К.В.

С-72 Основи алгоритмізації та програмування. Обчислювальний експеримент. Розв'язання проблем ефективності в алгоритмах пошуку та сортування: Навчальний посібник. – Херсон: Айлант. – 2011. – 100 с.: іл.

ISBN 978-966-630-049-7

У посібнику запропоновано модель вивчення основ алгоритмізації та програмування з використанням інтегрованого середовища, де на відміну від традиційного підходу головна увага приділяється задачі аналізу на всіх стадіях процесу проектування та реалізації алгоритмів. Описано новий підхід до вивчення поняття складності та вивчення властивостей алгоритмів і вибору оптимального алгоритма, що має величезне значення для порівняння різних алгоритмів і використання їх для розв'язування практичних задач. Головна увага приділяється проведенню обчислювального експерименту для вивчення складності та мажорованості алгоритмів сортування з використанням інтегрованого середовища WEBOAP (weboap.ksu.ks.ua), розробленого у НДІ ІТ Херсонського державного університету.

Посібник призначений для викладачів, аспірантів, студентів вищих навчальних закладів, вчителів з інформатики та учнів старших класів загальноосвітніх навчальних закладів.

ISBN 978-966-630-049-7

© Співаковський О.В., Осипова Н.В.,

Львов М.С., Бакуменко К.В.

© ХДУ, 2011

ЗМІСТ

ВСТУП.....	5
ПОДЯКИ	9
ПОЯСНЮВАЛЬНА ЗАПИСКА	10
ПРАКТИЧНА РОБОТА №1	16
ПРАКТИЧНА РОБОТА №2	23
ПРАКТИЧНА РОБОТА №3	26
ПРАКТИЧНА РОБОТА №4	31
ПРАКТИЧНА РОБОТА №5	33
ПРАКТИЧНА РОБОТА №6	35
ПРАКТИЧНА РОБОТА №7	38
ПРАКТИЧНА РОБОТА №8	41
ПРАКТИЧНА РОБОТА №9	46
ПРАКТИЧНА РОБОТА №10	50
ПРАКТИЧНА РОБОТА №11	55
ПРАКТИЧНА РОБОТА №12	61
ПРАКТИЧНА РОБОТА №13	65
ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМНОГО ЗАСОБУ ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ КУРСУ «ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ» ДЛЯ ВИЩИХ НАВЧАЛЬНИХ ЗАКЛАДІВ	72
1. ПРИЗНАЧЕННЯ ПРОГРАМНОГО ЗАСОБУ	72
2. ЗАПУСК ПРОГРАМНОГО ЗАСОБУ	74
3. МОДУЛЬ «ЕЛЕКТРОННИЙ НАВЧАЛЬНИЙ ПОСІБНИК» ..	79
4. МОДУЛЬ «БІБЛІОТЕКА ЛЕКЦІЙ»	82
5. СЕРЕДОВИЩЕ ДЕМОНСТРАЦІЇ	84
6. МОДУЛЬ «БІБЛІОТЕКА ЗАДАЧ»	92
7 МОДУЛЬ «СИСТЕМА ПОТОЧНОГО ТА ПІДСУМКОВОГО КОНТРОЛЮ ЗНАНЬ»	97
ПРЕДМЕТНИЙ ПОКАЖЧИК	100

ВСТУП

При традиційному навченні програмістів велику увагу приділяють вивченняю мов програмування, умінню працювати з транслятором, операційною системою. Як завдання найчастіше розглядають приклади чисельного характеру, основна суть яких – формульні обчислення. Цього досить, щоб писати нескладні прикладні програми, але мало для того, щоб навчитися створювати програми, з якими працюватимуть багато користувачів тривалий час: такі програми мають бути надійними і ефективними.

Професійний програміст повинен не лише володіти високою загальною культурою і фундаментальними основами неперервної та дискретної математики, але повинен також глибоко розуміти теорію складності алгоритмів, знати багато ефективних типових алгоритмів, уміти доводити властивості і досліджувати ефективність алгоритмів. Тільки тоді він буде здатний створювати хороші програми.

Коли програмісти, розв'язуючи навчальні задачі, тестиють свої програми, то зазвичай використовують невеликі значення параметрів. Тому навіть якщо при написанні програми був застосований неефективний алгоритм, це може залишитися непоміченим. Проте, якщо подібну програму спробувати застосувати в реальних умовах, її практична непридатність виявиться негайно.

Тому у змістовному компоненті курсу програмування поряд з теоретичним матеріалом ми передбачаємо проведення обчислювального експерименту для вивчення складності і підвищення ефективності алгоритмів. Такий підхід до змісту призводить до посилення дослідницької діяльності студентів, до фундаменталізації предметної підготовки майбутніх фахівців, за рахунок формально-логічного відображення причинно-наслідкових зв'язків і, як наслідок, до впливу на мотивацію студентів.

Основна мета цієї книги – розробка методичної системи з проведення обчислювального експерименту для вивчення складності алгоритмів з використанням інтегрованого середовища WEBOAP (інтегрованого середовища вивчення курсу основи алгоритмізації та програмування).

Обчислювальний експеримент – метод вивчення об'єктів та процесів за допомогою математичного моделювання. Він передбачає, що після побудови математичної моделі проводиться її чисельне дослідження, що дозволяє «програти» поведінку досліджуваного об'єкту в різних умовах або в різних модифікаціях [Самарський, 1979].

В ході обчислювального експерименту дослідник відкриває нові процеси і властивості, про які йому раніше нічого не було відомо.

Об'єктом дослідження є алгоритми сортування елементів масиву. Мабуть, жодна інша проблема не породила такої кількості різноманітних рішень, як задача сортування. Чи існує деякий "універсальний", найкращий алгоритм? Взагалі кажучи, ні. Проте, маючи приблизні характеристики вхідних даних, можна підібрати метод, що працює оптимальним чином.

Для того, щоб обґрунтовано зробити такий вибір, розглянемо параметри, по яких проводитиметься оцінка алгоритмів.

Час сортування – основний параметр, що характеризує швидкодію алгоритму.

Пам'ять – ряд алгоритмів вимагає виділення додаткової пам'яті під тимчасове зберігання даних. При оцінці пам'яті, що використовується, не враховуватиметься місце, яке займає початковий масив і незалежні від вхідної послідовності витрати, наприклад, на зберігання коду програми.

Стійкість – стійке сортування не міняє взаємного розташування рівних елементів. Така властивість може бути дуже корисною, якщо елементи складаються з декількох полів.

Природність поведінки – ефективність методу при обробці вже відсортованих, або частково відсортованих даних.

Мажорованість.

Для оцінки продуктивності алгоритмів можна використовувати різні підходи. Оцінити час виконання через символ $O(n)$ (читається так: О велике від n). n тут – як правило кількість оброблюваних елементів даних (наприклад, сортованих чисел).

Означення. $O(g(n))$ – множина функцій $f(n)$, для яких існують позитивні константи c, n_0 , такі що $f(n) \leq c * g(n)$ для всіх $n \geq n_0$.

- Проблема 1: Константа c може бути досить великою.
- Проблема 2: Чому дорівнює n_0 ?

Також таку оцінку називають «асимптотикою», оскільки вона позначає асимптотичну поведінку алгоритму ($n \rightarrow \infty$). Коли використовують позначення $O(\cdot)$, мають на увазі не точний час виконання, а тільки його границю зверху, причому з точністю до постійного множника. Коли говорять, наприклад, що алгоритму потрібний час порядку $O(n^2)$, мають на увазі, що час виконання задачі зростає не швидше, ніж квадрат кількості елементів. Щоб відчути, що це таке, погляньте на таблицю, де приведені числа, що ілюструють швидкість росту для декількох різних функцій.

n	$\log n$	$n * \log n$	n^2
1	0	0	1
16	4	64	256
256	8	2048	65536
4096	12	49152	16777216
65536	16	1048565	4294967296
1048476	20	20969520	1099301922576
16775616	24	402614784	281421292179456

Якщо вважати, що числа в таблиці відповідають мікросекундам, то для задачі з 1048476 елементами алгоритму з часом роботи $O(\log n)$ буде потрібно 20 мікросекунд, а алгоритму з часом роботи $O(n^2)$ – більше 12 днів.

Якщо обидва алгоритми, наприклад, $O(n * \log n)$, то це зовсім не означає, що вони однаково ефективні. Символ O не враховує константу, тобто перший може бути, скажімо в 1000 разів ефективніше. Це означає лише те, що час (або необхідна пам'ять або що-небудь ще, що треба оцінити) зростає приблизно з такою ж швидкістю, як функція $n * \log n$.

При *аналітичному* підході складність алгоритмів доводиться за допомогою строгих точних викладок. Проте на практиці студенти дуже важко сприймають цей матеріал.

Інший спосіб – запустити кожен алгоритм на декількох масивах даних і дослідити вказані характеристики (кількість порівнянь, кількість пересилань та час виконання), виявити масиви, для яких той чи інший алгоритм буде більш ефективним. Досить цікавим дослідженням для студентів є виявлення кількості елементів масиву, починаючи з якої один алгоритм має переваги над іншим.

Комп'ютерний експеримент дозволяє студенту зрозуміти особливості певних алгоритмів та усвідомити залежності, що пояснюють складність алгоритмів. Обчислювальний експеримент складається з планування, створення або вибору експериментальної установки і виконання контрольних випробувань. Потім слідує проведення серійних дослідів, обробка експериментальних даних та їх інтерпретація. Обчислювальний експеримент з вивчення ефективності алгоритмів проводиться за допомогою спеціально розробленого інтегрованого середовища. Таким чином можна говорити про обчислювальний експеримент як про нову технологію і методологію наукових і прикладних досліджень.



Створення ефективного програмного забезпечення дозволяє швидко проводити експерименти та багато разів варіювати параметри.

Переваги обчислювального експерименту очевидні. В обчислювальний експеримент можна легко і безпечно втрутатися. Його можна повторити і перервати у будь-який момент. В ході цього експерименту можна змоделювати необхідні умови.

Розробка програмного забезпечення для обчислювального експерименту в конкретній області діяльності приводить до створення програмного комплексу. Він складається із зв'язаних між собою прикладних програм і модулів, що включають засоби, які надаються користувачеві для управління ходом обчислювального експерименту, обробки і представлення його результатів. Такий комплекс програм іноді називають *проблемно-орієнтованим пакетом прикладних програм*.

При проведенні досліджень важливо пам'ятати, що обчислювальний експеримент має свої *обмеження*, які можуть привести до неефективних витрат часу і ресурсів, або навіть до отримання помилкових результатів.

Відомо, що застосовність результатів обчислювального експерименту *обмежена* рамками прийнятої математичної моделі.

Дійсно, обчислювальний експеримент не може повністю замінити аналітичні доведення, і майбутнє за їх розумним поєднанням.

ПОДЯКИ

Видання цієї книги було б неможливим без участі багатьох людей. Автори книги хотіли б висловити свою подяку всім, хто допоміг втілити проект у життя.

Особлива подяка Євгену Алфьорову за його вагомий внесок, підтримку та допомогу в підготовці цього видання.

А також всім викладачам і студентам факультету фізики, математики та інформатики Херсонського державного університету за допомогу в розробці, створенні та апробації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування» WebOAP.

ПОЯСНЮВАЛЬНА ЗАПИСКА

Курс “Основи алгоритмізації та програмування” є одним з провідних курсів професійної підготовки зі спеціальності Інформатика. Основна **мета курсу** полягає у формуванні фундаментальних понять, методів та ідіом програмування: поняття алгоритму, алгоритмічних конструкцій, структур даних, комп’ютерної програми, мови програмування, методології і технології програмування та методів їх застосування для розв’язання певних класів задач.

Програмування включає в себе: аналіз, проектування (розробку алгоритму), кодування і компіляцію (написання вихідного тексту програми та перетворення його у виконуваний код за допомогою компілятора), тестування та налагодження, супровождення.

Проте у даному курсі пропонується не просто вивчити лексичні конструкції мови програмування, а більш детально зупинитися на способах алгоритмізації та їх широкому застосуванню при розв’язані поставлених задач.

Завдання курсу:

- **Методичні**

- Розкрити значення програмування в загальній і професійній освіті людини, вплив технологій програмування на науково-технічний і соціально-економічний розвиток суспільства.
- Встановити міжпредметні зв’язки при викладанні дисципліни «Основи алгоритмізації та програмування».
- Показати розвиток мов та технологій програмування і необхідність постійного вдосконалення знань протягом всього життя для підвищення професійних компетенцій.
- Забезпечити студентам доступ до матеріалів курсу (підручників, навчальних і методичних посібників, інтегрованого середовища вивчення курсу).
- Звернути увагу студентів на аналіз складності алгоритмів та наочне представлення їх ефективності під час виконання обчислювального експерименту.

- Орієнтувати студентів у можливостях застосування отриманих знань у різних сферах, щоб майбутні програмісти могли застосовувати ці знання в практичній роботі.
- **Пізнавальні**
 - Сформувати погляд на програмування як на систематичну науково-практичну діяльність, що носить масовий характер (виробництво програм заданої якості у задані строки).
 - Сформувати базові теоретичні поняття, що лежать в основі процесу конструювання програм, у тому числі:
 - Заложити основи розробки ефективних програм, акцентуючи увагу на обчислювальному експерименті як основному методі аналізу складності та працездатності алгоритмів сортування та пошуку.
 - Заложити в основу конструювання і використання складних (динамічних) структур даних модель (парадигму) абстрактного типу даних.
 - Виховати у майбутніх програмістів творчий підхід до розв'язування практичних задач, формувати вміння і навички для самостійного аналізу та дослідження проблем.
 - Розвинути здатність і відчуття необхідності до постійної самоосвіти та самовдосконалення, наукового пошуку шляхів удосконалення процесу розробки алгоритмів та програм.
 - Розвинути та поглибити загальні уявлення про шляхи і перспективи глобальної інформатизації у всіх сферах людської діяльності.
 - Створити сприятливі умови для розвитку прагнення до наукового пошуку шляхів удосконалення своєї роботи, активізації пізнавальної діяльності, творчої активності, самостійного дослідницького пошуку нових знань. З цієї точки зору важливого значення набуває організація самостійної роботи студентів, їх участь у науково-дослідній роботі кафедр, самостійні дослідження у вигляді рефератів, курсових і дипломних робіт, участь у господарів різних науково-дослідних роботах, пропаганда серед населення сучасних засобів і методів збирання, зберігання, опрацювання, передавання, подання, використання інформації, роз'яснення їх впливу на розвиток інформаційного суспільства.

- **Практичні**

- Сформувати представлення про аналіз складності алгоритмів та програм.
- Сформувати навички технологій розробки коректних програм, (відносно) інваріантні до мови програмування високого рівня, що використовується.
- Навчити реалізації коректних програм на обраній робочій мові програмування Pascal з урахуванням особливостей її конкретної реалізації на комп’ютері (конкретної системи програмування – Borland Pascal).
- Продемонструвати теоретично і на практиці доцільність та можливість конструктивного використання базових теоретичних понять, методів та прийомів (абстрактних схем) програмування.
- Ознайомити студентів з аналізом складності алгоритмів сортування й пошуку та наочним представленням їх ефективності під час виконання обчислювального експерименту засобами інтегрованого середовища навчання.
- Забезпечити формування алгоритмічного стилю мислення та вміння комп’ютерної реалізації складних програмних систем.

Вимоги до рівня освоєння змісту дисципліни

студент повинен знати:

- поняття алгоритму, базові структури алгоритмів;
- базові поняття програмування: концепцію типу, операції, оператори, принципи та правила їх застосування;
- технологій програмування;
- суть процедурного підходу до проектування програм;
- поняття функції та способи передачі параметрів;
- принципи організації та застосування складених структур даних: масивів, структур, об’єднань;
- принципи управління пам’яттю за допомогою вказівників та застосування динамічних змінних;
- поняття, структуру та правила виконання рекурсивних алгоритмів;
- принципи та методи роботи з файлами;
- призначення, організацію та способи реалізації зв’язаних структур даних;

- поняття, структуру та основні принципи застосування алгоритмів сортування та пошуку;
- поняття складності алгоритму, мажорованості, обчислювального експерименту.

Студент повинен вміти:

- складати алгоритми, використовуючи базові структури;
- реалізовувати алгоритми структурними програмами;
- ефективно вибирати типи та структури даних для зберігання інформації;
- структурувати задачу за допомогою функцій;
- використовувати вказівники для динамічного управління пам'яттю;
- використовувати зв'язані структури даних для роботи з інформацією;
- застосовувати операції роботи з файлами як на стандартному, так й на системному рівні;
- розробляти алгоритми розв'язання невеликих задач і проектувати невеликі прикладні програмні системи.

Міждисциплінарні зв'язки:

- комп'ютерні інформаційні технології;
- алгебра;
- геометрія;
- математичний аналіз;
- фізика.

Тематичний план лекцій та практичних робіт з розділу «Алгоритми пошуку та сортування»

№	НАЗВА ТЕМИ	КІЛЬКІСТЬ ГОДИН
1.	Лекція № 1. Алгоритми пошуку	2
2.	Практична робота № 1. Лінійний пошук	2
3.	Практична робота № 2. Бінарний пошук в упорядкованому масиві	2
4.	Лекція № 2. Прості алгоритми сортування	2

5.	Практична робота № 3. Сортування обмінами	2
6.	Практична робота № 4. Покращене сортування обмінами	2
7.	Практична робота № 5. Сортування вибором	2
8.	Практична робота № 6. Сортування вставками	2
9.	Практична робота № 7. Сортування Шелла	2
10.	Практична робота № 8. Аналіз ефективності простих алгоритмів сортування масивів	2
11.	Лекція № 3. Швидкі алгоритми сортування та пошуку	2
12.	Практична робота № 9. Сортування злиттям	2
13.	Практична робота № 10. Швидке сортування Хоара	2
14.	Практична робота № 11. Піраміdalne сортування	2
15.	Практична робота № 12. Пошук k-го елемента в масиві	2
16.	Практична робота № 13. Дослідження роботи алгоритмів сортування на масивах різної довжини	2
	Всього	32

Список рекомендованих джерел.

1. Архангельский А.Я. Язык Pascal и основы программирования в Delphi. – М.: Бином, 2008. – 496 с.
2. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов.– М.: Мир, 1979.– 536 с.
3. Венц А. Н. Профессия – программист.– Ростов: Феникс, 1999.– 384 с.
4. Вilenkin N.Ya. Комбинаторика.– М.: Наука, 1969.– 328 с.
5. Виноградов И.М. Основы теории чисел.– М.: Наука, 1972.– 167 с.
6. Вирт Н. Алгоритмы + структуры данных = программы.– М.: Мир, 1985 г. 406 с.
7. Вирт Н. Алгоритмы и структуры данных. – М.: Мир, 1989. – 420 с.
8. Гусева А.И. Учимся информатике: задачи и методы решения.– М.: Диалог – МИФИ, 1998.– 320 с.

9. Кнут Д. Искусство программирования, том 3. Сортировка и поиск – М.: Вильямс, 2007. – 800 с.
10. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы. Построение и анализ.– М.: МЦНМО, 1999.– 960 с.
11. Культин Н.Б. Turbo Pascal в задачах и примерах. – СПб.: БХВ-Петербург, 2006. – 256 с.
12. Левитин А.В. Алгоритмы: введение в разработку и анализ. – М.: Вильямс, 2006. – 576 с.
13. Львов М.С., Спиваковский А.В., Белоусова С.В. Основы программирования на языке Паскаль. Херсон: МИБ, 1997.– 153 с.
14. Львов М.С., Співаковський О.В. Основи алгоритмізації та програмування. – Херсон: Айлант, 2000. – 214 с.
15. Марченко А.И., Марченко Л.А. Программирование в среде Turbo Pascal 7.0. – М.: Корона-век, 2007. – 464 с.
16. Окулов С.М. Основы программирования. – М.: ЮНИМЕДИАСТАЙЛ, 2002.– 424 с.
17. Павловская Т.А. Паскаль. Программирование на языке высокого уровня. – СПб.: Питер, 2003. – 320 с.
18. Пильщиков В.Н. Сборник упражнений по языку Паскаль. – М.: Наука, 1989. – 160 с.
19. Пратт С. Язык программирования C++. Лекции и упражнения. – СПб.: ДиаСофт. – 2003.
20. Фаронов В.В. Turbo Pascal 7.0. Начальный курс. – Учебное пособие. – М.: ОМД Групп, 2003 г. – 616 с.
21. Фаронов В.В. Turbo Pascal 7.0. Практика программирования. – М.: КноРус, 2009. – 416 с.
22. Фаронов В.В. Turbo Pascal 7.0. Учебный курс. – М.: КноРус, 2009. – 368 с.

ПРАКТИЧНА РОБОТА №1

Тема: Лінійний пошук

Мета: Навчитися застосовувати алгоритм лінійного пошуку при розв'язанні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Задача пошуку елемента у послідовності – одна з найважливіших задач програмування як з теоретичної, так і з практичної точки зору.

Нехай $A = \{a_1, a_2, \dots\}$ – послідовність однотипних елементів і b – деякий елемент, що має властивість P . Необхідно знайти місце елемента b у послідовності A .

Оскільки представлення послідовності у пам'яті може бути здійснено у вигляді масиву, то задачі можуть бути уточнені як задачі пошуку елемента у масиві A .

Наприклад:

1. Знайти максимальний елемент масива;
2. Знайти даний елемент масива;
3. Знайти k -тий за величиною елемент масива.

Найбільш прості і оптимальні алгоритми засновано на послідовному перегляді масиву A з перевіркою властивості P на кожному елементі. Цей метод називають лінійним переглядом (пошуком).

Лінійний, або послідовний пошук – найпростіший алгоритм пошуку, який дозволяє знаходити задане значення на деякому відрізку. Цей алгоритм не накладає обмежень на значення, яке потрібно знайти.

Пошук виконується порівнянням заданого значення та значення поточного елемента масиву до тих пір, поки вони не співпадуть.

Якщо масив має довжину n , то складність алгоритму буде $O(n)$.

Недоліки методу:

- низька ефективність (використовується при невеликому розмірі масиву).

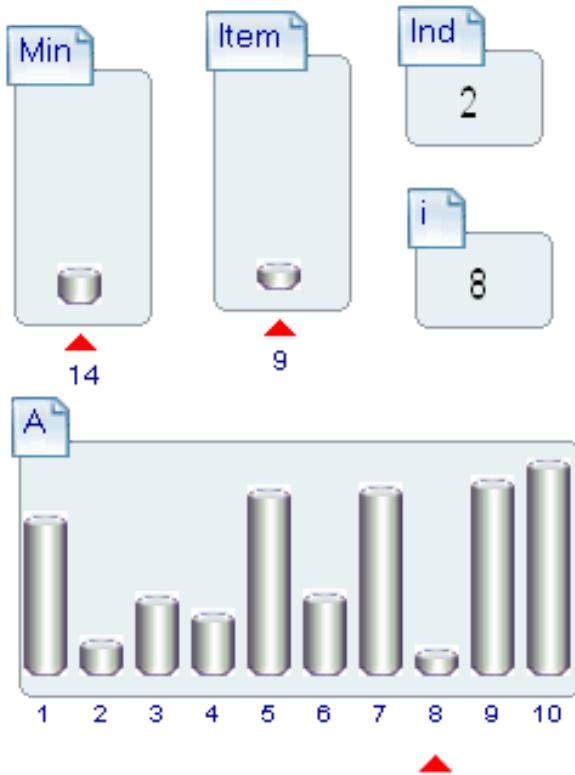
Переваги методу:

- не потребує додаткової пам'яті;
- може працювати у потоковому режимі (безпосереднє отримання даних з будь-якого джерела);

Приклад 1. Пошук мінімального елемента у масиві.

```
Program MinItem;
Const
  n = 20;
Var
  A : Array [1..n] of Data;
  Min, Item : Data;
  Ind, i : Integer;

Begin
  Ind := 1;
  Min := A[1];
  For i := 1 to n do begin
    Item := A[i];
    If Min > Item
      then begin
        Ind := i;
        Min := Item
      end
    end;
  End.
End.
```



Для наочного представлення масиву корисно зображувати його в системі координат *Індекс – Величина елемента*.

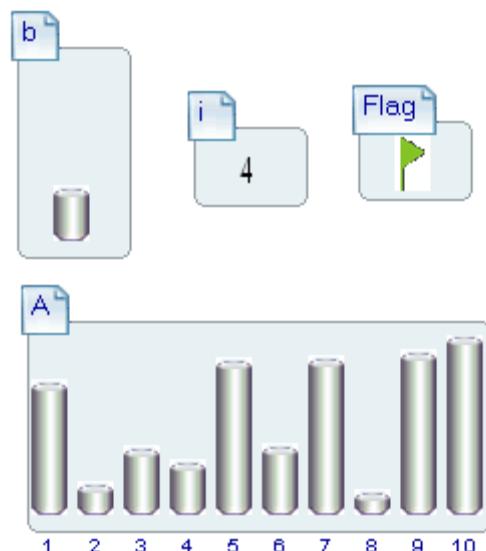
Min – поточне значення шуканого елемента. *Ind* – його номер у масиві. Перед першим кроком перегляду $Min = A[1]$, а $Ind = 1$. На i -тому кроці $A[i]$ порівнюється з min . Якщо $A[i] \geq min$, значення min не змінюється. На рисунку зображено ситуацію, у якій на i -тому кроці перегляду $A[i] < min$, тому min приймає значення $A[i]$, а Ind – значення i . Перегляд закінчується через n кроків. Упевнітесь, що алгоритм закінчує роботу, коли кількість порівнянь дорівнює n .

Змінна *Item* введена для того, щоб уникнути повторних обчислень індексного виразу $A[i]$.

Приклад 2. Пошук даного елемента у масиві з використанням мітки.

```
Program Search_in_Array;
Label 1;
Const
  n = 10;
Var
  A : Array [1..n] of Data;
  b : Data;
  Flag : Boolean;
  i : Integer;
  Found:boolean;

Begin
  Flag := false;
  For i := 1 to n do
    If A[i] = b
      then begin
        Flag := true;
        goto 1
      end;
  1:If Flag
    then Found:=true
    else Found:=false
End.
```



Прапорець *Flag* показує, чи знайдено даний елемент *b* в масиві. Так як перед початком перегляду масиву елемент *b* ще не знайдено, то *Flag=false*. Здійснюється лінійний перегляд елементів масива. На *i*-тому кроці *A[i]* порівнюється з *b*. Якщо значення поточного елемента дорівнює значенню *b*, то значення змінної *Flag* змінюється на *true*, та перегляд масиву завершується (вихід з циклу за допомогою мітки). Якщо елемент знайдено, то змінна *i* приймає значення номеру знайденого елемента.

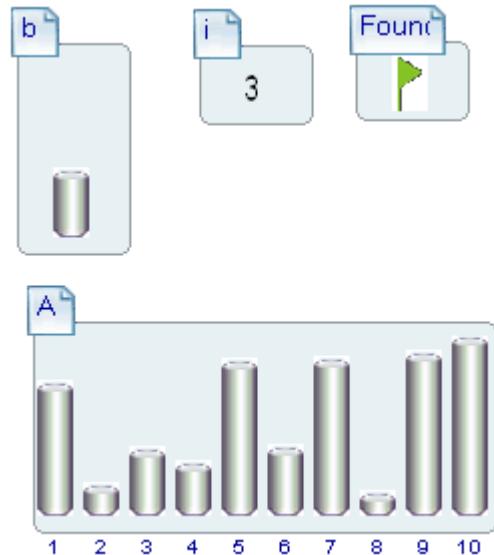
На рисунку зображено ситуацію, коли шуканим є 4-й елемент масиву. Зверніть увагу, що при виконанні алгоритма зроблено 4 порівняння.

Наведіть найкращий та найгірший приклади вхідних даних для алгоритму пошуку. Скільки порівнянь буде виконано у кожному випадку?

Приклад 3. Пошук даного елемента у масиві з використанням циклу while.

Використання оператора Goto вважається негарним стилем у програмуванні. Тому розглянемо розв'язок цієї задачі без застосування міток. Використаємо цикл While замість застосування цикла For з достроковим виходом за допомогою Goto.

```
Program WhileSearch;
Const
  n = 10;
Var
  A : Array[1..n] of Data;
  b : Data;
  i : Integer;
  Found:Boolean;
Begin
  i := 1;
  While (i <= n) and (A[i] <> b)
    i := Succ(i);
  If i = n + 1
    then Found:=false
    else Found:=true
End.
```



На рисунку зображено ситуацію, коли шуканим є 3-й елемент масиву. Зверніть увагу, що при виконанні алгоритма зроблено 3 порівняння (під порівнянням розуміємо порівняння значень елементів, а не індексів).

Перевірте роботу алгоритма, коли шуканий елемент є першим, останнім елементом масиву, знаходиться всередині масиву, відсутній у масиві. Слідкуйте за кількістю порівнянь. Зробіть висновки стосовно недоліків запропонованого алгоритму. Як їх можна усунути?

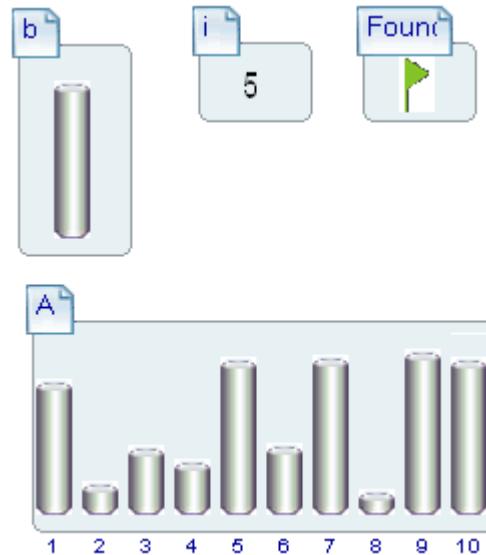
Недоліком такої реалізації є, по-перше, використання складеної умови в циклі, по-друге, можливо, некоректне обчислення підвиразу $A[i] <> b$ при $i = n + 1$. Недивлячись на те, що умова $(i \leq n)$ хибна, вираз $A[n+1] <> b$ буде обчислюватися, притому, що $A[n+1]$ невизначено!

Доповнимо масив A ще одним елементом: $A[n+1] = b$, і всі недоліки легко усуваються!

Приклад 4. Пошук даного елемента у масиві з використанням бар'єрного елементу.

```
Program LinearSearch;
Const
  n = 10;
Var
  A : Array[1..n] of Data;
  b : Data;
  i : Integer;
  Found:Boolean;

Begin
  i := 1;
  A[n] := b;
  While A[i] <> b do
    i := Succ(i);
  If i = n
    then Found:=false
    else Found:=true
End.
```



b – елемент, місце якого необхідно знайти. Перед 1-м кроком перегляду у масиві встановлюють бар'єр: $A[n] := b$. На *i*-тому кроці перегляду $A[i]$ порівнюється з *b*. Алгоритм закінчує роботу при $A[i] = b$. Якщо при цьому виявиться, що $i = n$, шуканий елемент у масиві відсутній. У протилежному випадку *i* – його місце. Наявність бар'єра гарантує коректне завершення цикла.

На рисунку зображене ситуацію, коли шуканим є 5-й елемент масиву. Зверніть увагу, що при виконанні алгоритма зроблено 5 порівнянь.

Перевірте роботу алгоритма, коли шуканий елемент є останнім елементом масиву, відсутній у масиві. Слідкуйте за кількістю порівнянь.

Література:

1. М.С. Львов, А. В. Спиваковский, С. В. Белоусова. Основы программирования на языке Паскаль. Херсон: МИБ, 1997.– 153 с.
2. С.М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.
3. Д. Кнут. Искусство программирования, том 3. Сортировка и поиск – М.: Вильямс, 2007. – 824 с.

Задачі для самостійного розв'язування

1. Розгляньте поведінку алгоритма **Пошук мінімального елемента у масиві** у випадку, коли мінімальних елементів у масиві декілька. Який з мінімальних елементів знайдено? Скільки порівнянь необхідно для виконання алгоритму?
2. Розробіть модифікацію алгоритма **Пошук мінімального елемента у масиві** у випадку, коли мінімальних елементів у масиві кілька і необхідно знайти останній мінімальний елемент. Скільки порівнянь необхідно для виконання алгоритму?
3. Розгляньте поведінку алгоритма **Пошук даного елемента у масиві з використанням мітки**. За допомогою середовища демонстрації перевірте, у яких випадках кількість порівнянь при виконанні алгоритму буде дорівнювати кількості елементів масиву.
4. Розробіть модифікацію алгоритма **Пошук даного елемента у масиві з використанням мітки** у випадку, коли елементів, рівних даному, у масиві кілька і необхідно знайти останній з них. Скільки порівнянь необхідно для виконання алгоритму? Розробіть алгоритм з мінімальною кількістю порівнянь.
5. Перевірте роботу алгоритма **Пошук даного елемента у масиві з використанням циклу while**, коли шуканий елемент є першим, останнім елементом масиву, знаходиться всередині масиву, відсутній у масиві. Слідкуйте за кількістю порівнянь. Зробіть висновки стосовно недоліків запропонованого алгоритму. Як їх можна усунути?
6. Розгляньте поведінку алгоритма **Пошук даного елемента у масиві з використанням бар'єрного елементу**. За допомогою середовища демонстрації перевірте, у яких випадках кількість порівнянь при виконанні алгоритму буде дорівнювати кількості елементів масиву.
7. Розробіть модифікацію алгоритма **Пошук даного елемента у масиві з використанням бар'єрного елементу** у випадку, коли елементів, рівних даному, у масиві декілька і необхідно знайти останній з них. Скільки порівнянь необхідно для виконання алгоритму? Розробіть алгоритм з мінімальною кількістю порівнянь.

8. Використовуючи лінійний пошук, знайдіть максимальний елемент у масиві A з 30, 50 та 100 елементів, розташованих хвилюю. Визначте кількість порівнянь за допомогою середовища демонстрації та порівняйте ефективність на кожному з масивів. Побудуйте графік залежності кількості порівнянь від кількості елементів масиву у Excel. Отримайте оцінку кількості $C(n)$ порівнянь метода.
9. Використовуючи лінійний пошук, знайдіть 5-ий за величиною елемент b у масиві A з 10, 20 та 50 елементів, розташованих довільним чином. Визначте кількість порівнянь за допомогою середовища демонстрації та порівняйте ефективність на кожному з масивів. Побудуйте графік залежності кількості порівнянь від кількості елементів масиву у Excel. Отримайте оцінку кількості $C(n)$ порівнянь метода.
10. Використовуючи лінійний пошук, знайдіть елемент b у масиві A з 20, 50 та 100 елементів, розташованих за зростанням. Визначте кількість порівнянь за допомогою середовища демонстрації та порівняйте ефективність на кожному з масивів. Побудуйте графік залежності кількості порівнянь від кількості елементів масиву у Excel. Отримайте оцінку кількості $C(n)$ порівнянь метода.

ПРАКТИЧНА РОБОТА №2

Тема: Бінарний пошук в упорядкованому масиві.

Мета: Навчитися застосовувати алгоритм бінарного пошуку при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Бінарний, або двійковий пошук – алгоритм пошуку елементу у відсортованому масиві. Це класичний алгоритм, ще відомий як метод дихотомії (ділення навпіл).

Якщо елементи масиву впорядковані, задача пошуку суттєво спрощується. Згадайте, наприклад, як Ви шукаєте слово у словнику. Стандартний метод пошуку в упорядкованому масиві – це метод поділу відрізка навпіл, причому відрізком є відрізок індексів $1..n$. Дійсно, нехай масив A впорядкований за зростанням і m ($k < m < l$) – деякий індекс. Нехай $Buffer = A[m]$. Тоді якщо $Buffer > b$, далі елемент необхідно шукати на відрізку $k..m-1$, а якщо $Buffer < b$ – на відрізку $m+1..l$.

Для того, щоб збалансувати кількість обчислень в тому і іншому випадку, індекс m необхідно обирати так, щоб довжина відрізків $k..m$, $m..l$ була (приблизно) рівною. Описану стратегію пошуку називають *бінарним пошуком*.

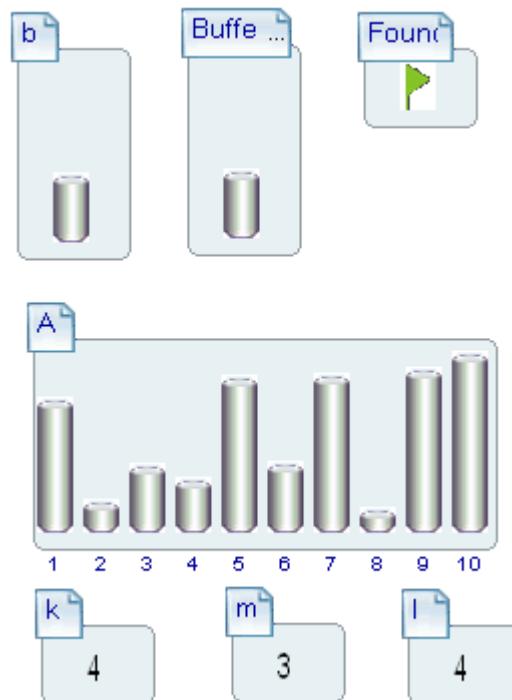
b – елемент, місце якого необхідно знайти. Крок бінарного пошуку полягає у порівнянні шуканого елемента з середнім елементом $Buffer = A[m]$ в діапазоні пошуку $[k..l]$. Алгоритм закінчує роботу при $Buffer = b$ (тоді m – шуканий індекс). Якщо $Buffer > b$, пошук продовжується ліворуч від m , а якщо $Buffer < b$ – праворуч від m . При $l < k$ пошук закінчується, і елемент не знайдено.

```

Program BinarySearch;
Const
  n = 10;
Var
  A : Array[1..n] of Data;
  b, Buffer : Data;
  k, l, m : Integer;
  Found:Boolean;

Begin
  k := 1;
  l := n;
  Repeat
    m := (k + l) div 2;
    Buffer := A[m];
    If Buffer > b
      then l := Pred(m)
      else k := Succ(m)
  Until (Buffer = b) or (l < k);
  If A[m] = b
    then Found:=true
    else Found:=false
End.

```



Література:

1. М.С. Львов, А. В. Спиваковский, С. В. Белоусова. Основы программирования на языке Паскаль. – Херсон: МИБ, 1997. – 153 с.
2. С.М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.
3. А.В. Левитин. Алгоритмы: введение в разработку и анализ. — М.: «Вильямс», 2006. – 576 с.

Задачі для самостійного розв'язування

1. Використовуючи середовище демонстрації, для одного і того ж упорядкованого за зростанням масиву A доберіть значення b , такі, щоб отримати найменшу та найбільшу кількості порівнянь для алгоритму бінарного пошуку.
2. Розробіть алгоритм бінарного пошуку для масиву, упорядкованого за спаданням. Використовуючи середовище демонстрації, для одного і того ж упорядкованого за спаданням

масиву A доберіть значення b , такі, щоб отримати найменшу та найбільшу кількості порівнянь для алгоритму бінарного пошуку.

3. Розробіть алгоритм лінійного пошуку для масиву, упорядкованого за спаданням. Використовуючи середовище демонстрації, для одного і того ж упорядкованого за спаданням масиву A доберіть значення b , такі, щоб отримати найменшу та найбільшу кількості порівнянь для алгоритму лінійного пошуку.
4. Використовуючи дані, отримані в результаті досліджень, виконаних у завданнях 2-3, зробіть висновки щодо ефективності алгоритмів лінійного та бінарного пошуку.
5. Використовуючи алгоритм бінарного пошуку, знайдіть елемент b у масиві A з кількістю елементів від 10 до 100, розташованих за зростанням. Визначте кількість порівнянь за допомогою середовища демонстрації та порівняйте ефективність на кожному з масивів. Побудуйте графік залежності кількості порівнянь від кількості елементів масиву у Excel. Побудуйте у тій же системі координат графіки функцій $y=n$ та $y=\log_2(n)$. Дослідивши графіки, зробіть оцінку кількості $C(n)$ порівнянь алгоритма бінарного пошуку.
6. Порівняйте отриману експериментально оцінку кількості $C(n)$ порівнянь алгоритма бінарного пошуку з теоретичною оцінкою. Нехай M – кількість повторень цикла. Тоді $M \leq \lceil \log_2(n) \rceil$. Оскільки на кожному кроці здійснюється 2 порівняння, у найгіршому випадку

$$C(n) = 2\lceil \log_2 n \rceil = O(\log_2 n).$$

При $n = 1000000$ алгоритм здійснює максимум 40 порівнянь, у той час коли лінійний перегляд потребує всіх 1000000 порівнянь.

ПРАКТИЧНА РОБОТА №3

Тема: Сортування обмінами

Мета: Навчитися застосовувати сортування обмінами (бульбашкове) при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Під сортуванням послідовності розуміють процес перестановки елементів послідовності у певному порядку. Мета такого впорядкування – полегшення подальшої обробки даних (наприклад, задачі пошуку). Тому задача сортування – одна з найбільш важливих внутрішніх задач програмування.

Цікаво, що сортування є ідеальним прикладом великої кількості різних алгоритмів, розв'язку однієї і тієї ж задачі. Розглядаючи різні методи сортування, ми побачимо, як зміни алгоритма призводять до нових, більш ефективних розв'язків.

Якщо послідовність a_1, a_2, \dots, a_n реалізована як масив $a[1..n]$, вся вона розташована в адресній пам'яті. Тому поряд з вимогою ефективності за часом основна вимога – економне використання пам'яті. Це означає, що алгоритм не повинен використовувати додаткових масивів і пересилань з масива a у ці масиви.

Постановка задачі сортування у загальному вигляді передбачає, що існують тільки два типи дій з даними типу, що сортуємо: порівняння двох елементів ($x << y$) і пересилання елемента ($x := y$). Тому зручна міра складності алгоритма сортування масива $a[1..n]$ за часом – кількість порівнянь $C(n)$ і кількість пересилань $M(n)$.

Відомі алгоритми сортування масивів діляться на прості і швидкі (ефективні).

Сортування обмінами, або бульбашкове сортування (англ. *bubble sort*) – алгоритм простого сортування.

Перевагою даного виду сортування є його простота у розумінні та реалізації, але у той самий час недоліком є ефективність лише на масивах невеликої довжини. Складність алгоритму: $O(n^2)$.

Нехай необхідно відсортувати масив $a[1..n]$ за зростанням, тобто переставити елементи масиву у такому порядку, щоб для кожних $i < j$ виконувалась нерівність $a[i] < a[j]$.

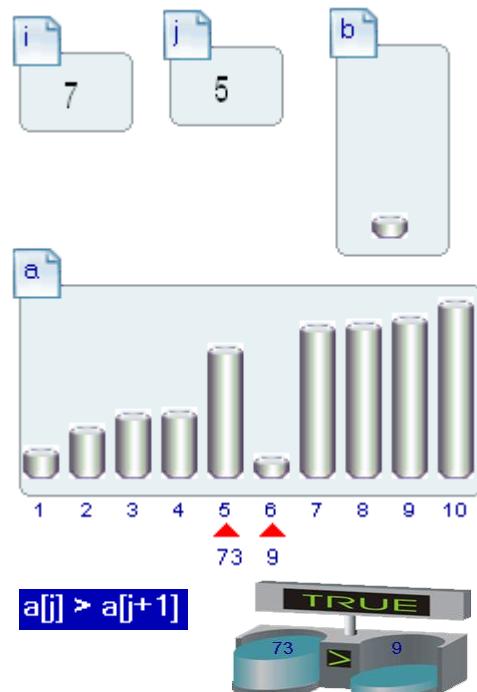
Основна дія сортування обмінами – порівняння двох елементів i , якщо результат порівняння негативний, перестановка їх місцями:

Якщо при $i < j$ $a[i] > a[j]$, то переставити $a[i]$ і $a[j]$.

У найбільш простому варіанті порівнюються елементи, що стоять поруч, $a[j]$ і $a[j+1]$:

```
Program BublSort;
Const
  n = 10;
Var
  a : array[1..n] of Data;
  i, j : Integer;
  b : Data;

Begin
  For i := n - 1 downto 1 do
    For j := 1 to i do
      If a[j] > a[j+1]
        then begin
          b := a[j+1];
          a[j+1] := a[j];
          a[j] := b
        end;
End.
```



Алгоритм BublSort здійснює $n-1$ лінійний перегляд масива a . У результаті кожного перегляду найбільший елемент переглянутої частини масива «спливає» на останнє місце. Перший перегляд здійснюється у всьому масиві, а кожен наступний – у діапазоні, на одиницю меншому. Таким чином, права частина масива стає впорядкованою. На рисунку це елементи – від $a[i]$ до $a[n]$. На кожному кроці перегляду порівнюються два елементи, що стоять поруч. На рисунку – це елементи $a[j]$ і $a[j+1]$. Якщо вони розташовані невірно (засортування), алгоритм міняє їх місцями.

Аналіз алгоритма сортування обмінами.

Аналіз алгоритма полягає в обґрунтуванні його властивостей. Найважливішими властивостями алгоритма є: коректність (правильність), оцінка складності за часом і пам'яттю, а також деякі інші властивості.

Для обґрунтування алгоритма сортування необхідно довести, що алгоритм завжди (незалежно від входу) завершує свою роботу і його результат – упорядкований за зростанням масив. Оцінка складності (ефективності) алгоритма за часом полягає у знаходженні $C(n)$, $M(n)$ у найгіршому випадку, в середньому. Оскільки алгоритм сортує масив «на місці», його складність по пам'яті – константа.

До інших властивостей алгоритма можна віднести властивість стійкості. (Алгоритм сортування називається стійким, якщо він не переставляє місцями рівні елементи.) Здійснимо аналіз алгоритма сортування обмінами.

Завершуваність. Алгоритм BublSort завжди завершує свою роботу, оскільки він використовує тільки цикли з параметром, і в тілі циклів параметри примусово не змінюються.

Коректність. Внутрішній цикл алгоритма (з параметром j) здійснює послідовний перегляд перших i елементів масива. На кожному кроці перегляду порівнюються i , якщо це необхідно, переставляються два сусідніх елемента. Таким чином, найбільший серед перших $i+1$ елементів «спливає» на $i+1$ -е місце.

За допомогою демонстрації роботи алгоритма упевніться, що після виконання оператора *If* має місце співвідношення:

$$a[j] = \text{Max}(a[1], \dots, a[j])$$

а після завершення цикла ($i = j$) має місце співвідношення

$$a[i+1] = \text{Max}(a[1], \dots, a[i], a[i+1]).$$

Зовнішній цикл (з параметром i) керує довжиною частини масива, що розглядається: вона зменшується на 1. Тому після завершення внутрішнього цикла має місце співвідношення:

$$a[i+1] \leq a[i+2] \leq \dots \leq a[n]$$

Після завершення зовнішнього цикла отримаємо:

$$a[1] \leq a[2], a[2] \leq a[3] \leq \dots \leq a[n]$$

тобто масив відсортовано.

Ефективність за часом. Зовнішній цикл виконався $n-1$ раз. Внутрішній цикл виконується i раз ($i = n-1, n-2, \dots, 1$). Кожне виконання тіла внутрішнього цикла полягає в одному порівнянні і, можливо, в одній перестановці. Тому

$$C(n) = 1+2+\dots+n-1 = n*(n-1)/2, \quad M(n) \leq n*(n-1)/2.$$

У найгіршому випадку, коли елементи вихідного масива розташовані в порядку спадання

$$C(n) = n*(n-1)/2 = O(n^2), \quad M(n) = n*(n-1)/2 = O(n^2).$$

Стійкість. Для доведення стійкості достатньо зауважити, що переставляються тільки сусідні нерівні елементи.

Література:

1. М.С. Львов, А.В. Спиваковский, С.В. Белоусова. Основы программирования на языке Паскаль. Херсон: МИБ, 1997. – 153 с.
2. С.М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.

Задачі для самостійного розв'язування

1. На малюнку зображено деякий крок сортування масиву обмінами за зростанням. Яка частина масиву відсортована? Які елементи порівнюються в даний момент? Чи буде здійснюватися обмін?



2. Використовуючи алгоритм сортування обмінами, співставте кількість перестановок та порівнянь на масивах від 10 до 100 елементів (масиви задаються автоматично) за допомогою колекції даних середовища демонстрації. Експортуйте дані в Excel та побудуйте графіки:

- а) залежності кількості порівнянь від кількості елементів масиву;

б) залежності кількості пересилань від кількості елементів масиву.

Порівняйте отримані графіки з графіками функцій $y=n$; $y=n^2$; $y=\log_2(n)$. Зробіть висновки щодо ефективності алгоритмів.

3. Перевірте у середовищі демонстрації, чи виконається сортування, якщо у внутрішньому циклі j змінювати від 1 до $n-1$. Співставте кількість порівнянь та перестановок з даними завдання 2. Зробіть висновок щодо ефективності алгоритмів.
4. Змініть алгоритм сортування обмінами таким чином, щоб сортування масиву відбувалося за спаданням і за кожен лінійний перегляд масиву більший елемент з частини масиву, що розглядається на даному кроці, «спливав» на початок. Візуалізуйте роботу алгоритму у середовищі демонстрації. Яка частина масиву буде відсортована після 3-х лінійних переглядів?
5. Виконайте алгоритм сортування обмінами за зростанням у середовищі демонстрації для наступних масивів: відсортованого за спаданням з 10 елементів, відсортованого за зростанням з 10 елементів. Знайдіть кількість порівнянь та пересилань у кожному випадку.
6. Як змінити алгоритм сортування обмінами, щоб він припиняв перегляд елементів у випадку, коли масив вже відсортовано? Яку мінімальну кількість порівнянь необхідно виконати, щоб упевнитись, що масив відсортовано?

ПРАКТИЧНА РОБОТА №4

Тема: Покращене сортування обмінами

Мета: Навчитися підвищувати ефективність сортування обмінами.

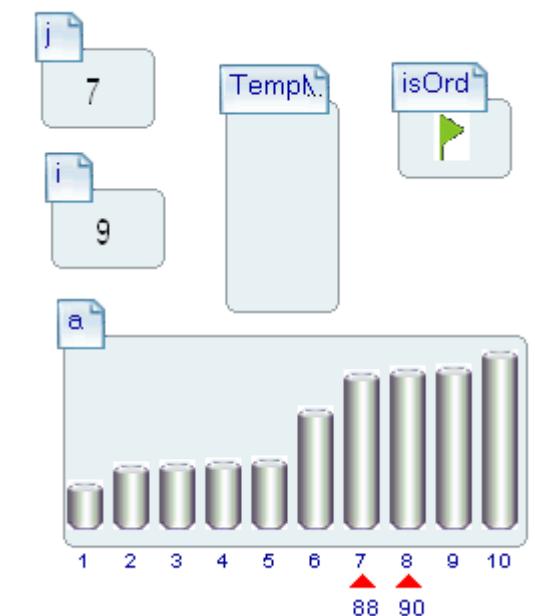
Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

В алгоритмі сортування обмінами обов'язково виконувалося $n-1$ лінійних переглядів, навіть якщо масив уже впорядкований після кількох перших проходів. Від зайніших проходів можна позбавитися, застосувавши замість зовнішнього циклу *for*, який виконується рівно $n-1$ раз, цикл *While* або *Repeat-Until* і перевіряючи у внутрішньому циклі масив на впорядкованість:

```
Program BublSort1v;
Const
  n = 10;
Var
  a : array[1..n] of Data
  i, j : Integer;
  TempMem : Data;
  isOrd : Boolean;

Begin
  i := n - 1;
  Repeat
    isOrd := True;
    For j := 1 to i do
      If a[j] > a[j+1]
        then begin
          TempMem := a[j+1];
          a[j+1] := a[j];
          a[j] := TempMem;
          isOrd := False
        end;
    i := Pred(i)
  Until isOrd
End.
```



Література:

1. М.С. Львов, А. В. Спиваковский, С. В. Белоусова. Основы программирования на языке Паскаль. Херсон: МИБ, 1997. – 153 с.
2. С.М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.
3. Д. Кнут. Искусство программирования, том 3. Сортировка и поиск – М.: Вильямс, 2007. – 800 с.

Задачі для самостійного розв'язування

1. Розробіть удосконалений алгоритм сортування обмінами за спаданням. Виконайте алгоритм у середовищі демонстрації для масивів з 10 елементів та знайдіть кількість порівнянь та пересилань:

- а) упорядкованого за зростанням;
- б) упорядкованого за спаданням;
- в) вверх-вниз;
- г) заповненого випадково.

Зробіть висновки, для яких масивів розроблений алгоритм є більш ефективним.

2. Алгоритм сортування обмінами можна ще покращити, якщо кожний наступний прохід починати не з початку ($j = 1$), а з того місця, де на попередньому проході відбувся перший обмін. Розробіть модифікований алгоритм сортування обмінами. Порівняйте його ефективність з алгоритмом, запропонованим у колекції. Для цього виконайте кожен алгоритм у середовищі демонстрації для масивів з 30 елементів:

- а) упорядкованого за зростанням;
- б) упорядкованого за спаданням;
- в) вверх-вниз;
- г) заповненого випадково.

Знайдіть середню кількість порівнянь і пересилань для кожного алгоритму. Зробіть висновок щодо ефективності.

3. Розробіть модифікований алгоритм сортування обмінами, у якому усувається асиметрія, тобто проходи по масиву чергуються вперед-назад. Порівняйте його ефективність з раніше розробленими.

ПРАКТИЧНА РОБОТА №5

Тема: Сортування вибором

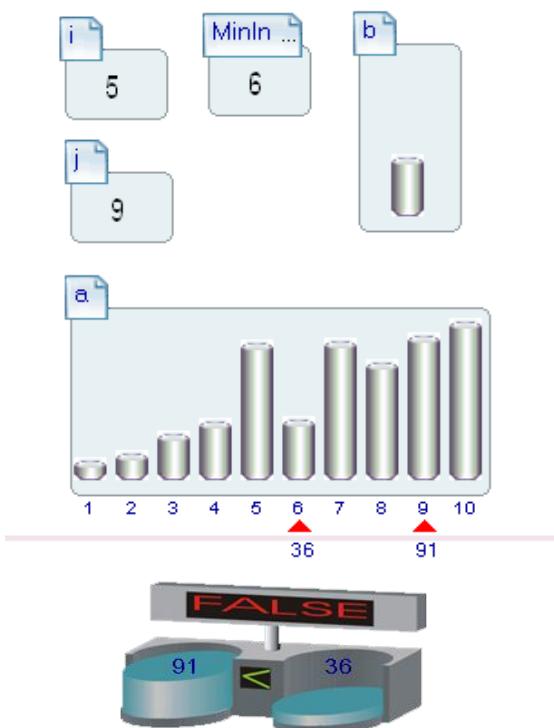
Мета: Навчитися застосовувати метод сортування вибором при розв'язуванні задач. Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Основна дія сортування вибором – пошук найменшого елемента у частині масива, що розглядаємо, і перестановка з першим елементом цієї частини:

```
For i := 1 to n - 1 do begin  
    k := Індекс(Min(a[i], ..., a[n]));  
    Переставити a[i], a[k]  
end;
```

```
Program SelectSort;  
Const  
    n = 10;  
Var  
    a : array[1..n] of Data;  
    i, j, MinInd : Integer;  
    b : Data;  
Begin  
    For i := 1 to n - 1 do begin  
        MinInd := i;  
        For j := i + 1 to n do  
            If a[j] < a[MinInd]  
                then MinInd := j;  
        b := a[MinInd];  
        a[MinInd] := a[i];  
        a[i] := b  
    end;  
End.
```



Алгоритм сортування вибором здійснює $n-1$ лінійний перегляд масива a . В результаті кожного перегляду найменший елемент розглянутої частини масива міняється місцями з першим елементом цієї частини. Перший перегляд здійснюється в усьому масиві, а

кожний наступний – в діапазоні, на одиницю меншому. Таким чином, ліва частина масива стає впорядкованою. На рисунку це елементи – від $a[1]$ до $a[i-1]$ ($i = 5$). Переставлятися на поточному кроці будуть елементи $a[i]$ і $a[MinInd]$ (на рисунку $a[5]$ і $a[6]$).

Література:

1. М. С. Львов, А. В. Спиваковский, С. В. Белоусова. Основы программирования на языке Паскаль. Херсон: МИБ, 1997. – 153 с.
2. С. М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.

Задачі для самостійного розв'язування

1. Виконайте у середовищі демонстрації алгоритм сортування вибором для масивів з кількістю елементів від 10 до 100. Побудуйте графіки залежності кількості порівнянь та кількості пересилань від кількості елементів масиву. Порівняйте отримані графіки з графіками функцій $y=n$; $y=n^2$; $y=\log_2(n)$. Зробіть висновки щодо ефективності алгоритмів.
2. Виконайте алгоритм сортування вибором у середовищі демонстрації для масиву з 10 елементів. Визначте:
 - а) Скільки порівнянь виконується при першому лінійному перегляді;
 - б) Скільки порівнянь виконується при другому лінійному перегляді;
 - в) Скільки порівнянь виконується при i -тому лінійному перегляді;
 - г) Скільки порівнянь виконується при $(n-1)$ -ому лінійному перегляді;
 - д) Скільки всього виконується порівнянь;
 - е) Скільки всього виконується пересилань.
3. Внесіть корективи до алгоритму сортування вибором таким чином, щоб використати двонаправлений варіант сортування методом вибору, у якому на кожному кроці відшукуються та встановлюються на свої місця й мінімальне, й максимальне значення. Виконайте у середовищі демонстрації модифікований алгоритм сортування вибором для масивів з кількістю елементів від 10 до 100. Побудуйте графіки залежності кількості порівнянь та кількості пересилань від кількості елементів масиву. Порівняйте отримані графіки з графіками функцій $y=n$; $y=n^2$; $y=\log_2(n)$. Зробіть висновки щодо ефективності алгоритмів.
4. Порівняйте результати, отримані у завданнях 1,3.

ПРАКТИЧНА РОБОТА №6

Тема: Сортування вставками

Мета: Навчитися застосовувати метод сортування вставками при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Сортування вставками – алгоритм простого сортування. Незважаючи на те, що цей метод сортування менш ефективний за деякі більш складні алгоритми, такі як швидке сортування, він має ряд переваг:

- простий у реалізації;
- є ефективним на невеликих наборах даних (на наборах даних до десятків елементів може виявитися найкращим);
- не потребує тимчасової пам'яті, навіть під стек;
- може сортувати список по мірі його надходження;
- стійкий алгоритм сортування, тобто не змінює порядок вже відсортованих елементів.

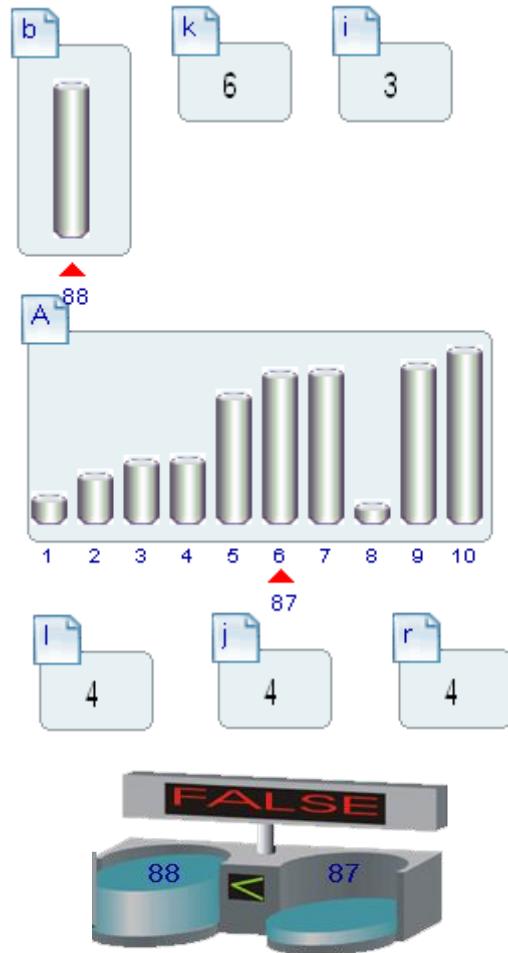
Алгоритм сортування вставками оснований на наступній ідеї: припустимо, що перші k елементів масива $A[1..n]$ вже упорядковані: $A[1] \leq A[2] \leq \dots \leq A[k]$, $A[k+1], \dots, A[n]$. Знайдемо місце елемента $A[k+1]$ у початковому відрізку $A[1], \dots, A[k]$ і вставимо цей елемент на своє місце, отримавши упорядковану послідовність довжини $k+1$. Оскільки початковий відрізок масива впорядкований, пошук треба реалізувати як бінарний. Вставці елемента на своє місце повинна передувати процедура зсуву «хвоста» початкового відрізка для звільнення місця.

```

Program InsSort;
Const
  n = 10;
Var
  A : array[1..n] of Data;
  k, l, r, i, j : Integer;
  b : Data;

Begin
  For k := 1 to n-1 do begin
    b := A[k+1];
    If b < A[k]
      then begin
        l := 1;
        r := k;
        Repeat
          j := (l + r) div 2;
          If b < A[j]
            then r := j
            else l := j + 1;
        Until (l = j);
        For i := k downto j do
          A[i+1] := A[i];
        A[j] := b ;
      end
    end;
End.

```



Алгоритм сортування вставками здійснює $n-1$ лінійний перегляд масива A . У кожному перегляді шукається місце елемента $A[k+1]$, наступного за вже упорядкованим діапазоном масива $1..k$. Потім елемент $A[k+1]$ вставляється на своє місце. Вставці елемента передує зсув діапазона $j..k$ праворуч на одну позицію. Таким чином, ліва, вже упорядкована частина масива робиться більшою на 1 елемент.

Література:

1. М.С. Львов, А.В. Спиваковский, С.В. Белоусова. Основы программирования на языке Паскаль. Херсон: МИБ, 1997.– 153 с.
2. С.М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.

Задачі для самостійного розв'язування

1. Виконайте у середовищі демонстрації алгоритм сортування вставками для масивів з кількістю елементів від 10 до 100. Побудуйте графіки залежності кількості порівнянь та кількості пересилань від кількості елементів масиву. Порівняйте отримані графіки з графіками функцій $y=n$; $y=n^2$; $y=\log_2(n)$, $y= n \log_2(n)$. Зробіть висновки щодо ефективності алгоритмів.
2. Виконайте алгоритм сортування вставками у середовищі демонстрації для масиву з 30 елементів з колекції даних системи. Визначте:
 - а) Скільки порівнянь та пересилань виконується, якщо масив вже відсортовано за зростанням;
 - б) Скільки порівнянь та пересилань виконується, якщо масив відсортовано за спаданням;
 - в) Скільки порівнянь та пересилань виконується, якщо масив містить елементи, розташовані хвилюю;
 - г) Скільки порівнянь та пересилань виконується, якщо масив заповнюється випадковим чином.Зробіть висновок щодо кращого, гіршого та середнього випадків.
3. Внесіть корективи до алгоритму сортування вставками таким чином, щоб сортування виконувалося за спаданням. Виконайте у середовищі демонстрації модифікований алгоритм сортування вставками для масивів з кількістю елементів від 10 до 100. Побудуйте графіки залежності кількості порівнянь та кількості пересилань від кількості елементів масиву. Порівняйте отримані графіки з графіками функцій $y=n$; $y=n^2$; $y=\log_2(n)$, $y= n \log_2(n)$. Зробіть висновки щодо ефективності алгоритмів.

ПРАКТИЧНА РОБОТА №7

Тема: Сортування Шелла

Мета: Навчитися застосовувати метод сортування Шелла при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Сортування Шелла — покращений алгоритм сортування вставками. Він за швидкістю суттєво переважає метод простими вставками, тому що в основі його полягає механізм переміщення записів не короткими кроками, а великими стрибками.

Перевагами даного алгоритму є:

- відсутність додаткової пам'яті;
- відсутність погіршення ефективності виконання на невдалих наборах даних.

Сортування виконується досить швидко завдяки тому, що на кожній з проміжних стадій у ньому приймають участь або досить короткі масиви, або вже досить добре впорядковані. Саме тому на кожному етапі можна користуватися методом простих вставок.

Основна ідея полягає у порівнянні на початкових стадіях пар значень, що відстають один від одного на досить великій відстані у впорядкованому наборі даних. Це дозволяє швидко переставляти далекі невпорядковані пари значень.

Алгоритм полягає у наступному: на першому кроці проходить впорядкування елементів $n \div 2$ пар при $i=1..n \div 2$; на другому кроці впорядковуються елементи у $n \div 4$ групах по 4 елементи при $i=1..n \div 4$ і т. д. На останньому кроці впорядковуються відразу всі елементи масиву. Таким чином видно, що загальна кількість ітерацій даного сортування дорівнює $\log_2 n$.

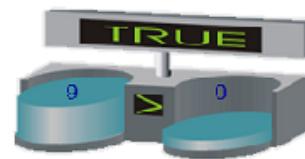
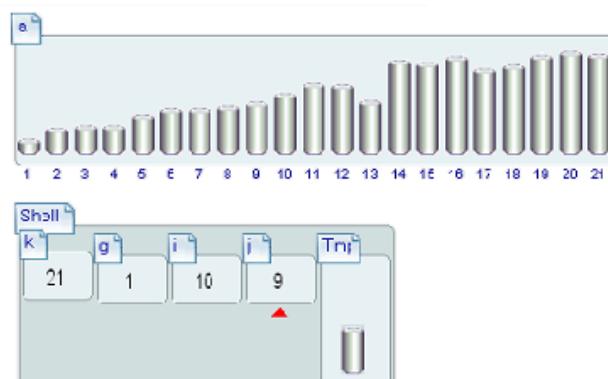
```

Program ShellSort;
Const n=21;
Var a : array[1..n] of Data;

Procedure Shell(k:Integer);
Var
  c : Boolean;
  g : Integer;
  i : Integer;
  j : Integer;
  Tmp : Data;
Begin
  g := n div 2;
  While (g>0) do
  begin
    For i:=g+1 to n do
    begin
      j:=i-g;
      While (j>0) do
      begin
        If (a[j]>a[j+g]) then
        begin
          Tmp:=a[j];
          a[j]:=a[j+g];
          a[j+g]:=Tmp;
          j:=j-1;
        end
        else
          j:=0;
      end;
    end;
    g:= g div 2;
  end;
End;

Begin
  Shell(n);
End.

```



Задачі для самостійного розв'язування

1. Використовуючи приклад алгоритму сортування, співставте кількість перестановок та порівнянь на масиві з 20, 50 та 100 елементів, розташованих за спаданням (задаються користувачем самостійно з клавіатури) за допомогою середовища демонстрації. Побудуйте графіки залежності кількості елементів від кількості порівнянь та перестановок відповідно.
2. Використовуючи приклад алгоритму сортування, співставте кількість перестановок та порівнянь на масиві з 50 елементів, у наступних випадках: 1). елементи відсортовані за зростанням; 2). елементи відсортовані за спаданням; 3). елементи розташовані

хвилеподібно. Побудуйте графіки залежності кількості порівнянь та перестановок від кількості елементів відповідно. Визначте самостійно, який з варіантів розташування елементів у масиві є «найкращим», «найгіршим».

3. Співставте кількість перестановок та порівнянь на масиві з 50 елементів, розташованих хвилеподібно, використовуючи алгоритми сортування простими, бінарними вставками та метод Шелла. Побудуйте 3 графіки залежності кількості порівнянь та перестановок від кількості елементів для наочного представлення та порівняння ефективності даних алгоритмів.

ПРАКТИЧНА РОБОТА №8

Тема: Аналіз ефективності простих алгоритмів сортування масивів

Для кількісного аналізу візьмемо наступні алгоритми колекції системи у середовищі демонстрації:

- Сортування масиву обмінами
- Покращене сортування обмінами (1 варіант)
- Сортування масиву методом вставки

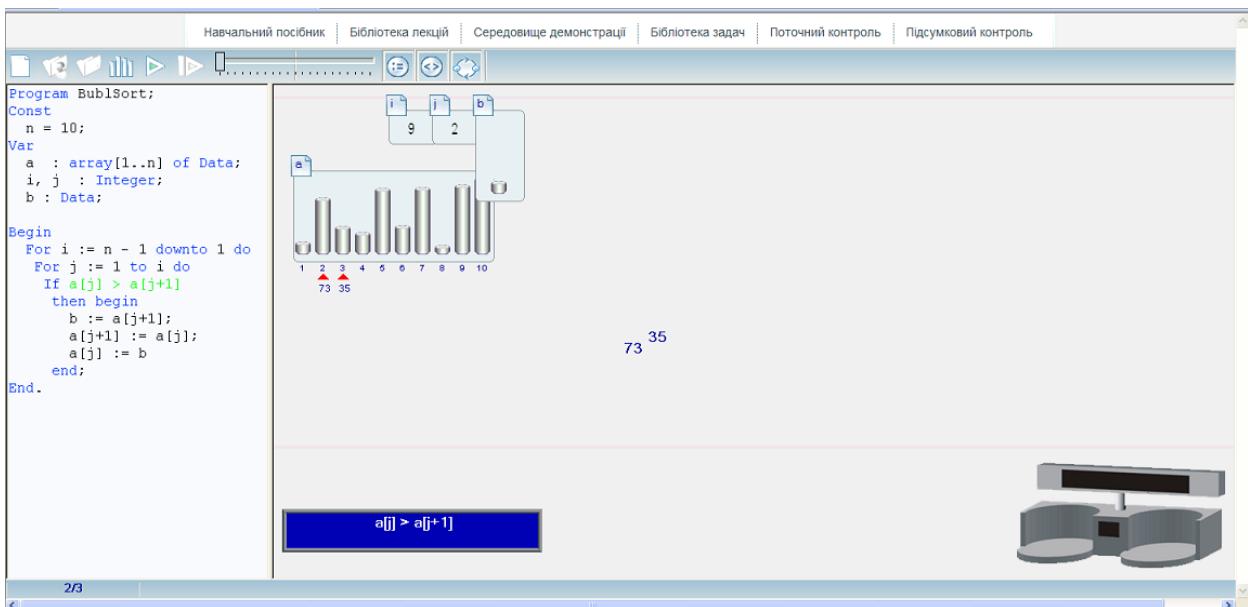
Проведемо моделювання роботи кожного з них на масивах різної довжини. При цьому проаналізуємо ефективність та швидкодію кожного з алгоритмів.

Для порівняння алгоритмів використаємо масиви довжиною 10, 100 та 200 елементів.

Отже, перейдемо до покрокового виконання сортувань масивів. Перш за все, необхідно увійти до середовища демонстрації системи дистанційного навчання WebOAP. Потрібно натиснути на кнопку «Відкрити колекцію». У діалоговому вікні, що з'явилося, обрати перший вид сортування з наведеного вище переліку (у колекції – 03.Сортування масиву обмінами), натиснути на «Колекція даних» і обрати тут 3 пункти: 01.Масив на 10 елементів, 11.Масив на 100 елементів, розташованих хвилею та 12.Масив на 200 елементів, розташованих хвилею. Після цього натиснути кнопку «OK» і, не виходячи з колекції системи, повторити ці дії для кожного з видів сортувань.

Після натиснення «OK» у вікні колекції системи перший з обраних алгоритмів відобразиться у лівій панелі екрану.

Після натиснення на кнопку «Заповнити даними» () необхідно обрати «Застосувати». Коли дані буде сформовано, стане активною кнопка «Почати» (). Після її натиснення на екрані буде відображатися наочне виконання алгоритму.



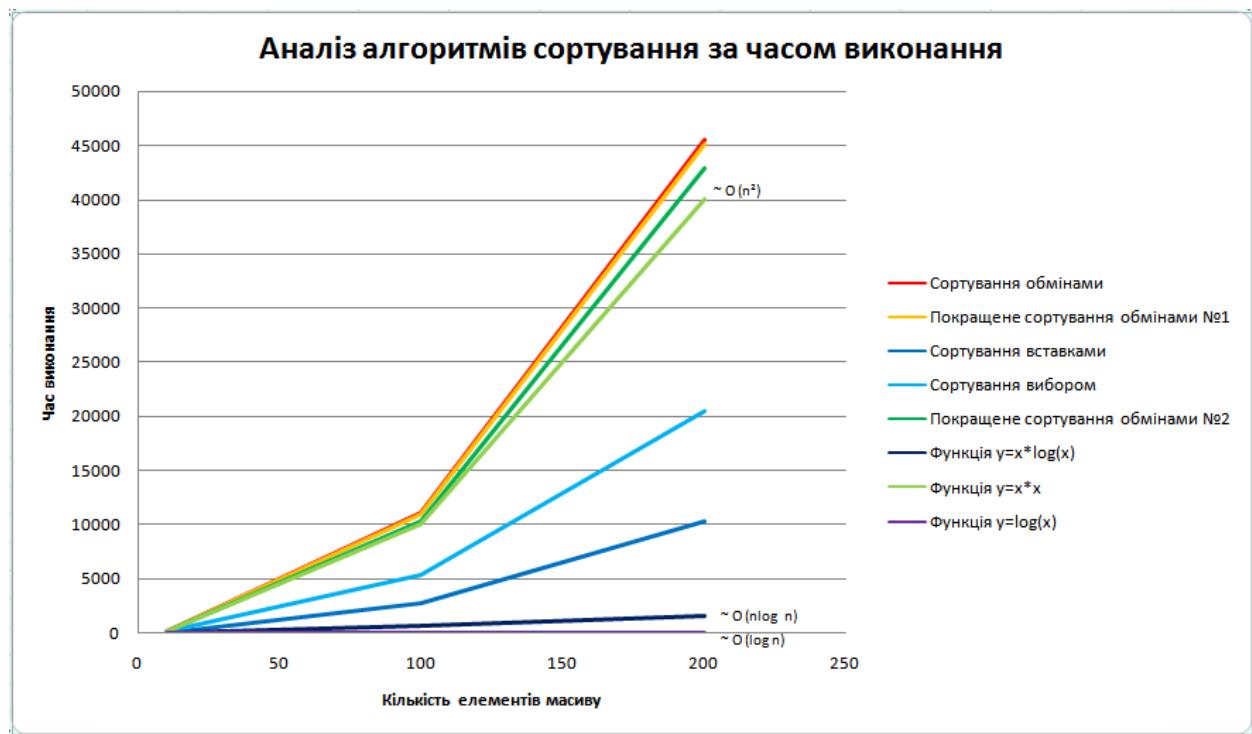
Отже, після виконання алгоритму ми бачимо кількість порівнянь та пересилань у лівому нижньому куті екрану ([2/3](#)). Таким чином ми отримали наступну відповідь: в результаті виконання алгоритму було виконано 2 порівняння та 3 пересилання.

Для того, щоб продовжити виконання всіх алгоритмів сортувань, необхідно знову натиснути на кнопку «Почати» (завантажиться текст алгоритму), потім на кнопку «Заповнити даними» (обрати «Застосувати») і натиснути на кнопку «Почати». Таким чином потрібно виконати всі заплановані алгоритми, після чого натиснути на кнопку «Почати». На екрані з'явиться вікно з результатами, які потрібно експортувати до Excel (кнопка «Експорт у Excel»).

Отже, тепер можна зробити аналіз алгоритмів за тими показниками, які були перераховані у вступі, а саме: часом сортування, пам'яттю, стійкістю, природністю поведінки та мажорованістю.

Для первого параметру – часу сортування – представимо діаграму з графіками залежності часу виконання від кількості елементів масиву, якщо їх розмірність не перевищує 200.

З аналізу графіків стає зрозумілим, яку складність має кожний алгоритм сортування за кількістю порівнянь.



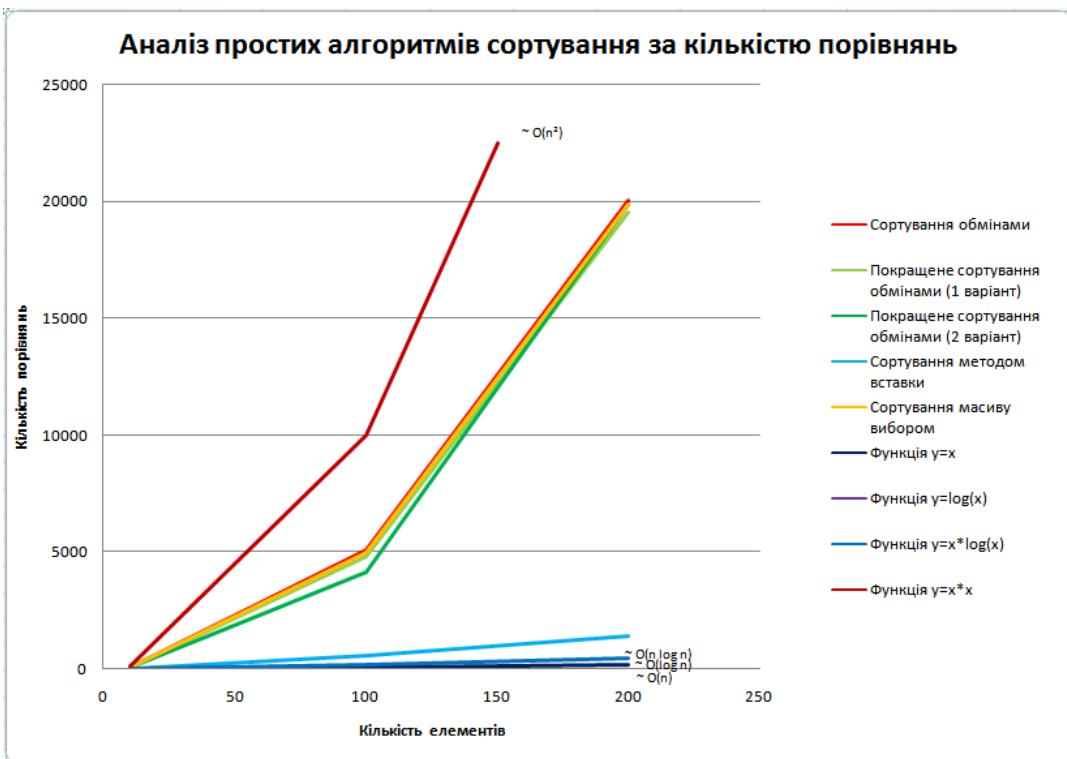
З діаграми видно, що складність майже всіх алгоритмів сортування пропорційна функції $y=n^2$, тобто вони мають складність $O(n^2)$.

Другий критерій – *пам'ять*. Додатковий масив для роботи даних методів не використовується. Тому додаткової пам'яті вони не потребують.

Стійкість. Не змінили взаємного розташування елементів у вже відсортованому масиві даних сортування лише сортування обмінами, перший та другий його покращені варіанти. Тож саме вони є стійкими і у разі використання масивів з елементами складної структури дозволяють значно зекономити час сортування.

Природністю поведінки відзначилися алгоритми сортування обмінами, перший та другий його покращені варіації. Вони є ефективними при обробці вже відсортованих даних.

Мажорованість. Використовуючи дані, представлені на рисунку, можна знайти функцію мажоранту $f(n)$.



Наприклад, для сортування обмінами константа c (відношення кількості порівнянь до функції $y=n^2$ для відповідної кількості елементів) наблизено дорівнює 0,5, як і для сортування масиву вибором. Тепер для знаходження мажоранти необхідно помножити цю константу c на значення функції $y=n^2$ для відповідних кількостей елементів та побудувати графіки. Наприклад, для сортування обмінами вони будуть мати наступний вигляд:



Отже, мажорантою для сортування обмінами за кількістю порівнянь буде функція $y=0,5 \cdot n^2$.

Задачі для самостійного розв'язування

1. Виконайте у середовищі демонстрації покращені алгоритми сортування обмінами (1 та 2 варіант) для масивів з кількістю елементів від 10 до 100 з кроком 10. Побудуйте графіки залежності кількості порівнянь та пересилань від кількості елементів масиву. Побудуйте функцію мажоранту для даного алгоритму сортування за кількістю порівнянь та пересилань. Зробіть висновки щодо ефективності алгоритмів.
2. Виконайте у середовищі демонстрації алгоритм сортування вставками для масивів з кількістю елементів від 10 до 100 та кроком 20. Побудуйте функцію мажоранту для даного алгоритму за часом виконання. Зробіть висновки щодо ефективності алгоритму.
3. Виконайте у середовищі демонстрації алгоритм сортування вибором для масивів з кількістю елементів від 10 до 200 з кроком 15. Побудуйте графіки залежності кількості порівнянь та часу виконання від кількості елементів масиву. Побудуйте функцію мажоранту для даного алгоритму сортування за цими показниками. Зробіть висновки щодо ефективності алгоритму.

ПРАКТИЧНА РОБОТА №9

Тема: Сортування злиттям

Мета: Навчитися застосовувати метод сортування злиттям при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Сортування злиттям (англ. merge sort) – алгоритм сортування, який впорядковує списки (або інші структури даних, доступ до яких можна отримувати лише послідовно) у визначеному порядку.

Це сортування використовує метод «розділяй та володій». Спочатку задача розбивається на кілька невеликих підзадач, які розв'язуються за допомогою рекурсивного виклику або безпосередньо (якщо їх розмір достатньо малий). Наприкінці їх розв'язання комбінуються.

Для розв'язання задачі сортування ці три етапи виглядають наступним чином:

1. масив розбивається навпіл (частини повинні бути приблизно однакові);
2. кожна з частин сортується окремо (наприклад, тим самим алгоритмом);
3. два впорядкованих масиви з'єднуються в один.

Рекурсивне розбиття задачі виконується до тих пір, поки розмір масиву не досягне одиниці, так як будь-який масив довжини 1 можна вважати впорядкованим.

Даний алгоритм був винайдений Джоном фон Нейманом (1945 р.)

Злиття двох відсортованих масивів можна пояснити наступним чином: у будь-який момент часу ми бачимо початковий елемент у кожному з масивів. Нехай необхідно отримати масив, що містить всі елементи двох відсортованих масивів, та також відсортований за зростанням. Таким чином, на кожному кроці ми беремо менший з

двох елементів і переносимо його до результиручого масиву. Коли один з вхідних масивів вичерпується – додаємо всі елементи масиву, що залишився, до результиручого.

Час роботи описаного вище алгоритму $O(n \log n)$, якщо відсутнє виродження на невдалих випадках (у випадку з швидким сортуванням це значно збільшує час його виконання).

```
Program MergingSort;
Const
  n = 10;
Var
  A : array[1..n] of Data;
  d : array[1..n] of Data;

Procedure Slit(k, q: Integer );
Var
  m: Integer;
  i, j, T: Integer;
Begin
  m := k + (q-k) div 2;
  i := k;
  j := m + 1;
  t := 1;
  While (i <= m) and (j <= q) Do
    Begin
      If A[i] <= A[j] Then
        Begin
          d[T] := A[i];
          i := i + 1;
        End
      Else Begin
        d[T] := A[j];
        j := j + 1;
      End;
      T := T + 1;
    End;

  While i <= m Do
    Begin
      d[T] := A[i];
      i := i + 1;
      T := T + 1;
    End;
  While j <= q Do Begin
    d[T] := A[j];
    j := j + 1;
    T := T + 1;
  End;
End;
```

```

For i := 1 to T - 1 Do
    A[k+i-1] := d[i];
End;

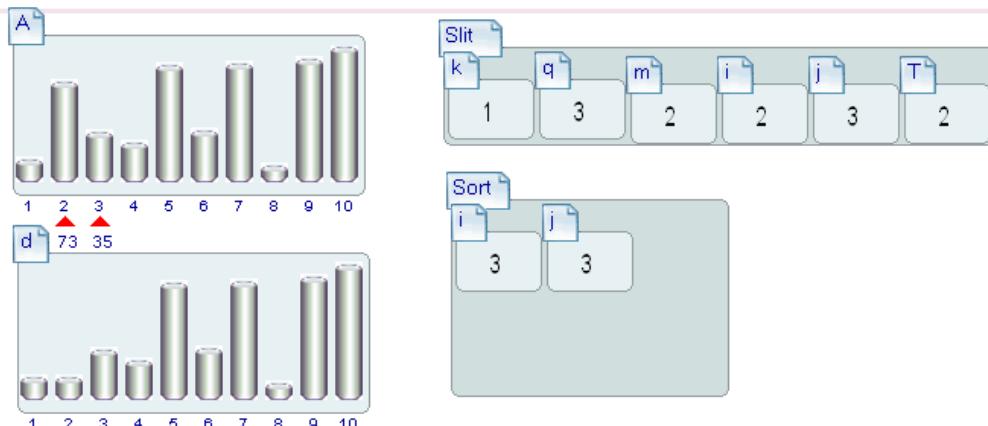
Procedure Sort(i, j: Integer);
Var T: Data;
Begin
    If i < j Then
    Begin
        If j-i = 1 Then Begin
            If A[j] < A[i] Then Begin
                T := A[i];
                A[i] := A[j];
                A[j] := T;
            End
        End
        Else Begin
            Sort(i, i + (j-i) div 2);
            Sort(i + (j-i) div 2 + 1, j);
            Slit(i, j)
        End;
    End;
End;

```

```

Begin
    Sort(1, n);
End.

```



A[i] <= A[j]



Література:

1. М.С. Львов, А. В. Спиваковский, С.В. Белоусова. Основы программирования на языке Паскаль. Херсон: МИБ, 1997. – 153 с.
2. С.М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.
3. Д.Кнут. Искусство программирования, том 3. Сортировка и поиск. – М.: Вильямс, 2007. – 800 с.

Задачі для самостійного розв'язування

1. Використовуючи середовище демонстрації, виконайте алгоритм сортування злиттям, співставте кількість перестановок та порівнянь на масивах з кількістю елементів від 10 до 100, розташованих за спаданням (здаються автоматично) за допомогою середовища демонстрації. Виконайте експорт даних в Excel. Побудуйте графіки залежності кількості порівнянь та кількості пересилань від кількості елементів масиву. Порівняйте отримані графіки з графіками функцій $y=n$; $y=n^2$; $y=\log_2(n)$, $y= n \log_2(n)$. Зробіть висновки щодо ефективності алгоритму.
2. Виконайте алгоритм сортування злиттям на масиві з максимальною кількістю елементів, передбаченою у середовищі демонстрації. Визначте кількість порівнянь і перестановок. Співставте кількість перестановок та порівнянь у «найкращому», «найгіршому» та «середньому» випадках.
3. Розробіть алгоритм сортування масиву за спаданням методом злиття. Перевірте працездатність алгоритму для масивів зі 100 та 200 елементами, завантаживши їх з колекції даних системи.
4. Використовуючи середовище демонстрації, виконайте алгоритми сортування злиттям, сортування обмінами та сортування вибором, співставте кількість перестановок та порівнянь на масивах з кількістю елементів від 10 до 100, розташованих за спаданням (здаються автоматично) за допомогою середовища демонстрації. Виконайте експорт даних в Excel. Побудуйте графіки залежності кількості порівнянь та кількості пересилань від кількості елементів масиву. Зробіть висновки щодо ефективності алгоритмів. Зробіть рекомендації щодо використання певних алгоритмів у залежності від кількості елементів.

ПРАКТИЧНА РОБОТА №10

Тема: Швидке сортування Хоара

Мета: Навчитися застосовувати метод швидкого сортування Хоара при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Швидке сортування, або сортування Хоара (англ. quicksort) – поширений алгоритм сортування, один із швидких універсальних алгоритмів, винайдений англійцем Ч. Хоаром та названий на його честь. Має складність виконання в середньому $O(n \log n)$ (кількість обмінів при впорядкуванні n елементів).

Удосконаливши метод сортування, заснований на обмінах, Ч. Хоар запропонував алгоритм QuickSort сортування масивів, який дає на практиці відмінні результати і дуже просто програмується. Автор назвав свій алгоритм швидким сортуванням.

Ідея Ч. Хоара полягає в наступному:

1. Оберемо деякий елемент x масива A випадковим чином.
2. Переглянемо масив у прямому напрямку ($i = 1, 2, \dots$), шукаючи в ньому елемент $A[i]$ не менший, ніж x .
3. Переглянемо масив у зворотному напрямку ($j = n, n-1, \dots$), шукаючи в ньому елемент $A[j]$ не більший, ніж x .
4. Міняємо місцями $A[i]$ і $A[j]$.

Пункти 2-4 повторюємо до тих пір, поки $i < j$.

В результаті такого зустрічного проходу початок масива $A[1..i]$ і кінець масива $A[j..n]$ виявляються розділеними «бар’єром» x :

$$A[k] \leq x \text{ при } k < i, A[k] \geq x \text{ при } k > j,$$

причому на розмежування ми витратимо не більше $n/2$ перестановок. Тепер залишилось зробити ті ж дії з початком і кінцем масива, тобто застосувати їх рекурсивно! Таким чином, описана нами процедура

Hoare залежить від параметрів k і m – початкового і кінцевого індексів відрізка масива, що обробляється.

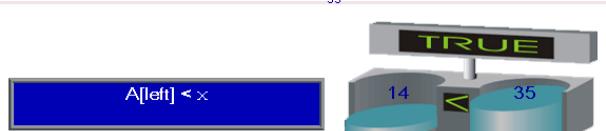
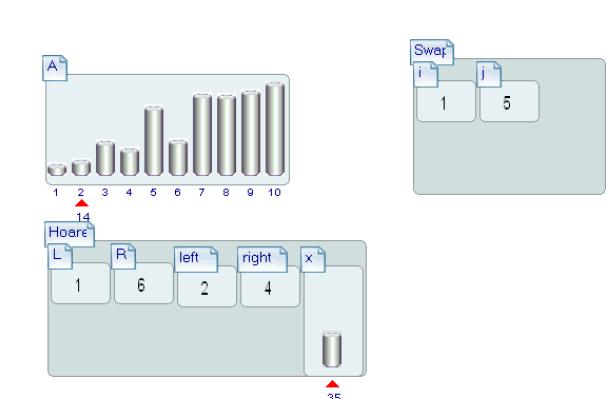
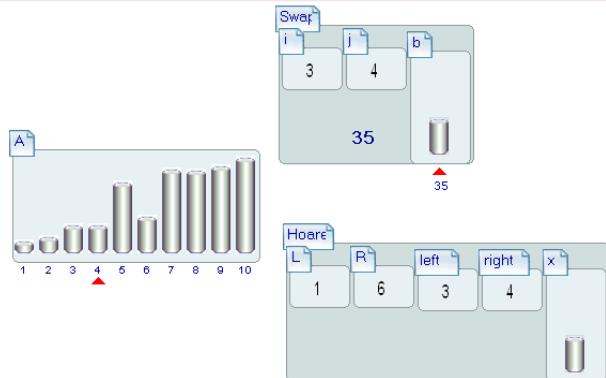
```

Program QuickSort;
Const
  n = 10;
Var
  A : array[1..n] of Data;

Procedure Swap(i, j : Integer);
Var
  b : Data;
Begin
  b := a[i];
  a[i] := a[j];
  a[j] := b
End;

Procedure Hoare(L, R : Integer);
Var
  left, right : Integer;
  x : Data;
Begin
  If L < R
    then begin
      x := A[(L + R) div 2];
      left := L;
      right := R ;
      Repeat
        While A[left] < x do
          left := left + 1;
        While A[right] > x do
          right:=right - 1;
        If left <= right
          then begin
            Swap(left, right);
            left := left + 1;
            right := right - 1;
          end
        until left > right;
      Hoare(L, right);
      Hoare(left, R)
    end
  End;
Begin
  Hoare(1, n);
End.

```



The screenshot shows a software interface with a code editor and a visualization window.

Code Editor (Pascal):

```

left, right : Integer;
x : Data;
Begin
  If L < R
    then begin
      x := A[(L + R) div 2];
      left := L;
      right := R ;
      Repeat
        While A[left] < x do
          left := left + 1;
        While A[right] > x do
          right:=right - 1;
        If left <= right
          then begin
            Swap(left, right);
            left := left + 1;
            right := right - 1;
          end
        until left > right;
      Hoare(L, right);
      Hoare(left, R)
    end
End;
Begin
  Hoare(1, n);
End.
  
```

Execution Trace:

- Initial State:** An array A with elements [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. A pointer A points to index 1.
- Swap Step:** Two boxes labeled 'Swap' show indices i and j. The value at index 9 is moved to index 10, and vice versa. The array state is [1, 2, 3, 4, 5, 6, 7, 8, 10, 9].
- Hoare Step:** Two boxes labeled 'Hoare' show pointers L and R. The value at index 10 is swapped with the value at index 10. The array state is [1, 2, 3, 4, 5, 6, 7, 8, 10, 10].
- Final State:** A blue button labeled "End." is shown, indicating the completion of the sorting process.

Переваги алгоритму швидкого сортування Хоара:

- простий у реалізації;
- один з найшвидших на практиці алгоритм сортування загального призначення;
- мінімальні потреби обсягу пам'яті для роботи (всього $O(\log n)$);
- має ефективну модифікацію для сортування рядків (алгоритм Седжвіка);
- поєднується з механізмами кешування та віртуальної пам'яті.

Недоліки алгоритму швидкого сортування Хоара:

- нестійкий;
- сильно деградує за швидкістю при невдалому виборі опорного елементу;
- може привести до вичерпання пам'яті при потребі виклику $O(n)$ вкладених процедур.

Література:

1. М.С. Львов, А.В. Спиваковский, С.В. Белоусова. Основы программирования на языке Паскаль. Херсон: МИБ, 1997. – 153 с.
2. С.М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.
3. Д. Кнут. Искусство программирования, том 3. Сортировка и поиск – М.: Вильямс, 2007. – 800 с.

Задачі для самостійного розв'язування

1. Використовуючи середовище демонстрації виконайте алгоритм швидкого сортування Хоара, співставте кількість перестановок та порівнянь на масивах з кількістю елементів від 10 до 100, розташованих за спаданням (здаються автоматично) за допомогою середовища демонстрації. Виконайте експорт даних в Excel. Побудуйте графіки залежності кількості порівнянь та кількості пересилань від кількості елементів масиву. Порівняйте отримані графіки з графіками функцій $y=x$; $y=x^2$; $y=\log_2(x)$, $y= x \log_2(x)$. Зробіть висновки щодо ефективності алгоритму.
2. Використовуючи визначення різних випадків (кращого, гіршого та середнього) для методу швидкого сортування, співставте кількість порівнянь та перестановок для масиву довжиною 50 елементів на кожному з них. Результати виконання алгоритму на різних масивах експортуйте з середовища демонстрації у редактор Excel.
3. Внесіть корективи до алгоритму Хоара так, щоб опорний елемент знаходився випадковим чином. Засобами середовища демонстрації визначте кількість порівнянь та перестановок та порівняйте їх з результатами, отриманими у завданні 1. Яку перевагу надає пошук опорного елементу таким способом?
4. Виконайте модифікацію алгоритму швидкого сортування, таким чином, щоб усунути одну гілку рекурсії. Після розподілу масиву викличте рекурсивну процедуру тільки для меншого підмасиву, а більший опрацюйте у циклі в межах того самого виклику. За допомогою середовища демонстрації визначте кількість порівнянь та перестановок у модифікованому варіанті алгоритму. Порівняйте ефективність алгоритму Хоара та модифікованого алгоритму для середнього випадку. Визначіть глибину рекурсії в середньому та

найгіршому випадках. Яка особливість стає наочною при покращеному розв'язанні?

5. Використовуючи середовище демонстрації виконайте алгоритми швидкого сортування Хоара, сортування обмінами, сортування вибором та сортування вставками. Співставте кількість перестановок та порівнянь на масивах з кількістю елементів від 10 до 100, розташованих за спаданням (задаються автоматично) за допомогою середовища демонстрації. Виконайте експорт даних в Excel. Побудуйте графіки залежності кількості порівнянь та кількості пересилань від кількості елементів масиву. Зробіть висновки щодо ефективності алгоритмів. Зробіть рекомендації щодо використання певних алгоритмів у залежності від кількості елементів.

ПРАКТИЧНА РОБОТА №11

Тема: Піраміdalne сортування

Мета: Навчитися застосовувати метод піраміdalного сортування при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Алгоритм піраміdalного сортування HeapSort використовує представлення масива у вигляді дерева. Цей алгоритм не потребує допоміжних масивів, сортуючи «на місці». Розглянемо спочатку метод представлення масива у вигляді дерева:

Нехай $A[1 .. n]$ – деякий масив. Співставимо йому дерево, використовуючи наступні правила:

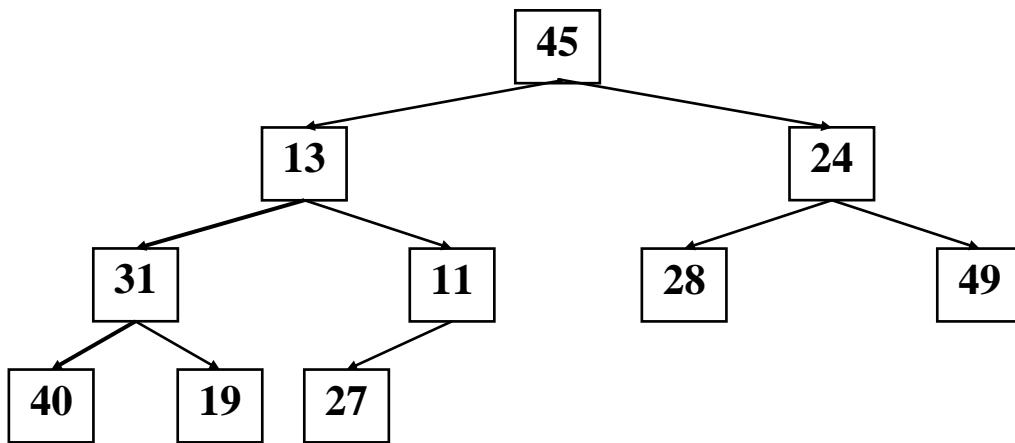
1. $A[1]$ – корінь дерева;
2. Якщо $A[i]$ – вузол дерева і $2i \leq n$, то $A[2*i]$ – вузол-«лівий син» вузла $A[i]$;
3. Якщо $A[i]$ – вузол дерева і $2i + 1 \leq n$, то $A[2*i+1]$ – вузол – «правий син» вузла $A[i]$.

Правила 1-3 визначають у масиві структуру дерева, причому глибина дерева не перевищує $\lceil \log_2 n \rceil + 1$. Вони ж задають і спосіб руху по дереву від кореня до листів. Рух вверх задається правилом 4:

4. Якщо $A[i]$ – вузол дерева і $i > 1$, то $A[i \bmod 2]$ – вузол – «батько» вузла $A[i]$.

Приклад: Нехай $A = [45 13 24 31 11 28 49 40 19 27]$ – масив. Відповідне йому дерево має вигляд:

Зверніть увагу на те, що всі рівні дерева, за виключенням останнього, повністю заповнені, останній рівень заповнений зліва і індексація елементів масива здійснюється зверху-вниз і зліва-направо. Тому дерево упорядкованого масива задовольняє наступним властивостям: $A[i] \leq A[2*i]$, $A[i] \leq A[2*i+1]$, $A[2*i] \leq A[2*i+1]$.



Як це не дивно, алгоритм *HeapSort* спочатку буде дерево, яке задовольняє прямо протилежним спiввiдношенням

$$A[i] \geq A[2*i], A[i] \geq A[2*i+1],$$

а потім мiняє мiсцями $A[1]$ (найбiльший елемент) i $A[n]$.

Алгоритм *HeapSort* працює в два етапа:

- I. Побудова сортуючого дерева;
- II. Просiвання елементiв по сортуючому дереву.

Дерево, що представляє масив, називається сортующим, якщо виконуються умови. Якщо для деякого i ця умова не виконується, будемо говорити, що має мiсце (сiмейний) конфлiкт у трикутнику i .

I на I-ому, i на II-ому етапах елементарна дiя алгоритма полягає в розв'язаннi сiмейного конфлiкта: якщо найбiльший з синiв бiльше, нiж батько, то переставляються батько i цей син (процедура *ConSwap*).

В результатi перестановки може виникнути новий конфлiкт у тому трикутнику, куди переставлений батько. Таким чином можна говорити про конфлiкт (рода) у пiдеревi з коренем у вершинi i . Конфлiкт рода розв'язується послiдовним розв'язуванням сiмейних конфлiктiв проходом по дереву зверху-вниз. Конфлiкт рода розв'язаний, якщо прохiд закiнчився ($i > n \div 2$) або в результатi перестановки не виник новий сiмейний конфлiкт. (процедура *Conflict*).

```

Procedure ConSwap(i, j : Integer);
Var
  b : Data;
Begin
  If a[i] < a[j]
    then begin
      b := a[i];
      a[i] := a[j];
      a[j] := b
    end
End;

Procedure Conflict(i, k : Integer);
Var
  j : Integer;
Begin
  j := 2*i;
  If j = k
    then ConSwap(i, j)
  else if j < k
    then begin
      if a[j+1] > a[j]
        then j := j + 1;
      ConSwap(i, j);
      Conflict(j, k)
    end
End;

```

I етап – побудова сортуючого дерева – оформимо у вигляді рекурсивної процедури, використовуючи визначення:

Якщо ліве і праве піддерева $T(2i)$ і $T(2i+1)$ сортуючого дерева $T(i)$, то для перебудови $T(i)$ необхідно розв'язати конфлікт роди в цьому дереві.

```

Procedure SortTree(i : Integer);
Begin
  If i <= n div 2
    then begin
      SortTree(2*i);
      SortTree(2*i+1);
      Conflict(i, n)
    end
End;

```

На II-ому етапі – етапі просіювання – для k від n до 2 повторюються наступні дії:

1. Переставити $A[1]$ і $A[k]$;
2. Побудувати сортуюче дерево початкового відрізка масива $A[1..k-1]$, усунувши конфлікт рода в корені $A[1]$. Зауважимо, що друга дія вимагає введення в процедуру *Conflict* параметра k .

```

Program HeapSort;
Const
  n = 100;
Var
  A : Array[1..n] of Data;
  k : Integer;

Procedure ConSwap(i, j : Integer);
Var
  b : Data;
Begin
  If a[i] < a[j]
    then begin
      b := a[i];
      a[i] := a[j];
      a[j] := b
    end
End;

Procedure Conflict(i, k : Integer);
Var
  j : Integer;
Begin
  j := 2*i;
  If j = k
    then ConSwap(i, j)
  else if j < k
    then begin
      if a[j+1] > a[j]
        then j := j + 1;
      ConSwap(i, j);
      Conflict(j, k)
    end
End;

```

```

Procedure SortTree(i : Integer);
Begin
  If i <= n div 2
    then begin
      SortTree(2*i);
      SortTree(2*i+1);
      Conflict(i, n)
    end
  End;

Begin
  SortTree(1);
  For k := n downto 2 do begin
    ConSwap(k, 1);
    Conflict(1, k - 1)
  end;
End.

```

Переваги методу:

- має доведену оцінку найгіршого випадку $O(n \log n)$;
- потребує всього $O(1)$ додаткової пам'яті (якщо дерево організовувати таким чином, як вказано вище).

Недоліки методу:

- складний у реалізації;
- нестійкий – для забезпечення стійкості потрібно розширювати ключ;
- на майже відсортованих масивах працює так само довго, як і на хаотичних даних;

Література:

1. М.С. Львов, А.В. Спиваковский, С.В. Белоусова. Основы программирования на языке Паскаль. Херсон: МИБ, 1997. – 153 с.
2. С.М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.

Задачі для самостійного розв'язування

1. Виконайте алгоритми піраміdalного сортування, сортування обмінами, вибором і вставками у середовищі демонстрації на масивах зі 100, 200 та 500 елементами (масиви задаються автоматично). Співставте кількість перестановок та порівнянь.

2. Виконайте алгоритм піраміdalного сортування у середовищі демонстрації при сортуванні масивів з 200 елементами:
- а) Для масиву, відсортованого за зростанням;
 - б) Для масиву, відсортованого за спаданням;
 - в) Для масиву, елементи якого розташовані хвилюю;
 - г) Для масиву, елементи якого задано випадковим чином.
- Проаналізуйте результати виконання алгоритму на різних масивах (кількість порівнянь, кількість пересилань). Зробіть висновки щодо найкращого та найгіршого випадків.
3. Виконайте алгоритм піраміdalного сортування у середовищі демонстрації при «найгіршому» та «найкращому» випадках при сортуванні 40 елементів (в діалоговому вікні заповнення даними оберіть «Клавіатура» та «Показати» і заповніть масив власними даними). Порівняйте кількість перестановок та порівнянь, отриманих при виконанні алгоритму.
4. Виконайте у середовищі демонстрації алгоритм піраміdalного сортування для масивів з кількістю елементів від 10 до 100. Побудуйте графіки залежності кількості порівнянь та кількості пересилань від кількості елементів масиву. Порівняйте отримані графіки з графіками функцій $y=x$; $y=x^2$; $y=\log_2(x)$, $y= x \log_2(x)$. Зробіть висновки щодо ефективності алгоритмів.
5. Розробіть програму, яка здійснює піраміdalне сортування масиву з 50 елементів без використання рекурсії. Порівняйте результати роботи модифікованого алгоритму з алгоритмом піраміdalного сортування з колекції алгоритмів середовища демонстрації.

ПРАКТИЧНА РОБОТА №12

Тема: Пошук k -го елемента в масиві

Мета: Навчитися застосовувати алгоритм пошуку k -го елемента в масиві при розв'язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально перевірити складність алгоритму. Закріпити навички роботи у середовищі демонстрації інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування».

Короткі теоретичні відомості

Із задачею сортування тісно пов'язана задача знаходження k -того за величиною елемента у масиві. З її окремим випадком – пошуком першого (мінімального) і n -ного елементів (максимального) – ми вже знайомі. Аналогічно можна розв'язати задачу пошуку 2-го і $(n - 1)$ -го за величиною елементів, однак кількість порівнянь зростає вдвічі. Перш, ніж розглянути задачу у загальному вигляді, приведемо необхідне означення.

K -тим за величиною елементом послідовності називається такий елемент b цієї послідовності, для якого в послідовності існує не більш, ніж $k-1$ елемент, менший, ніж b , і не менше k елементів, менших або рівних b .

Наприклад, у послідовності {2, 1, 3, 5, 4, 2, 2, 3} четвертим за величиною буде 2.

Іншакше, якщо відсортувати масив, то k -тий елемент опиниться на k -тому місці ($b = A[k]$). Можна очкувати, що найбільш складним буде алгоритм пошуку елемента при $k = n \text{ div } 2$ – тобто елемента, рівновіддаленого від кінців масиву. Цей елемент назовемо медіаною масиву.

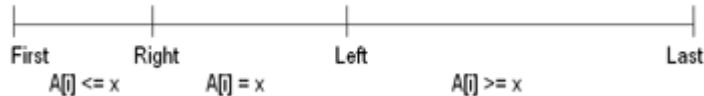
Очевидний розв'язок задачі пошуку k -того елемента – повне або часткове (до k) сортування масиву А. Наприклад, алгоритм TreeSort можна перервати після просівання k елементів. Тоді $C(n) = O(n + k \log_2 n)$

Однак ефективний на практиці розв'язок можна отримати, якщо застосувати ідею К.Хоара розподілу масиву бар'єром. Насправді,

аналіз методу розподілу елементів по бар'єру в процедурі Hoare показує, що після її закінчення мають місце співвідношення

$$Right < Left, \text{ при } i < Left A[i] \leq x, \text{ при } i > Right A[i] \geq x$$

Таким чином, відрізок $First .. Last$ розбиває на 3 частини:



Подальший пошук, отже, можна організувати так:

Якщо $k \leq Right$

то шукати k -тий елемент в $A[First .. Right]$

інакше якщо $k \leq Left$

то k -тий елемент дорівнює x

інакше шукати k -тий елемент в $A[Left + 1 .. Last]$

```
Program Hoare_Search;
Const
n = 20;
Var
A : Array[1..n] of Data;
k : Integer;
x : Data;
Procedure Swap(var i, j : Integer);
Var
temp : Data;
Begin
temp := a[i];
a[i] := a[j];
a[j] := temp
End;
```

```
Procedure HoareSearch(First, Last : Integer);
Var
l, r : Integer;
Found:Boolean;
Begin
If First = Last
then Found:=false
else If First < Last
then begin
x := A[k];
l := First;
r := Last;
```

```

Repeat
  While A[l] < x do
    l := Succ(l);
  While A[r] > x do
    r := Pred(r);
  If l <= r
    then begin
      Swap(l, r);
      l := Succ(l);
      r := Pred(r);
    end
  until l > r;
  If k <= r
    then HoareSearch(First, r)
  else If k < l
    then Found:=true
  else HoareSearch(l, Last)
end
End;

Begin
HoareSearch(1, n);
x:=A[k];
end.

```

Аналіз алгоритма.

Як і алгоритм швидкого сортування, цей алгоритм у гіршому випадку неефективний. Якщо при кожному розподілу 1-а частина масиву буде містити 1 елемент, алгоритм витратить $O(k*n)$ кроків. Однак в середньому його складність лінійна (незалежно від k). Зокрема, навіть при пошуку медіани $A[n \text{ div } 2]$

$$C_{cp}(n) = O(n), M_{cp}(n) = O(n)$$

Література:

3. М.С. Львов, А.В. Спиваковский, С.В. Белоусова. Основы программирования на языке Паскаль. Херсон: МИБ, 1997. – 153 с.
4. С.М. Окулов. Основы программирования. – Москва: ЮНИМЕДИА-СТАЙЛ, 2002. – 424 с.

Задачі для самостійного розв'язування

1. Використовуючи алгоритм знаходження k -го елемента в масиві, співставте кількість перестановок та порівнянь на масивах від 10

до 100 елементів за допомогою колекції даних середовища демонстрації. Експортуйте дані в Excel та побудуйте графіки:

- а) залежності кількості порівнянь від кількості елементів масиву;
- б) залежності кількості пересилань від кількості елементів масиву.

Порівняйте отримані графіки з графіками функцій $y=n$; $y=n^2$; $y=\log_2(n)$. Зробіть висновки щодо ефективності алгоритмів.

2. Виконайте пошук k -го елементу засобами середовища демонстрації для наступних масивів: відсортованого за спаданням з 10 елементів, відсортованого за зростанням з 10 елементів. Знайдіть кількість порівнянь та пересилань у кожному випадку.
3. Яку мінімальну кількість порівнянь необхідно виконати, щоб упевнитись, що k -ий елемент знайдено?

ПРАКТИЧНА РОБОТА №13

Тема: Дослідження роботи алгоритмів сортування на масивах різної довжини.

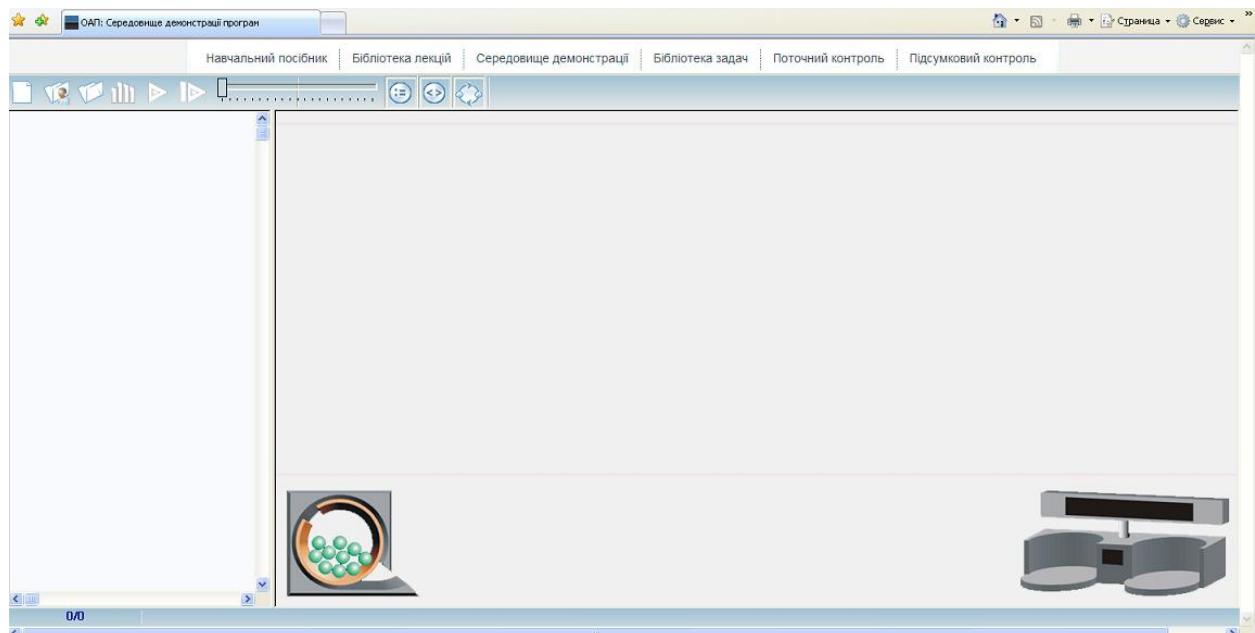
Для кількісного аналізу візьмемо наступні алгоритми колекції системи у середовищі демонстрацій:

- Сортування масиву обмінами.
- Покращене сортування обмінами (1 варіант).
- Сортування масиву методом вставки.
- Швидке сортування масиву методом Хоара.
- Сортування масиву вибором.
- Покращене сортування обмінами (2 варіант).
- Сортування злиттям.

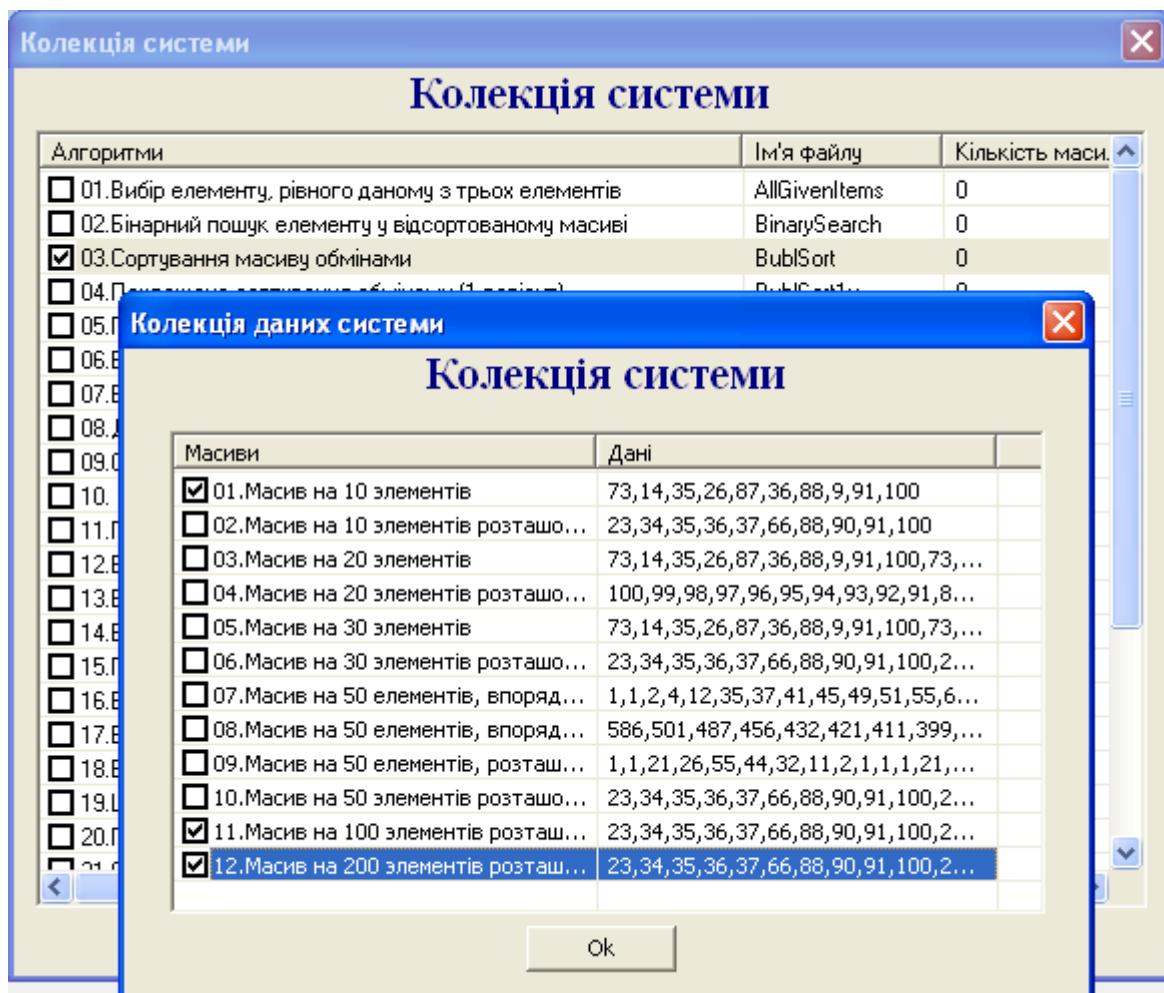
Проведемо моделювання роботи кожного з них на масивах різної довжини. При цьому проаналізуємо ефективність та швидкодію кожного з алгоритмів.

Для порівняння алгоритмів пропонується використати масиви довжиною в 10, 100 та 200 елементів.

Отже, перейдемо до покрокового виконання сортувань масивів. Перш за все, необхідно увійти до середовища демонстрації системи дистанційного навчання WebOAP. Воно має наступний вигляд:



Потрібно натиснути на кнопку «Відкрити колекцію». У діалоговому вікні, що з'явилося, обрати перший вид сортування з наведеного вище переліку (у колекції – 03. Сортування масиву обмінами), натиснути на «Колекція даних» і обрати тут 3 пункти: 01. Масив на 10 елементів, 11. Масив на 100 елементів, розташованих хвилюю та 12. Масив на 200 елементів, розташованих хвилюю.



Після цього натиснути кнопку «ОК» і, не виходячи з колекції системи, повторити для кожного з видів сортувань.

Після натиснення «ОК» у вікні колекції системи перший з обраних алгоритмів відобразиться у лівій панелі екрану.

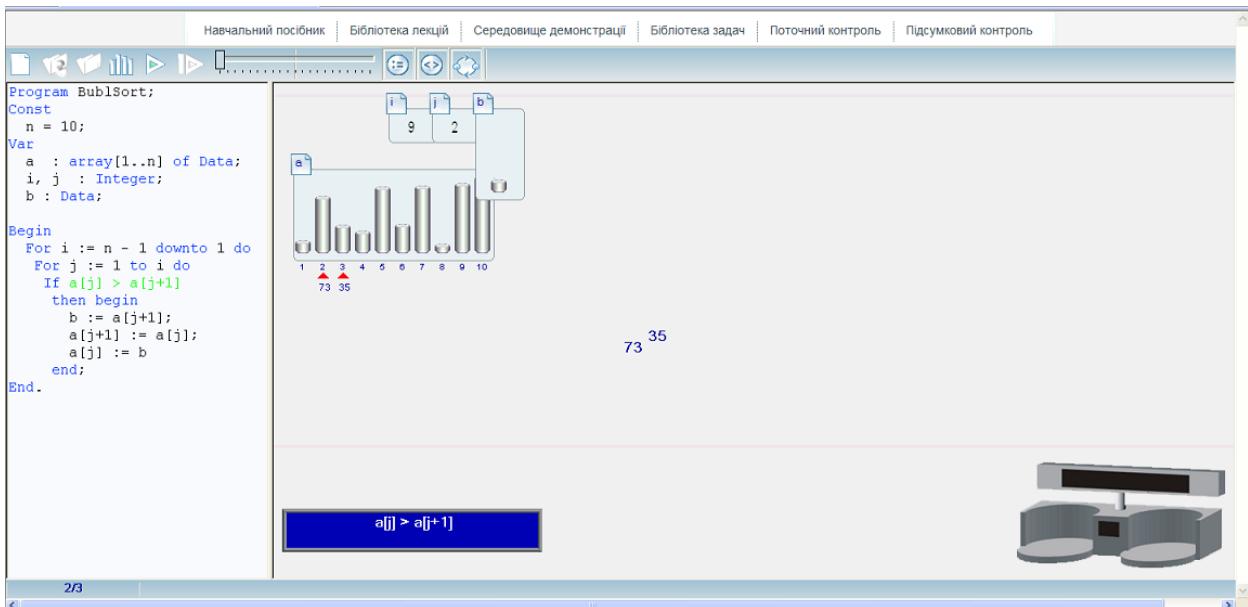
```

Program BublSort;
Const
    n = 10;
Var
    a : array[1..n] of Data;
    i, j : Integer;
    b : Data;

Begin
    For i := n - 1 downto 1 do
        For j := 1 to i do
            If a[j] > a[j+1]
            then begin
                b := a[j+1];
                a[j+1] := a[j];
                a[j] := b
            end;
End.

```

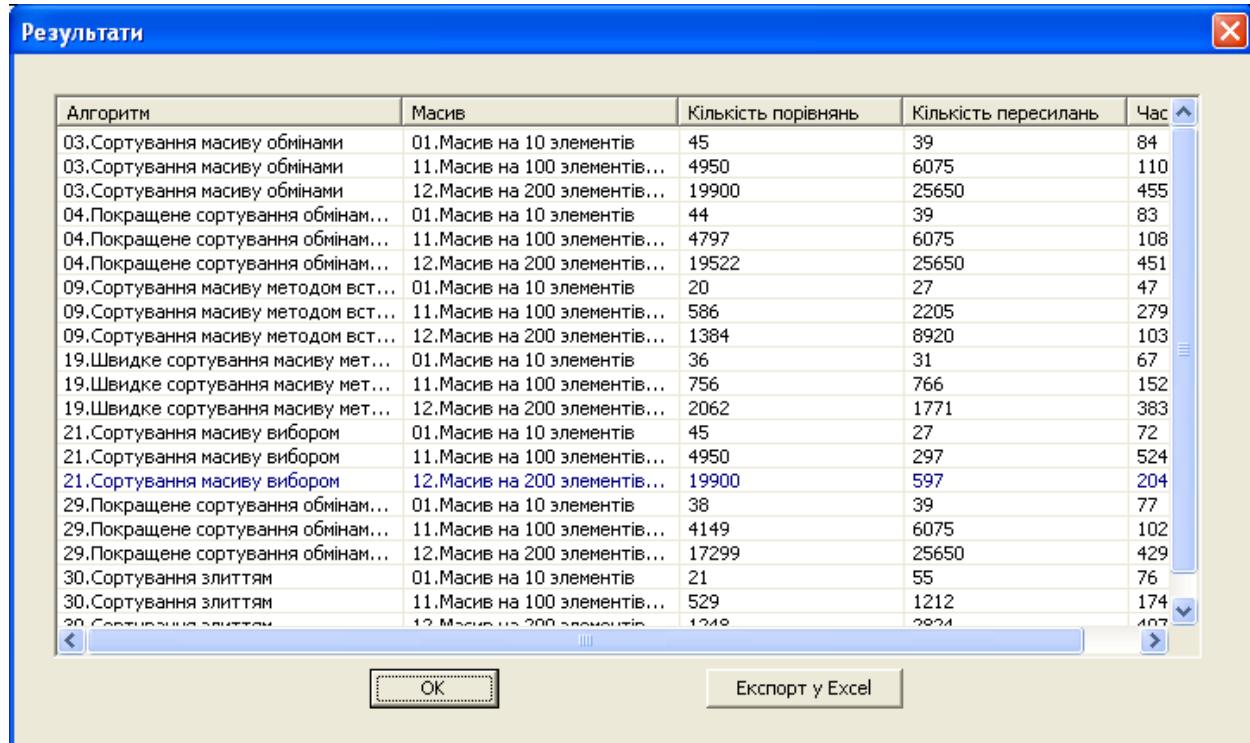
Після натиснення на кнопку «Заповнити даними» () необхідно обрати «Застосувати». Коли змінні заповняться, стане активною кнопка «Почати» (). Після її натиснення на екрані буде відображатися наочне виконання алгоритму.



Отже, після виконання алгоритму ми бачимо кількість порівнянь та пересилань у лівому нижньому куті екрану (). Таким чином ми отримали наступну відповідь: в результаті виконання алгоритму було виконано 2 порівняння та 3 пересилання.

Для того, щоб продовжити виконання всіх алгоритмів сортувань, необхідно знову натиснути на кнопку «Почати» (завантажиться текст алгоритму), потім на кнопку «Заповнити даними» (обрати

«Застосувати») і натиснути на кнопку «Почати». Таким чином потрібно виконати всі заплановані алгоритми, після чого натиснути на кнопку «Почати». На екрані з'явиться вікно з результатами, які потрібно експортувати до Excel (кнопка «Експорт у Excel»).



A	B	C	D	E	F
1 Алгоритм	Масив	Кількість порівнянь	Кількість пересилань	Час	
2 03.Сортування масиву обмінами	01.Масив на 10 елементів	45	39	84	
3 03.Сортування масиву обмінами	11.Масив на 100 елементів розташовані хвилею	4950	6075	11025	
4 03.Сортування масиву обмінами	12.Масив на 200 елементів розташовані хвилею	19900	25650	45550	
5 04.Покращене сортування обмінами (1 варіант)	01.Масив на 10 елементів	44	39	83	
6 04.Покращене сортування обмінами (1 варіант)	11.Масив на 100 елементів розташовані хвилею	4797	6075	10872	
7 04.Покращене сортування обмінами (1 варіант)	12.Масив на 200 елементів розташовані хвилею	19522	25650	45172	
8 09.Сортування масиву методом вставки	01.Масив на 10 елементів	20	27	47	
9 09.Сортування масиву методом вставки	11.Масив на 100 елементів розташовані хвилею	586	2205	2791	
10 09.Сортування масиву методом вставки	12.Масив на 200 елементів розташовані хвилею	1384	8920	10304	
11 19.Швидке сортування масиву методом Хоара	01.Масив на 10 елементів	36	31	67	
12 19.Швидке сортування масиву методом Хоара	11.Масив на 100 елементів розташовані хвилею	756	766	1522	
13 19.Швидке сортування масиву методом Хоара	12.Масив на 200 елементів розташовані хвилею	2062	1771	3833	
14 21.Сортування масиву вибором	01.Масив на 10 елементів	45	27	72	
15 21.Сортування масиву вибором	11.Масив на 100 елементів розташовані хвилею	4950	297	5247	
16 21.Сортування масиву вибором	12.Масив на 200 елементів розташовані хвилею	19900	597	20497	
17 29.Покращене сортування обмінами (2 варіант)	01.Масив на 10 елементів	38	39	77	
18 29.Покращене сортування обмінами (2 варіант)	11.Масив на 100 елементів розташовані хвилею	4149	6075	10224	
19 29.Покращене сортування обмінами (2 варіант)	12.Масив на 200 елементів розташовані хвилею	17299	25650	42949	
20 30.Сортування злиттям	01.Масив на 10 елементів	21	55	76	
21 30.Сортування злиттям	11.Масив на 100 елементів розташовані хвилею	529	1212	1741	
22 30.Сортування злиттям	12.Масив на 200 елементів розташовані хвилею	1248	2824	4072	
23					

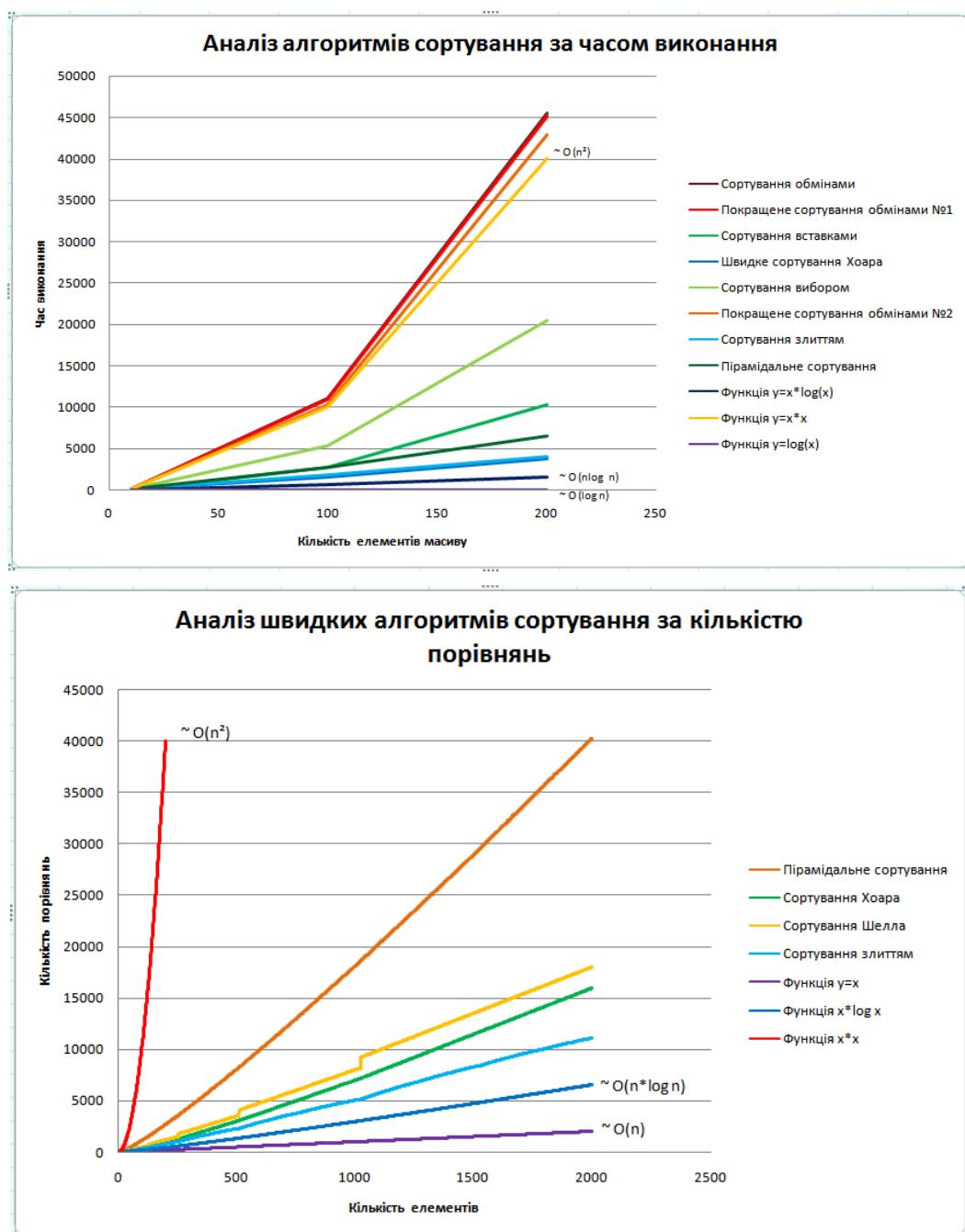
Знову ж таки проведемо аналіз алгоритмів за такими показниками, як час сортування, пам'ять, стійкість, природність поведінки та мажорованість.

Для першого параметру – часу сортування – представимо діаграму з графіками залежності часу виконання від кількості елементів масиву, якщо їх розмірність не перевищує 200.

З аналізу графіків стає зрозумілим, яку складність має кожний алгоритм сортування за кількістю порівнянь.

Однак, в класичній літературі з програмування наголошується, що складні алгоритми сортувань розраховані на роботу з масивами великої розмірності, тому інколи при невиконанні цієї умови ми отримуємо відповідь, що вони працюють гірше за прості сортування. Тож, для доведення даного факту проаналізуємо поведінку чотирьох алгоритмів – сортування Хоара, сортування Шела, сортування злиттям та піраміdalне сортування – на масивах довжиною до 2000 елементів.

Ось що ми отримали в результаті даної задачі:



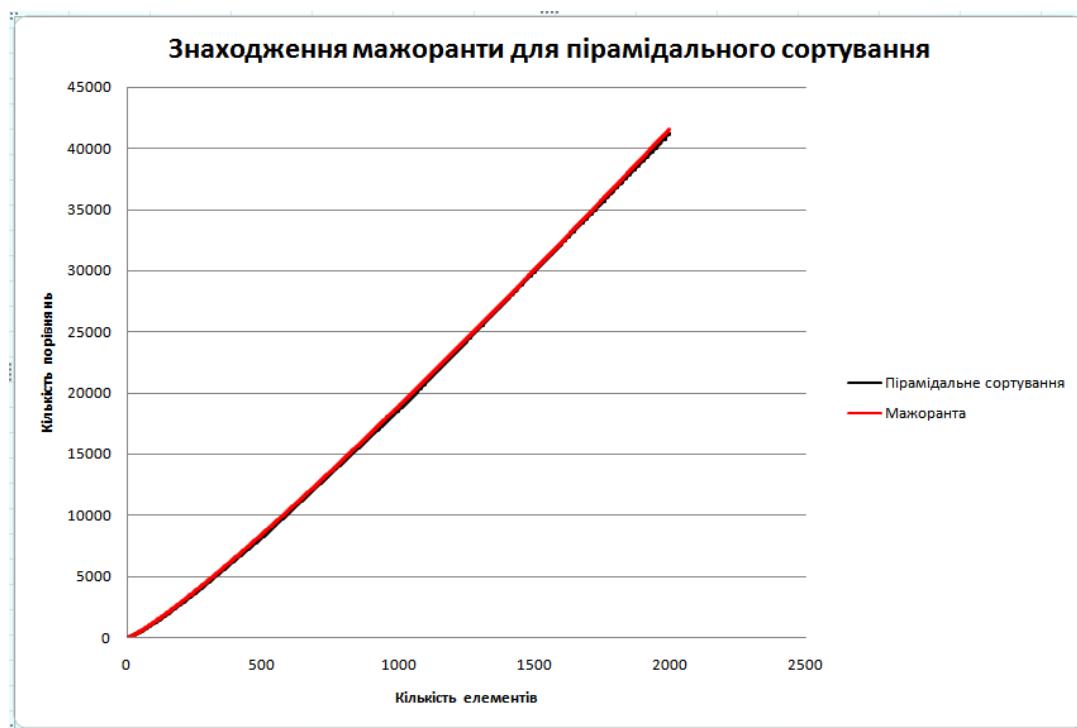
З діаграмами видно, що алгоритми мають складність $O(n \log n)$.

Другий критерій – *пам'ять*. Додатковий масив використовується лише в алгоритмі сортування злиттям. Інші ж алгоритми використовують лише одну додаткову змінну для пересилань елементів масиву.

Стійкість. Не змінили взаємного розташування елементів у вже відсортованому масиві даних сортування лише сортування обмінами, перший та другий його покращені варіанти, динамічне сортування та алгоритм Шела. Тож саме вони є стійкими і у разі використання масивів з елементами складної структури дозволять значно зекономити час сортування.

Природністю поведінки відзначилися алгоритми сортування обмінами, перший та другий його покращені варіації та алгоритм Шела. Вони є ефективними при обробці вже відсортованих даних.

Мажорованість. Використовуючи дані, представлені на рисунку, можна знайти таку функцію мажоранту $f(n)$. Наприклад, для піраміdalного сортування константа c (відношення кількості порівнянь до функції $y=n \cdot \log(n)$ для відповідної кількості елементів) наближено дорівнює 6,3, а для сортування Хоара – 2,9. Тепер для знаходження мажоранти необхідно помножити цю константу c на значення функції $y=n \cdot \log(n)$ для відповідних кількостей елементів та побудувати графіки. Наприклад, для піраміdalного сортування вони будуть мати наступний вигляд:



Отже, мажорантою для піраміdalного сортування за кількістю порівнянь буде функція $y=6,3 \cdot n \cdot \log(n)$.

Таким чином, нам вдалося практичним шляхом довести твердження Кнута про те, що «при $N=1000$ значення середнього часу виконання дорівнюють приблизно

160000и для піраміdalного сортування,

130000и для сортування методом Шела,

80000и для швидкого сортування (Хоара)»

Мова тут не йде про числові показники, проте чітко видно, що відносність розташування та ефективності алгоритмів сортування співпадає з наведеним зауваженням.

Задачі для самостійного розв'язування

1. Виконайте у середовищі демонстрації алгоритм сортування Хоара для масивів з кількістю елементів від 10 до 100 з кроком 10. Побудуйте графіки залежності кількості порівнянь та пересилань від кількості елементів масиву. Побудуйте функцію мажоранту для даного алгоритму сортування за кількістю порівнянь та пересилань. Зробіть висновки щодо ефективності алгоритму.
2. Виконайте у середовищі демонстрації алгоритм сортування Шелла для масивів з кількістю елементів від 10 до 100 та кроком 20. Побудуйте функцію мажоранту для даного алгоритму за часом виконання. Зробіть висновки щодо ефективності алгоритму.
3. Виконайте у середовищі демонстрації алгоритм сортування злиттям для масивів з кількістю елементів від 10 до 200 з кроком 15. Побудуйте графіки залежності кількості порівнянь та часу виконання від кількості елементів масиву. Побудуйте функцію мажоранту для даного алгоритму сортування за цими показниками. Зробіть висновки щодо ефективності алгоритму.

ІНСТРУКЦІЯ КОРИСТУВАЧА ПРОГРАМНОГО ЗАСОБУ ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ КУРСУ «ОСНОВИ АЛГОРИТМІЗАЦІЇ ТА ПРОГРАМУВАННЯ» ДЛЯ ВІЩИХ НАВЧАЛЬНИХ ЗАКЛАДІВ

1. ПРИЗНАЧЕННЯ ПРОГРАМНОГО ЗАСОБУ

Інтегроване середовище вивчення курсу "Основи алгоритмізації та програмування" для вищих навчальних закладів створено для студентів педагогічних, технічних та економічних напрямків і призначено для використання у лекційно-аудиторній, дистанційній та заочній формах навчання. Основною перевагою середовища є організація самостійної роботи та поточний і підсумковий контроль знань студентів. Середовище надає як викладачу, так і студентам усі можливості ефективного вивчення курсу з основ алгоритмізації та програмування.

Користувачами програмного засобу є учні старших класів та вчителі загальноосвітніх навчальних закладів, студенти вищих навчальних закладів, які вивчають основи алгоритмізації і програмування, викладачі.

Робочою мовою програмування в курсі основ алгоритмізації та програмування обрана навчальна мова Паскаль.

Клас задач інтегрованого середовища курсу «Основи алгоритмізації та програмування» – алгоритми обробки масивів даних, у тому числі сортування, пошук унікальних елементів (максимуми, мінімуми і т.п.). Утім, використання програми не обмежено лише цим класом задач.

Програмний засіб «Інтегроване середовище вивчення курсу «Основи алгоритмізації та програмування» для вищих навчальних закладів» працює в операційному середовищі Windows 2003 Server та подальших його версіях. Документи системи зберігаються в базі даних My SQL Server 5, а також у файлах формату XML, HTML та документах Microsoft Office. Система має архітектуру Клієнт-Сервер. Система встановлюється з компакт-диску і інсталюється та експлуатується на персональному комп'ютері – сервері для забезпечення роботи користувачів в мережі Інтернет, або в комп'ютерному класі, обладнаному локальною мережею.

Крім того, необхідні:

- знання даної інструкції в повному обсязі;
- вміння працювати в текстовому редакторі;
- спеціальні знання відповідних засобів мови програмування Паскаль, які студенти мають опановувати в процесі використання даного програмного засобу під час вивчення розділу «Основи алгоритмізації та програмування» шкільного курсу «Основи інформатики».

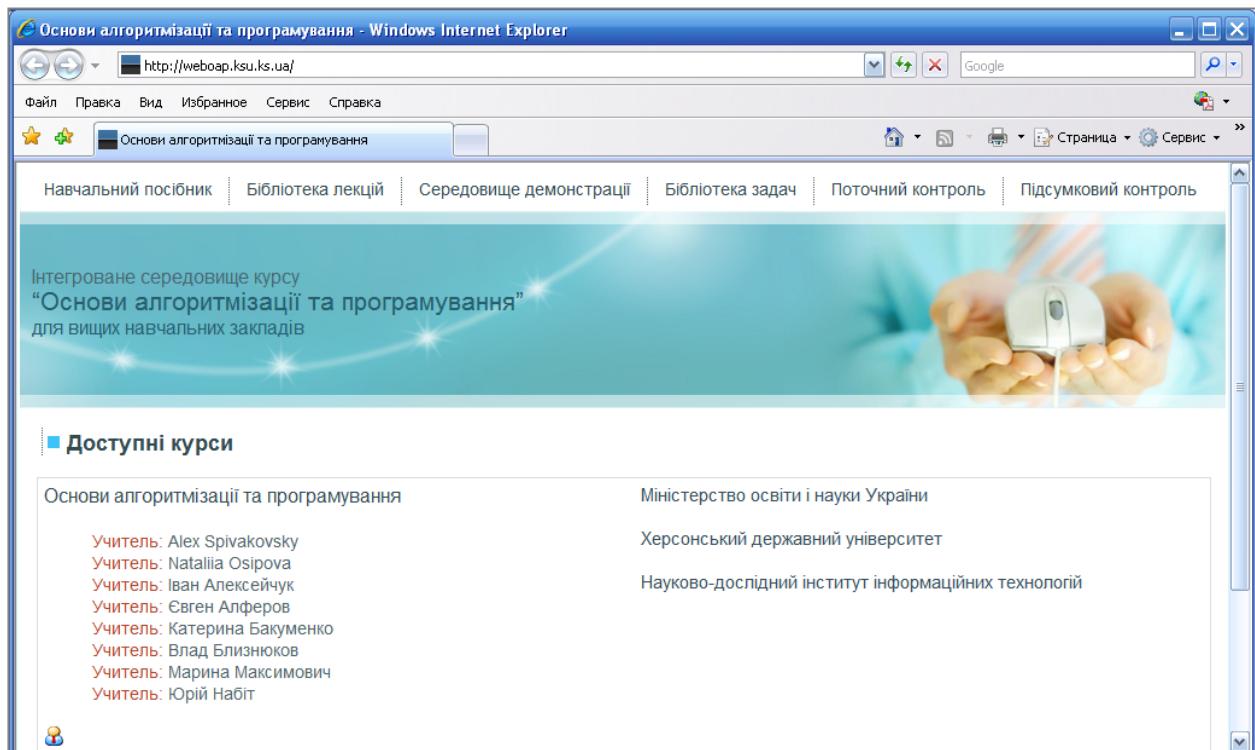
Інтегроване середовище вивчення курсу «Основи алгоритмізації та програмування» реалізовано, як Web-додаток, для використання на лекційних та лабораторних заняттях, для організації самостійної роботи студентів ВНЗ в аудиторіях, обладнаних мережею. Головна особливість програмного засобу полягає у врахуванні специфіки предметної області та у реалізації за єдиною методологією та у взаємодії усіх електронних засобів навчання: електронного посібника, задачника, середовища демонстрації програм, системи поточного та підсумкового контролю знань, що містить алгоритмічні тести.

При розробці інтегрованого середовища використано архітектурні та технологічні рішення, застосовані у середовищі «Програмно-методичний комплекс «Відеоінтерпретатор алгоритмів пошуку та сортування»», ПМК «Лінійна алгебра», розроблених НДІ інформаційних технологій Херсонського державного університету.

2. ЗАПУСК ПРОГРАМНОГО ЗАСОБУ

2.1 Запуск програмного засобу

Для того, щоб увійти до інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування» достатньо використати стандартне програмне забезпечення на стороні клієнта (Web-браузер Internet Explorer). В адресному рядку браузера ввести адресу сайту <http://weboap.ksu.ks.ua/>.



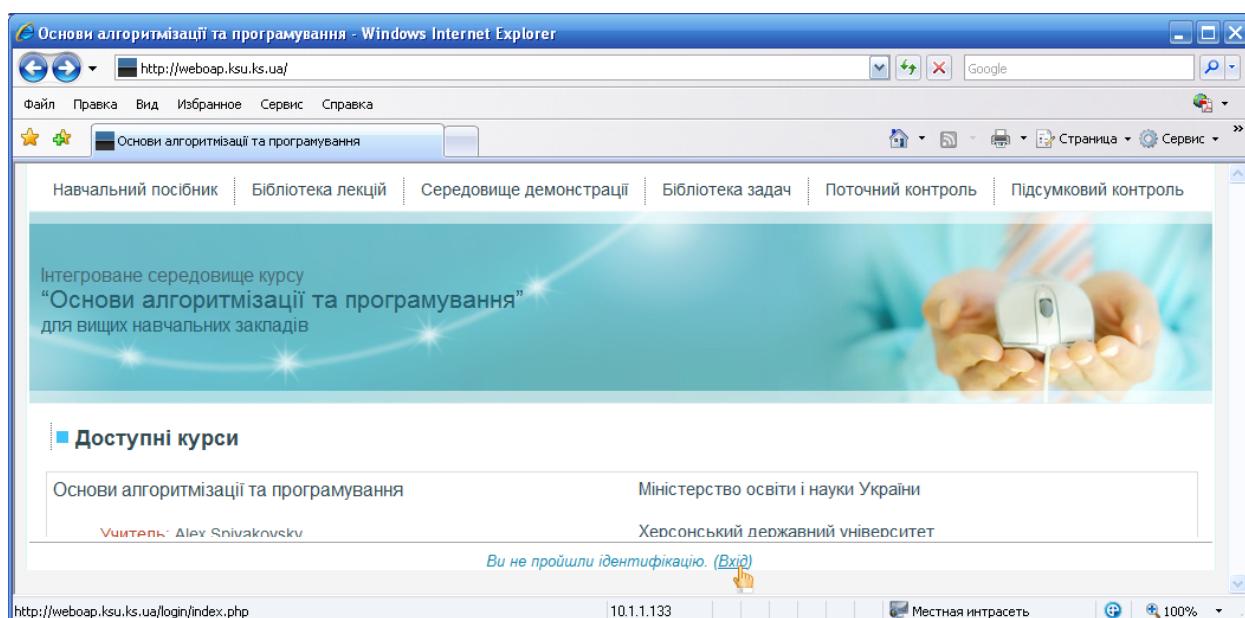
При запуску відкривається головна сторінка (головне вікно) середовища (рис. 1), на якій користувач:

- Здійснює ідентифікацію.
- Вибирає той модуль, з якого він починає роботу. Модулі середовища:
 - ⇒ Навчальний посібник «Основи алгоритмізації та програмування».
 - ⇒ Бібліотека лекцій.
 - ⇒ Середовище демонстрації.

- ⇒ Бібліотека задач.
- ⇒ Поточний контроль.
- ⇒ Підсумковий контроль.

2.2 Ідентифікація користувача

Програмний модуль «Система ідентифікації» забезпечує реєстрацію користувачів, надання їм прав доступу (адміністратора, викладача, викладача без права редагування та студента), захист даних системи від несанкціонованого доступу.



За допомогою посилання *Вхід* здійснюється перехід до сторінки, на якій можна зареєструватися на курс «Основи алгоритмізації та програмування» або відкрити курс, якщо користувач вже зареєстрований.

Для доступу до курсу користувачу необхідно створити обліковий запис на сайті. Для цього необхідно заповнити новий обліковий запис (форму, що містить відомості про користувача). Після підтвердження реєстрації адміністратором на вказаний e-mail буде відправлено листа.

Для доступу до всіх ресурсів курсу на сторінці *Логін для сайту* необхідно ввести ім'я користувача та пароль і натиснути кнопку *Вхід*.

Для курсу «Основи алгоритмізації та програмування» тимчасово передбачений гостьовий доступ.

Основи алгоритмізації та програмування: Логін для сайту - Windows Internet Explorer

http://weboap.ksu.ks.ua/login/index.php

Файл Правка Вид Избранное Сервис Справка

О.А.П. << Логін для сайту

Ви не пройшли ідентифікацію. (Вхід)

Інтегроване середовище курсу
“Основи алгоритмізації та програмування”
для вищих навчальних закладів

Навчальний посібник | Бібліотека лекцій | Середовище демонстрації | Бібліотека задач | Поточний контроль | Підсумковий контроль

Вхід у систему ДН

Тут використовується ваше ім'я користувача, логін і пароль
(Cookies повинні бути дозволені у Вашому браузері)

Ім'я користувача
Пароль [Вхід]

Деякі курси, можливо, дозволяють гостевий доступ
[Логін для гостя]

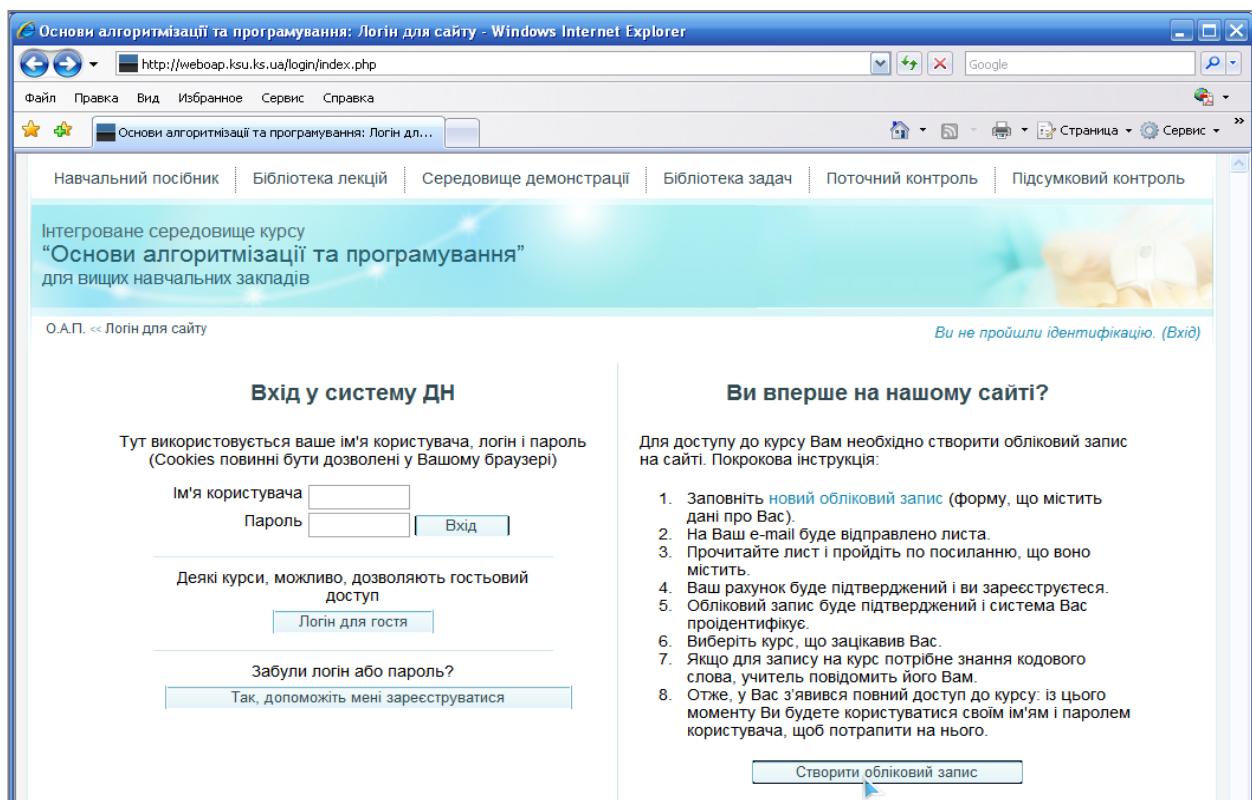
Забули логін або пароль?
[Так, допоможіть мені зареєструватися]

Ви вперше на нашому сайті?

Для доступу до курсу Вам необхідно створити обліковий запис на сайті. Покрокова інструкція:

1. Заповніть **новий обліковий запис** (форму, що містить дані про Вас).
2. На Ваш e-mail буде відправлено листа.
3. Прочитайте лист і пройдіть по посиланню, що воно містить.
4. Ваш рахунок буде підтверджений і ви зареєструєтесь.
5. Обліковий запис буде підтверджений і система Вас пройдентирикує.
6. Виберіть курс, що зацікавив Вас.
7. Якщо для запису на курс потрібне знання кодового слова, учитель повідомить його Вам.
8. Отже, у Вас з'явиться повний доступ до курсу: із цього моменту Ви будете користуватися своїм ім'ям і паролем користувача, щоб потрапити на нього.

[Створити обліковий запис]



Новий акаунт - Windows Internet Explorer

http://weboap.ksu.ks.ua/login/signup.php

Файл Правка Вид Избранное Сервис Справка

О.А.П. << Вхід << Новий акаунт

Ви не пройшли ідентифікацію. (Вхід)

Інтегроване середовище курсу
“Основи алгоритмізації та програмування”
для вищих навчальних закладів

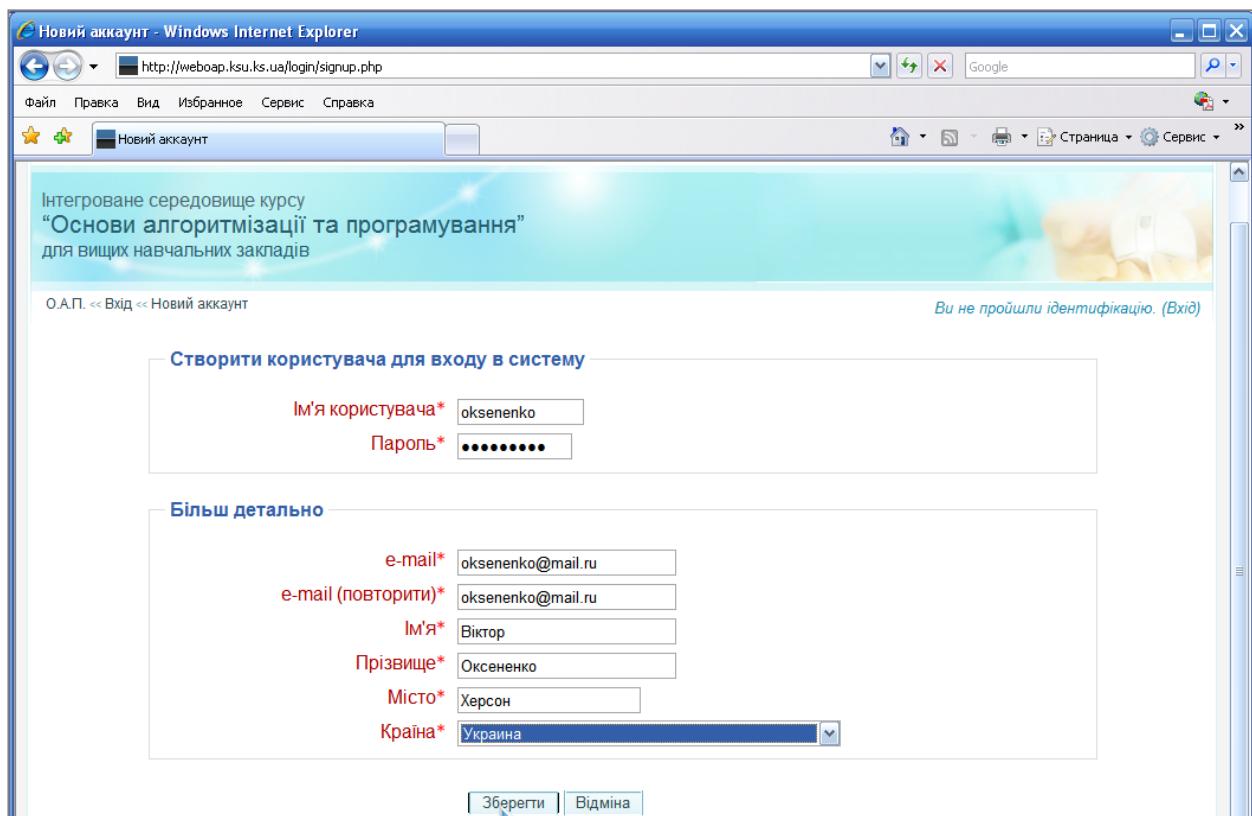
Створити користувача для входу в систему

Ім'я користувача*
Пароль*

Більш детально

e-mail*
e-mail (повторити)*
Ім'я*
Прізвище*
Місто*
Країна*

[Зберегти] [Відміна]



2.3 Вибір модуля середовища для подальшої роботи

Курс «Основи алгоритмізації та програмування» складається з наступних модулів «Навчальний посібник», «Бібліотека лекцій», «Середовище демонстрації», «Бібліотека задач», «Поточний контроль» та «Підсумковий контроль».

Щоб обрати модуль середовища, в якому користувач бажає працювати, необхідно виконати одну з наступних дій:

1. Обрати відповідний пункт горизонтального меню, що розташоване у верхній частині web-сторінки.
2. На головній сторінці інтегрованого середовища обрати курс «Основи алгоритмізації та програмування». У вікні даного курсу у розділі «Заголовки тем» натиснути на посилання обраного модуля.
3. У вікні курсу «Основи алгоритмізації та програмування» у списку «Елементи курсу» обрати певний розділ.

The screenshot shows a Windows Internet Explorer window with the following details:

- Title Bar:** Курс: Основи алгоритмізації та програмування - Windows Internet Explorer
- Address Bar:** http://weboap.ksu.ks.ua/course/view.php?id=2&sesskey=RxObdCVfd&switchrole=5
- Menu Bar:** Файл Правка Вид Избранное Сервис Справка
- Toolbar:** Курс: Основи алгоритмізації та програмування
- Header:** Навчальний посібник | Бібліотека лекцій | Середовище демонстрації | Бібліотека задач | Поточний контроль | Підсумковий контроль
- Main Content Area:**
 - Курси:** Основи алгоритмізації та програмування | Всі курси ...
 - Елементи курсу:** Бази даних | Задачники | Книги | Середовища демонстрації | Тести | Форуми
 - Управління:** Призначити ролі | Оцінки | Виключите мене з ОАП
 - Заголовки тем:**
 - Форум новин
 - Навчальний посібник "Основи алгоритмізації та програмування"
 - Бібліотека лекцій з основ алгоритмізації та програмування
 - Середовище демонстрації програм
 - Бібліотека задач з основ алгоритмізації та програмування
 - Лінійка тем:**
 - Тема 1. Алгоритми
 - Тема 2. Комп'ютери та програми
 - Тема 3. Мова програмування Pascal
 - Тема 4. Прості типи даних. Лінійні програми
 - Тема 5. Процедурне програмування
 - Тема 6. Програмування розгалужень
 - ...
- Right Sidebar:**
 - Користувачі на сайті:** (останні 5 хвилин) Nataliia Osipova
 - Останні Новини:** 16 окт 14:30 Катерина Бакуменко Відправка задачі ще... Старі теми ...
- Bottom Status Bar:** Выполнено, но с ошибками на странице. 10.1.1.133 | Местная интрасеть | 100% |

2.4 Вихід із середовища

Щоб закінчiti роботу в інтегрованому середовищі вивчення курсу «Основи алгоритмізації та програмування», треба натиснути на посилання «Вихід», яке розміщене під ім'ям та прізвищем користувача.

Курс: Основи алгоритмізації та програмування - Windows Internet Explorer
http://weboap.ksu.ks.ua/course/view.php

Файл Правка Вид Избранное Сервис Справка

Навчальний посібник | Бібліотека лекцій | Середовище демонстрації | Бібліотека задач | Поточний контроль | Підсумковий контроль

Інтегроване середовище курсу
«Основи алгоритмізації та програмування»
для вищих навчальних закладів

О.А.П. << ОАП

Доброого дня, Natalia Osipova
[Вихід](#)

Користувачі на сайті
(останні 5 хвилин)
Natalia Osipova

Останні Новини
Додати нову тему...
16 окт 14:30
Катерина Бакуменко
Відправка задачі ще...
Старі теми ...

■ Курси
■ Основи алгоритмізації та програмування
Всі курси ...

■ Елементи курсу
■ Бази даних
■ Задачники
■ Книги
■ Середовища демонстрації
■ Тести
■ Форуми

■ Управління
■ Редагувати
■ Параметри
■ Призначити ролі
■ Групи
■ Резервне копіювання
■ Відновлення

■ Заголовки тем

1 Тема 1. Алгоритми
2 Тема 2. Комп'ютери та програми
3 Тема 3. Мова програмування Pascal
4 Тема 4. Прості типи даних. Лінійні програми
5 Тема 5. Процедурне програмування
6 Тема 6. Програмування розгалужень

http://weboap.ksu.ks.ua/login/logout.php?sesskey=RxObdCvfd 10.1.1.133 Местная интрасеть 100%

3. МОДУЛЬ «ЕЛЕКТРОННИЙ НАВЧАЛЬНИЙ ПОСІБНИК»

Програмний модуль «Навчальний посібник» – сучасний мультимедійний гіпертекстовий додаток, побудований у вигляді структурованої колекції тем і алгоритмів, зміст якого відповідає програмі з основ алгоритмізації та програмування для ВНЗ.

У лівій частині сторінки розташовано зміст посібника з курсу «Основи алгоритмізації та програмування», а у правій розміщується текст відповідного розділу.

The screenshot shows a Windows Internet Explorer window with the following details:

- Title Bar:** - Windows Internet Explorer
http://weboap.ksu.ksu.ua/mod/book/view.php?id=14
- Menu Bar:** Файл, Правка, Вид, Избранное, Сервис, Справка
- Toolbar:** Back, Forward, Stop, Refresh, Home, Search, Print, Page Number, Services, Page Setup, Help, More
- Address Bar:** Страница, Сервис, >
- Content Area:**
 - Header: Інтегроване середовище курсу "Основи алгоритмізації та програмування" для вищих навчальних закладів
 - Breadcrumbs: О.А.П. << ОАП << Книги << Навчальний посібник "Основи алгоритмізації та програмування"
 - Search: Пошук
 - Left Sidebar (Listed under "Файл" menu):
 - Ф1. Вступ
 - Ф1. Алгоритми
 - Ф2. Комп'ютери і програми
 - Ф3. Мова програмування Pascal
 - Ф4. Прості типи даних. Лінійні програми
 - Ф5. Процедурне програмування
 - Ф6. Програмування розгалужень
 - Ф7. Оператори повторення з параметром.
Масиви
 - Ф8. Ітераційні цикли
 - Ф9. Рекурсія
 - Ф10. Швидкі алгоритми сортування і пошуку
 - Ф11. Складні типи даних: записи і файли
 - Ф12. Множини
 - Ф13. Динамічні структури даних
 - Ф14. Методологія структурного програмування: підсумки
 - Main Content:
 - Основи алгоритмізації та програмування**
 - A graphic of a globe with binary code at its base.

Посібник складається з об'єктів наступних типів: розділ (Chapter), підрозділ (SubChapter), текст (None), приклад (Example), фрагмент (Fragment), задача (Problem), коментар (Commentary).

Розрізняють об'єкти, що містять наступні види контенту: текст (Text); малюнок (Image); лістинг (Listing).

The screenshot shows a Windows Internet Explorer window with the following details:

- Title Bar:** Windows Internet Explorer - http://weboap.ksu.ks.ua/mod/book/view.php?id=14#subject_909
- Menu Bar:** Файл Правка Вид Избранное Сервис Справка
- Toolbar:** Back, Forward, Stop, Refresh, Home, Print, Search, etc.
- Navigation Links:** Навчальний посібник | Бібліотека лекцій | Середовище демонстрації | Бібліотека задач | Поточний контроль | Підсумковий контроль
- Section Header:** Інтегроване середовище курсу
“Основи алгоритмізації та програмування”
для вищих навчальних закладів
- Text:** О.А.П. << ОАП << Книги << Навчальний посібник “Основи алгоритмізації та програмування”
- Search Bar:** Пошук
- Left Sidebar (Mассивы):**
 - Массивы
 - ⊕ 8. Ітераційні цикли
 - ⊕ 9. Рекурсія
 - 10. Швидкі алгоритми сортування і пошуку
 - *****
 - 10.1. Нижня оцінка часу задачі сортування масиву за числом порівнянь
 - 10.2. Швидкі алгоритми сортування: Сортування деревом
 - 10.2.1. *Аналіз складності алгоритму
 - 10.3. Піраміdalne сортування
 - 10.3.1.*Аналіз складності алгоритму
 - 10.4. Швидке сортування Хоара
 - 10.5. Пошук k-того в масиві. Пошук медіані масиву
- Code Editor (Program QuickSort*):**

```

1 Program QuickSort;
2 Const n = 21;
3 Var
4 A : array[1..n] of Data;
5
6 Procedure Swap(i, j : Integer);
7 Var
8 b : Data;
9 Begin
10 b := a[i];
11 a[i] := a[j];
12 a[j] := b
13 End;
14 Procedure Hoare(L, R : Integer);
15 Var

```

Виконати
- Text (Right Panel):**

Таким чином, процедура Hoare, що описана нами, залежить від параметрів k і m - початкового і кінцевого індексів відрізу масиву, що оброблюється.

Найбільш важливе місце алгоритму Хоара – правильне опрацювання моменту закінчення руху покажчиків left, right. Помітте, що в нашій версії переставляються місцями елементи, що дорівнюють x. Якщо в циклах While замінити умови ($A[\text{left}] < x$) і ($A[\text{right}] > x$) на ($A[\text{left}] \leq x$) і ($A[\text{right}] \geq x$), при $x = \text{Max}(A)$ індекси left і right пробіжать весь масив і побіжуть далі! Ускладнення ж умов продовження перегляду погіршить ефективність програми.

Навігація по посібнику здійснюється:

- за змістом;
- з використанням гіперпосилань;
- з використанням функції пошуку.

Навігація за змістом навчального посібника. Щоб переглянути необхідний матеріал, потрібно розкрити відповідний розділ у структурі тем посібника за допомогою натиснення на «+» (знак плюс), що розташований перед назвою потрібного розділу. Розкриється відповідний список підрозділів. Після вибору необхідного підрозділу (натисненням на його назві) у правій частині вікна відкриється відповідний йому текст.

Навігація з використанням гіперпосилань. Електронний навчальний посібник забезпечує нелінійний перегляд матеріалу та

створення індивідуальної траєкторії ознайомлення з матеріалом за допомогою гіперпосилань. Ступінь деталізації тексту посібника регулюється згортанням та розгортанням окремих його об'єктів (прикладів, лістингів та інших) методом клацання на назві об'єкта в тексті посібника.

Навігація з використанням функції пошуку. Для пошуку розділів посібника, у яких зустрічається необхідний фрагмент тексту, необхідно ввести ключові слова у рядок пошуку та натиснути кнопку **Пошук**. У виведеному списку розділів обрати необхідний для перегляду.

Для більш зручного перегляду тексту обраного розділу навчального посібника за допомогою кнопок та можна збільшити область відображення тексту відповідно по горизонталі та вертикалі.

Засоби інтеграції навчального посібника з іншими модулями середовища. У навчальному посібнику забезпечена можливість редактування лістингів (без збереження змін), що дозволяє змінювати тексти алгоритмів при їх демонстрації викладачем. Здійснюється також підсвічування ключових слів для полегшення сприйняття тексту програми студентами.

```
1 Procedure Picture;
2 Procedure Power(X: Real; n: Integer; Var u, v: Real);
3 Procedure Integral( a, b, epsilon: Real; Var S: Real);
```

Головною особливістю навчального посібника є його інтеграція з модулем середовище демонстрації. Приклади програм, що наведені у тексті навчального посібника можуть бути експортовані до середовища демонстрації натисненням на кнопку **Виконати**, що розташована безпосередньо під прикладом. Необхідно зауважити, що у середовище демонстрації можна завантажити і відредактований користувачем алгоритм. Така функція робить навчальний посібник «живим», так як викладач має змогу продемонструвати роботу наведених у певному розділі прикладів, а студент краще зрозуміти логіку роботи алгоритмів.

4. МОДУЛЬ «БІБЛІОТЕКА ЛЕКЦІЙ»

Модуль "Бібліотека лекцій" містить презентації до лекцій, що структуровані за тематичним планом курсу "Основи алгоритмізації та програмування".

Бібліотека лекцій з основ алгоритмізації та програмування

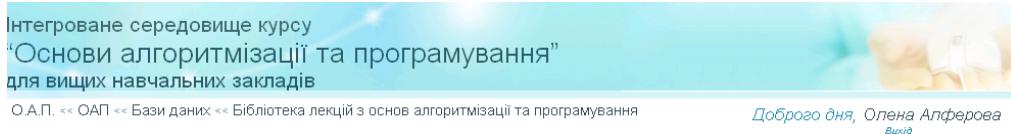
The screenshot shows a web-based library interface. At the top, there are several tabs: 'Проглядання списку' (List view), 'Перегляд по одному запису' (View one record), 'Додати запис' (Add record), 'Шаблони' (Templates), 'Поля' (Fields), and 'Presets'. Below the tabs, a list of lecture topics is displayed in a hierarchical structure. Topics include: Алгоритми та основні поняття, Комп'ютери і програми, Мова програмування Паскаль, Структура програми на мові програмування Паскаль, Процедурне програмування, Програмування розгалужень, Оператори повторення з параметром. Масиви, Ітераційні цикли, Масиви. Алгоритми пошуку і сортування, Рекурсія, Швидкі алгоритми пошуку і сортування, Складні типи даних: записи і файли, Динамічні структури даних, and Методологія структурного програмування. Each topic has a small icon and three small buttons (edit, delete, preview) next to it. At the bottom of the list, there are search and filter options: 'Записів на сторінку' (15), 'Пошук' (Search), 'Сортувати близько' (Sort closest), 'Дата додавання' (Date added), 'За збільшенням' (By increase), and a 'Зберегти настройки' (Save settings) button.

У бібліотеці містяться лекційні матеріали для викладання курсу з основ алгоритмізації та програмування. Представлені лекції з наступних тем:

1. Алгоритми та основні поняття.
2. Комп'ютери і програми.
3. Мова програмування Паскаль.
4. Структура програми мовою програмування Паскаль.
5. Процедурне програмування.
6. Програмування розгалужень.
7. Оператори повторення з параметром. Масиви.
8. Ітераційні цикли.
9. Масиви. Алгоритми пошуку і сортування.
10. Рекурсія.
11. Швидкі алгоритми пошуку і сортування.

12. Складні типи даних: записи і файли.
13. Динамічні структури даних.
14. Методологія структурного програмування.

За допомогою вкладок у нижній частині сторінки можна обрати зручний для користувача варіант перегляду списку тем лекцій.



Бібліотека лекцій з основ алгоритмізації та програмування

[Проглядання списку](#) [Перегляд по одному запису](#) [Додати запис](#)

Сторінка: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 (Наступний)

Лекція: Алгоритми та основні поняття

Сторінка: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 (Наступний)

Записів на сторінку Пошук Сортувати близько Зберегти настройки

Бібліотекою лекцій має змогу користуватися у режимі перегляду як викладач, так і студент. Кожну з презентацій лекцій, які входять до складу "Бібліотеки", можна переглянути або зберегти.

Викладач має змогу редагувати презентації до лекцій, забезпечуючи диференційований підхід до навчання. Викладач також має права для завантаження інших презентацій з тем курсу.



Бібліотека лекцій з основ алгоритмізації та програмування

[Проглядання списку](#) [Перегляд по одному запису](#) [Додати запис](#)

Новий запис

Лекція: Файл
Ім'я файлу (необов'язково)

Всі лекції оформлені за однаковою структурою. Шаблон презентації містить наступні елементи: титульну сторінку; план лекції; слайди, на яких розкривається матеріал; посилання на інші джерела.

5. СЕРЕДОВИЩЕ ДЕМОНСТРАЦІЇ

Середовище демонстрації програм призначене для використання на лекціях, при проведенні практичних занять і лабораторних робіт для наочної демонстрації роботи алгоритмів. Використання модуля «Середовище демонстрації» програм дозволяє більше уваги приділити саме аналізу алгоритмів: на різних масивах даних в результаті виконання демонстрації визначаються основні характеристики – кількість порівнянь та кількість перестановок.

В середовищі демонстрації користувач має можливість вибрати і відкрити алгоритм для демонстрації з колекції системи або з колекції користувача, ініціювати дані демонстрації, налагодити демонстрацію, виконати демонстрацію в неперервному або покроковому режимах.

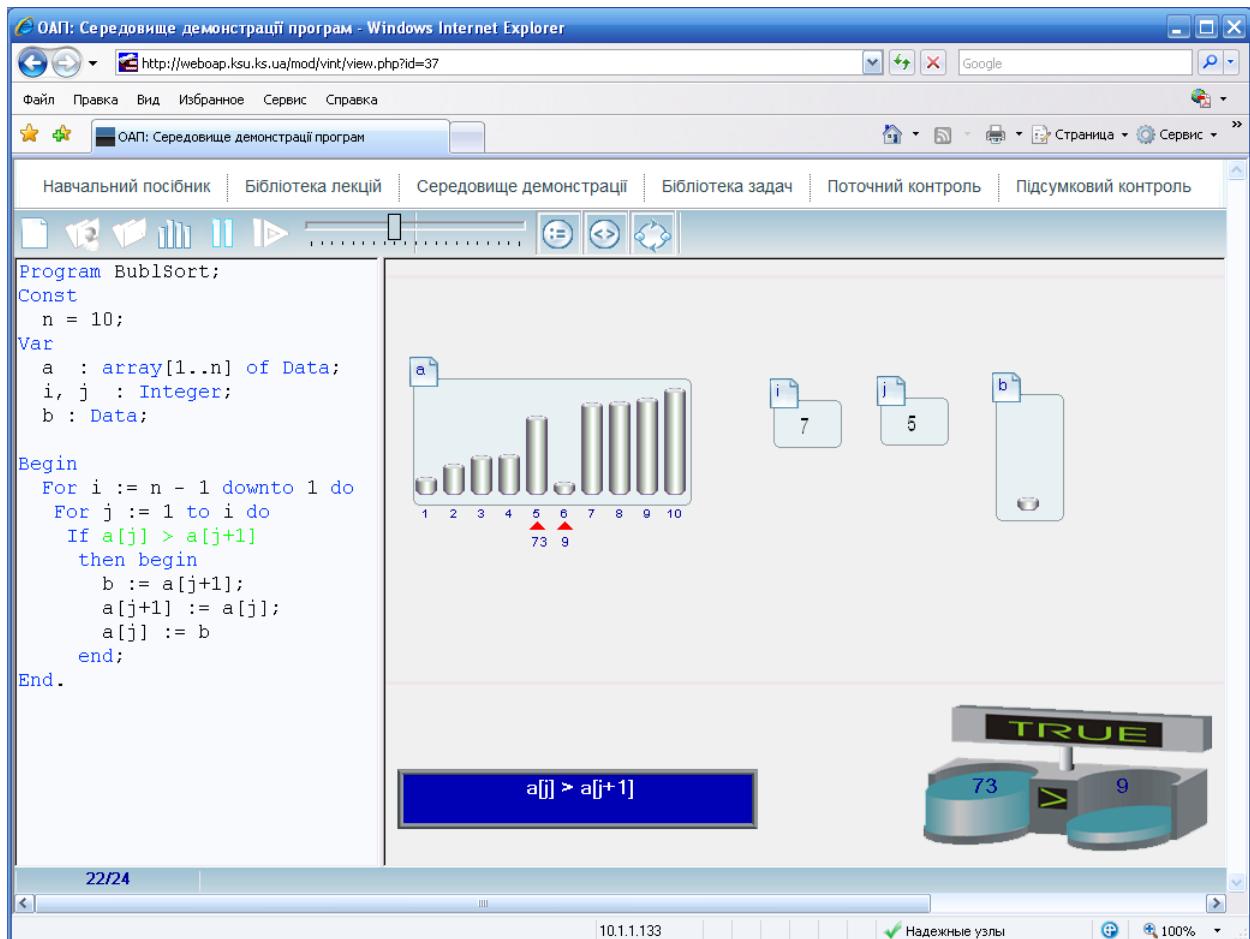
Використання динамічних образів операцій присвоювання, порівняння, передачі параметрів в процедури та функції, рекурсивних викликів процедур та функцій, процесу генерації вхідних даних робить середовище демонстрації виключно корисним засобом вивчення основ алгоритмізації.

Таким чином, завдяки можливостям середовища викладач має змогу урізноманітнити види практичних завдань з алгоритмізації:

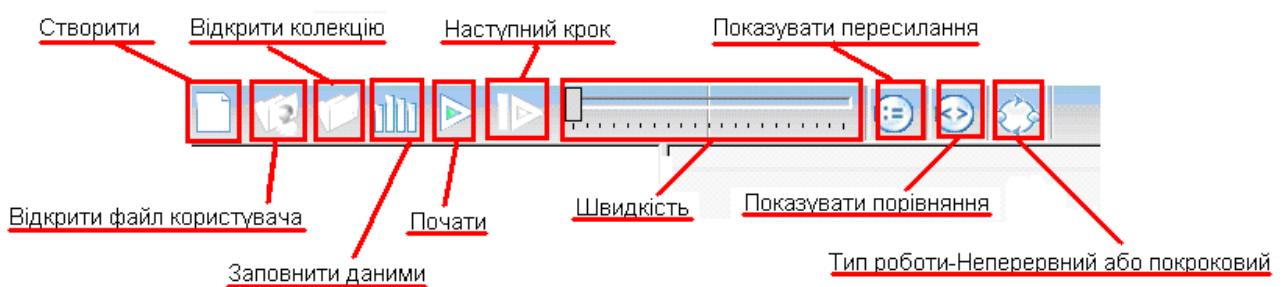
- виконати алгоритм з колекції системи або колекції користувача для певних даних;
- скласти алгоритм розв'язання задачі;
- визначити ефективність алгоритму;
- порівняти ефективність алгоритмів для певного набору даних;
- дослідити та зmodелювати дані для певного алгоритму (випадковим чином, найкращий та найгірший випадки та ін.);
- узагальнити результати аналізу алгоритмів при порівнянні різних методів розв'язання задачі;
- запропонувати більш ефективний алгоритм розв'язання задачі.

Вікно середовища демонстрації складається з наступних елементів: рядок меню, панель інструментів, поле відображення тексту алгоритма, поле візуалізації, рядок стану.

Рядок меню містить назви всіх модулів «Інтегрованого середовища вивчення курсу «Основи алгоритмізації та програмування», що забезпечує легкість навігації.



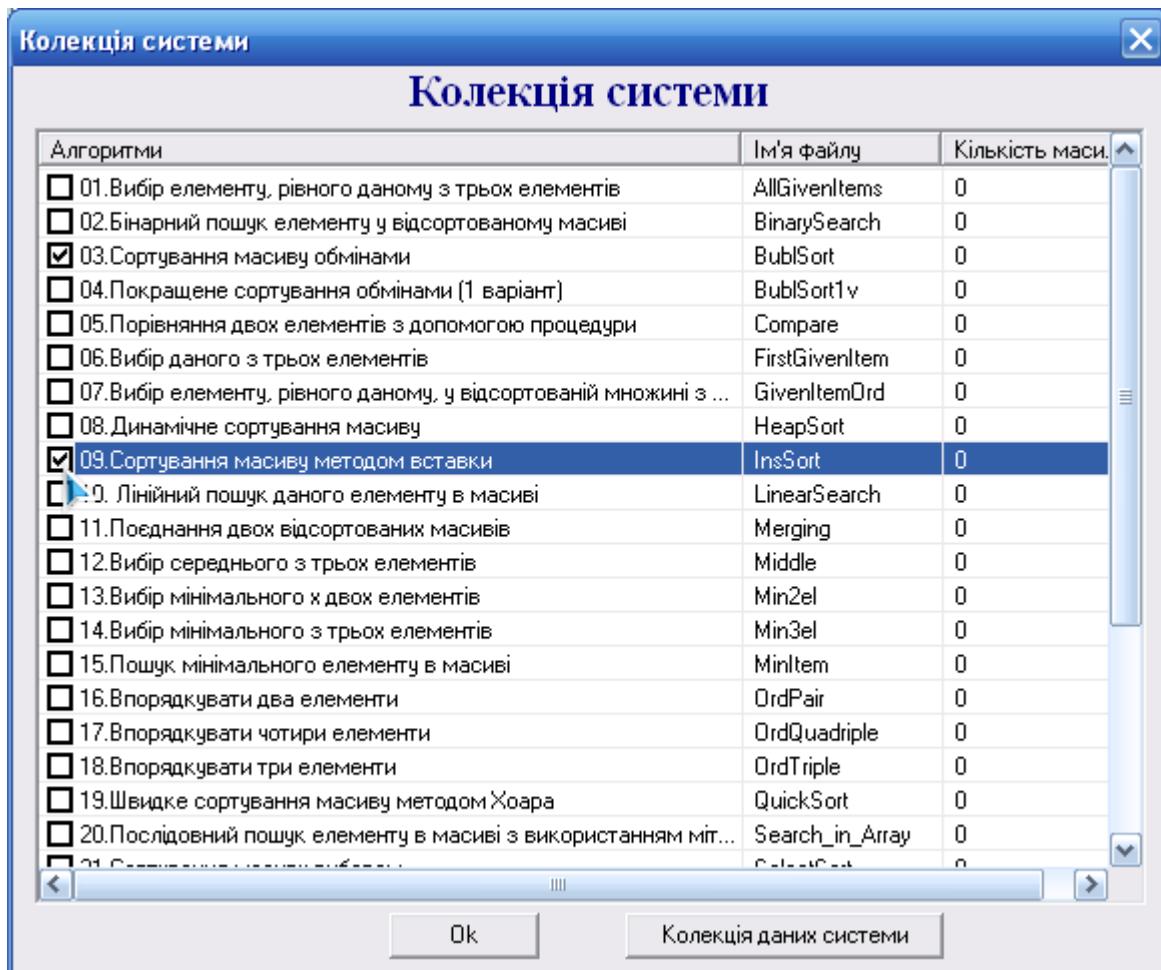
Панель інструментів середовища демонстрації містить наступні кнопки:



Створити – очищує поле відображення тексту алгоритма та поле візуалізації.

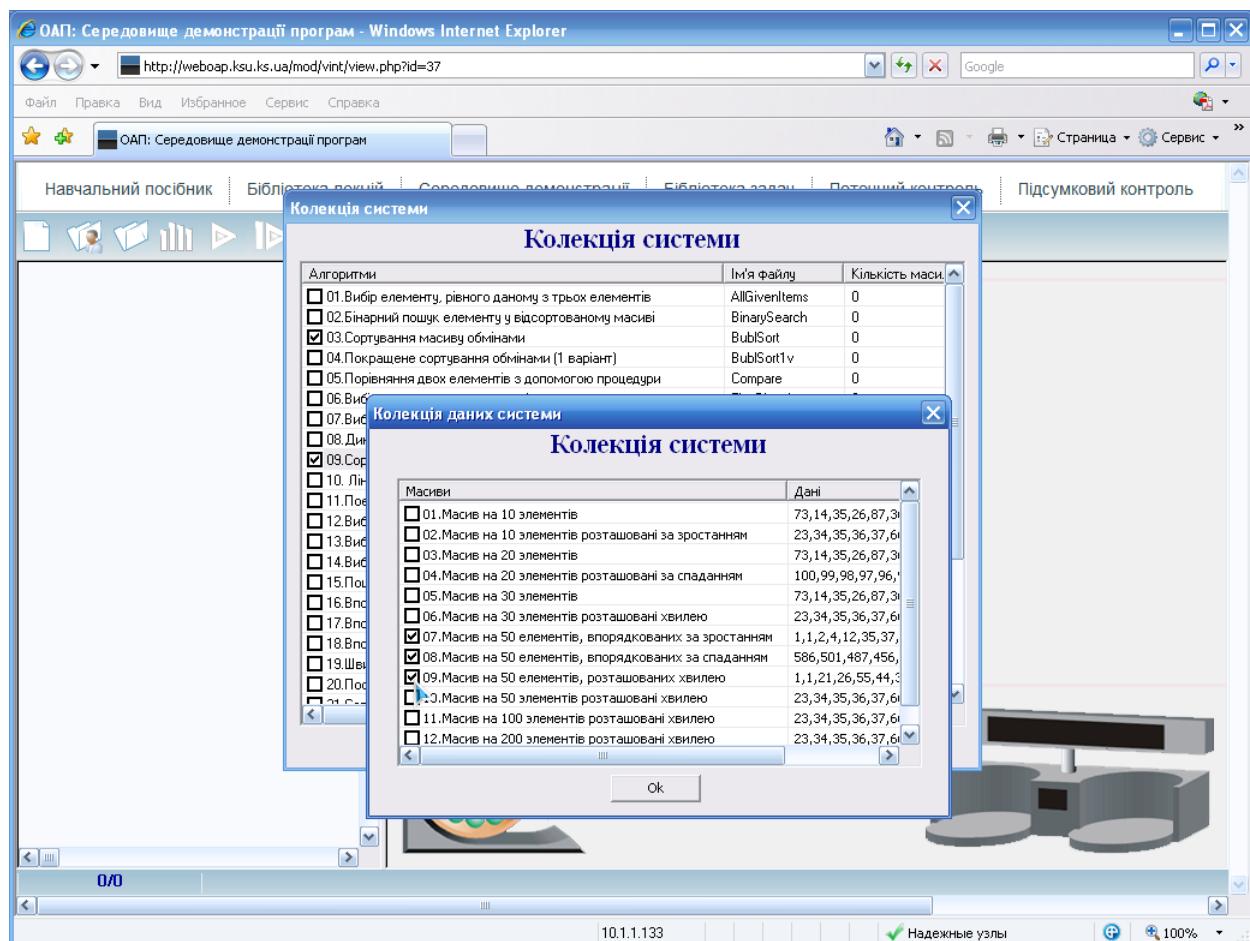
Відкрити файл користувача – відображає діалог, у якому можна вибрати та завантажити у поле відображення тексту алгоритма текстовий файл. При завантаженні текст програми перевіряється на відповідність синтаксису мови програмування Pascal.

Відкрити колекцію – відкриває колекцію системи, що містить основні алгоритми з курсу ОАП.



У вікні *Колекція системи* необхідно обрати один або декілька алгоритмів для візуалізації. Для кожного алгоритму за допомогою кнопки *Колекція даних системи*, розташованої у вікні *Колекція системи*, можна обрати один або декілька наборів даних, на яких буде виконуватися відповідний алгоритм.

Як правило, для аналізу алгоритму необхідно визначити кількість порівнянь та кількість пересилань у кращому, гіршому випадку та у середньому. Тому колекція даних системи містить різні типи масивів: впорядковані за зростанням, за спаданням та з елементами, розташованими хвилею. Змінюючи кількість елементів масиву та, користувач може дослідити, починаючи з якого числа елементів певний вид сортування стає ефективним.

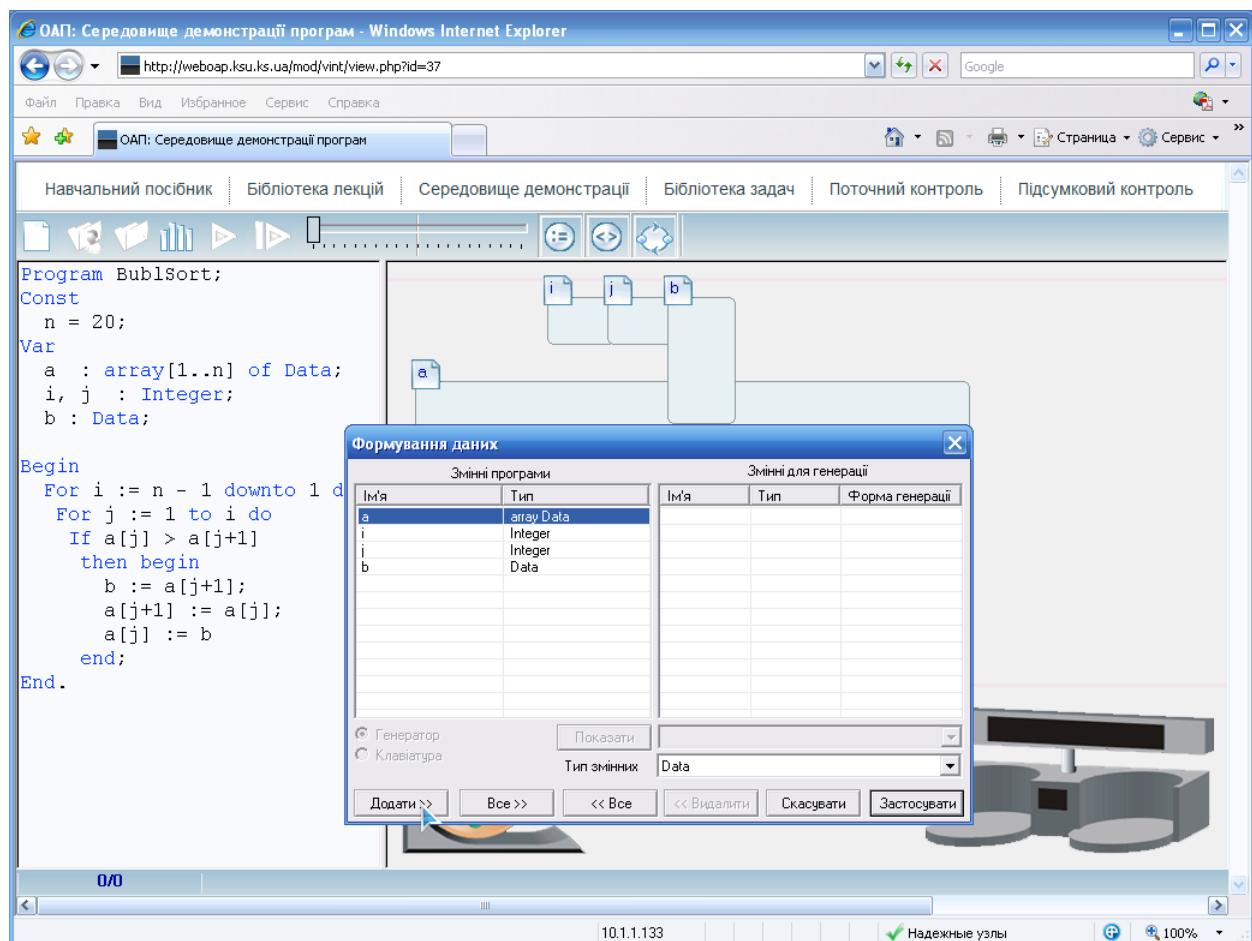


Кожен обраний у колекції системи алгоритм завантажується у середовище демонстрації для візуалізації.

Заповнити даними – формує дані для візуалізації виконання алгоритму у середовищі демонстрації.

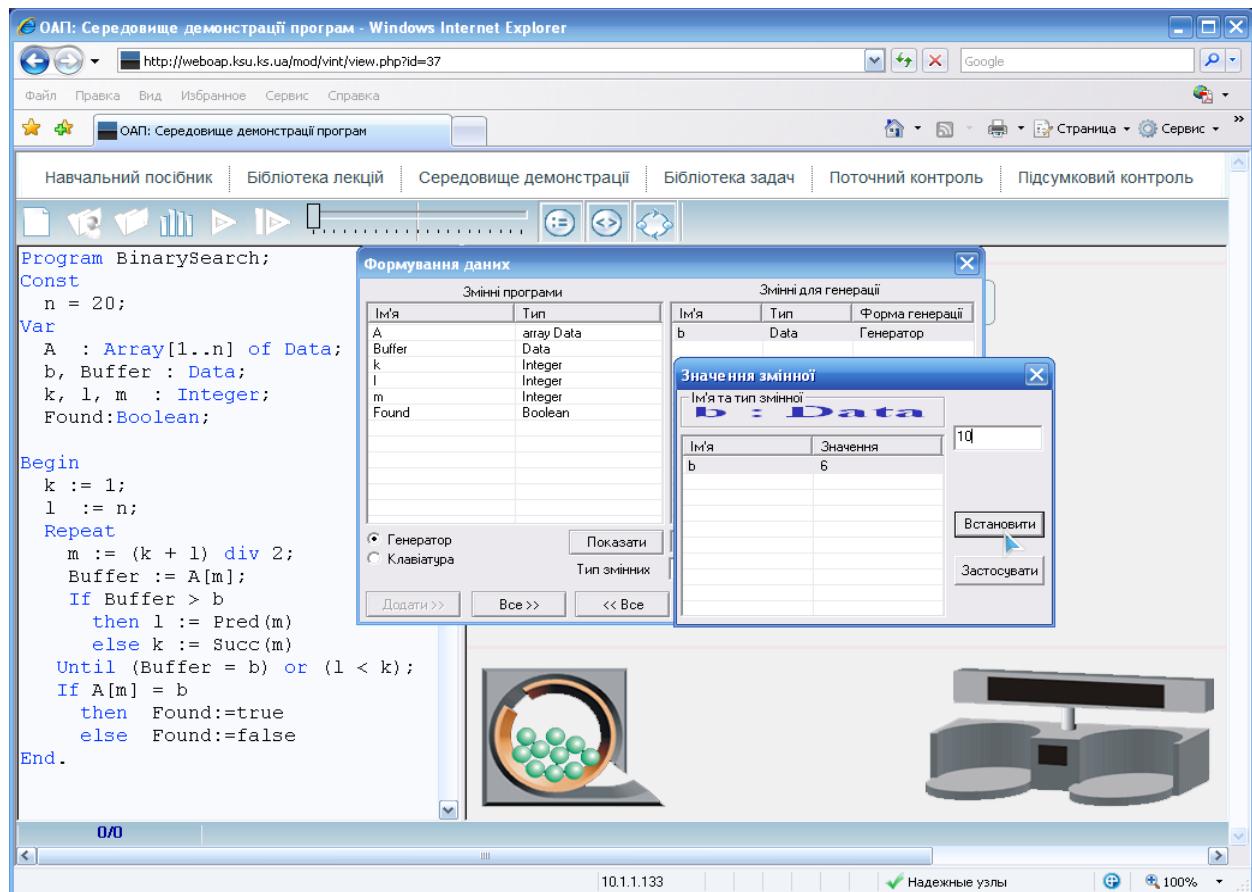
Якщо дані не було обрано з колекції системи, то користувач може обрати вхідні дані алгоритму та сформувати їх одним із способів:

- за зростанням;
 - за спаданням;
 - випадковим чином;
 - введенням даних;
 - завантаженням даних.

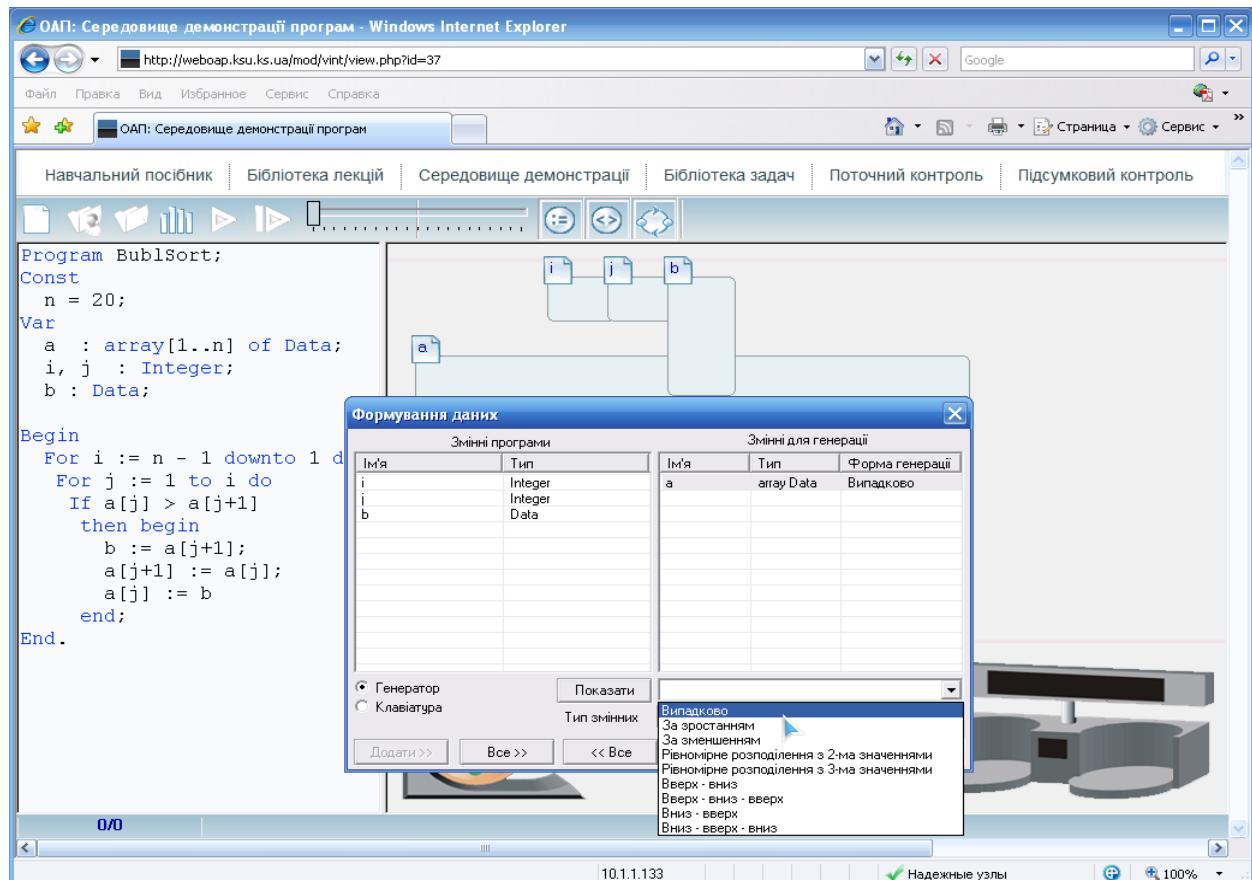


Для формування даних після вибору алгоритма необхідно:

1. Натиснути кнопку *Заповнити даними*.
2. У вікні *Формування даних* у полі *Змінні програми* виділити змінну і натиснути кнопку *Додати*. Змінна буде переміщена у поле *Змінні для генерації*.
3. Виділіть змінну та у полі *Змінні для генерації* встановіть перемикач у позицію *Генератор* для надання їй випадкового значення. Щоб ввести власне значення встановіть перемикач у позицію *Клавіатура* та натисніть кнопку *Показати*. У вікні *Значення змінної* виділіть змінну, введіть нове значення та натисніть кнопки *Встановити* і *Застосувати*.



Якщо у полі *Змінні для генерації* виділено масив, то у списку, що розкривається можна вибрати варіант заповнення масиву.



Почати/Пауза – виконує алгоритм у середовищі демонстрації або тимчасово припиняє його виконання. При виконанні алгоритма у полі відображення тексту алгоритма кольором виділяється рядок, що виконується, а у полі візуалізації демонструється виконання відповідної команди.

Наступний крок – виконує візуалізацію однієї виділеної команди та перехід на наступну команду (активна лише у покровому режимі).

Регулятор швидкості візуалізації алгоритму дозволяє уповільнювати візуалізацію для більш ретельного пояснення роботи алгоритму або пришвидшувати, коли алгоритм зрозумілий та необхідно отримати результати його виконання.

Показувати пересилання – вмикає режим при якому візуалізується процес копіювання значення однієї змінної в іншу. При вимкненні цього режиму візуалізатор відображає тільки результат копіювання значення змінної.

Показувати порівняння – вмикає режим при якому візуалізується процес порівняння значень двох змінних. При вимкненні цього режиму візуалізатор не відображає порівняння значень двох змінних.

Тип роботи – неперервний або покрововий – перемикає неперервний та покрововий режими виконання алгоритму.

У рядку стану на кожному кроці візуалізації алгоритму відображається кількість порівнянь / кількість пересилань.

Натиснення на кнопку Почати після виконання всіх обраних алгоритмів на всіх масивах даних виводить зведену таблицю результатів виконання, у якій відображаються назви алгоритмів, обрані масиви даних, кількість порівнянь, кількість пересилань та час виконання алгоритму.

Отримані дані можна експортувати в програму Microsoft Excel для обробки результатів та графічного їх відображення.

ОАП: Середовище демонстрації програм - Windows Internet Explorer

http://weboap.ksu.ks.ua/mod/vint/view.php?id=37

Файл Правка Вид Избранное Сервис Справка

ОАП: Середовище демонстрації програм Страница Сервис

Навчальний посібник Бібліотека лекцій Середовище демонстрації Бібліотека задач Поточний контроль Підсумковий контроль

Program Quicksort

```

Const
  n = 30;
Var
  A : array[1..n];
Procedure Swap(i)
Var
  b : Data;
Begin
  b := a[i];
  a[i] := a[j];
  a[j] := b;
End;

Procedure Hoare()
Var
  left, right :
  x : Data;
Begin
  If L < R
    then begin
      x := A[(L + R) div 2];
      left := L;
      right := R ;
      Repeat
        End.

```

Результати

Алгоритм	Масив	Кількість порівнянь	Кількість пересилань	Час
03.Сортування масиву обмінами	01.Масив на 10 елементів	45	39	84
03.Сортування масиву обмінами	03.Масив на 20 елементів	190	213	403
03.Сортування масиву обмінами	05.Масив на 30 елементів	435	774	1209
04.Покращене сортування обмінами...	01.Масив на 10 елементів	44	39	83
04.Покращене сортування обмінами...	03.Масив на 20 елементів	187	213	400
04.Покращене сортування обмінами...	05.Масив на 30 елементів	435	774	1209
09.Сортування масиву методом вст...	01.Масив на 10 елементів	20	27	47
09.Сортування масиву методом вст...	03.Масив на 20 елементів	70	104	174
09.Сортування масиву методом вст...	05.Масив на 30 елементів	134	311	445
19.Швидке сортування масиву мет...	01.Масив на 10 елементів	36	31	67
19.Швидке сортування масиву мет...	03.Масив на 20 елементів	143	99	242
19.Швидке сортування масиву мет...	05.Масив на 30 елементів	199	161	360

Бар-графік

OK Експорт у Excel

199/161

10.1.1.133 Надежные узлы 100%

Microsoft Excel - Книга1

Файл Діректория Вид Вставка Формат Сервис Данные Окно Справка Adobe PDF Введите вопрос

SnagIt | Окно

A1 'Алгоритм

A	B	C	D	E
1 Алгоритм	Масив	Кількість порівнянь	Кількість пересилань	Час
2 03.Сортування масиву обмінами	01.Масив на 10 елементів	45	39	84
3 03.Сортування масиву обмінами	03.Масив на 20 елементів	190	213	403
4 03.Сортування масиву обмінами	05.Масив на 30 елементів	435	774	1209
5 04.Покращене сортування обмінами (1 варіант)	01.Масив на 10 елементів	44	39	83
6 04.Покращене сортування обмінами (1 варіант)	03.Масив на 20 елементів	187	213	400
7 04.Покращене сортування обмінами (1 варіант)	05.Масив на 30 елементів	435	774	1209
8 09.Сортування масиву методом вставки	01.Масив на 10 елементів	20	27	47
9 09.Сортування масиву методом вставки	03.Масив на 20 елементів	70	104	174
10 09.Сортування масиву методом вставки	05.Масив на 30 елементів	134	311	445
11 19.Швидке сортування масиву методом Хоара	01.Масив на 10 елементів	36	31	67
12 19.Швидке сортування масиву методом Хоара	03.Масив на 20 елементів	143	99	242
13 19.Швидке сортування масиву методом Хоара	05.Масив на 30 елементів	199	161	360
14				
15				
16				

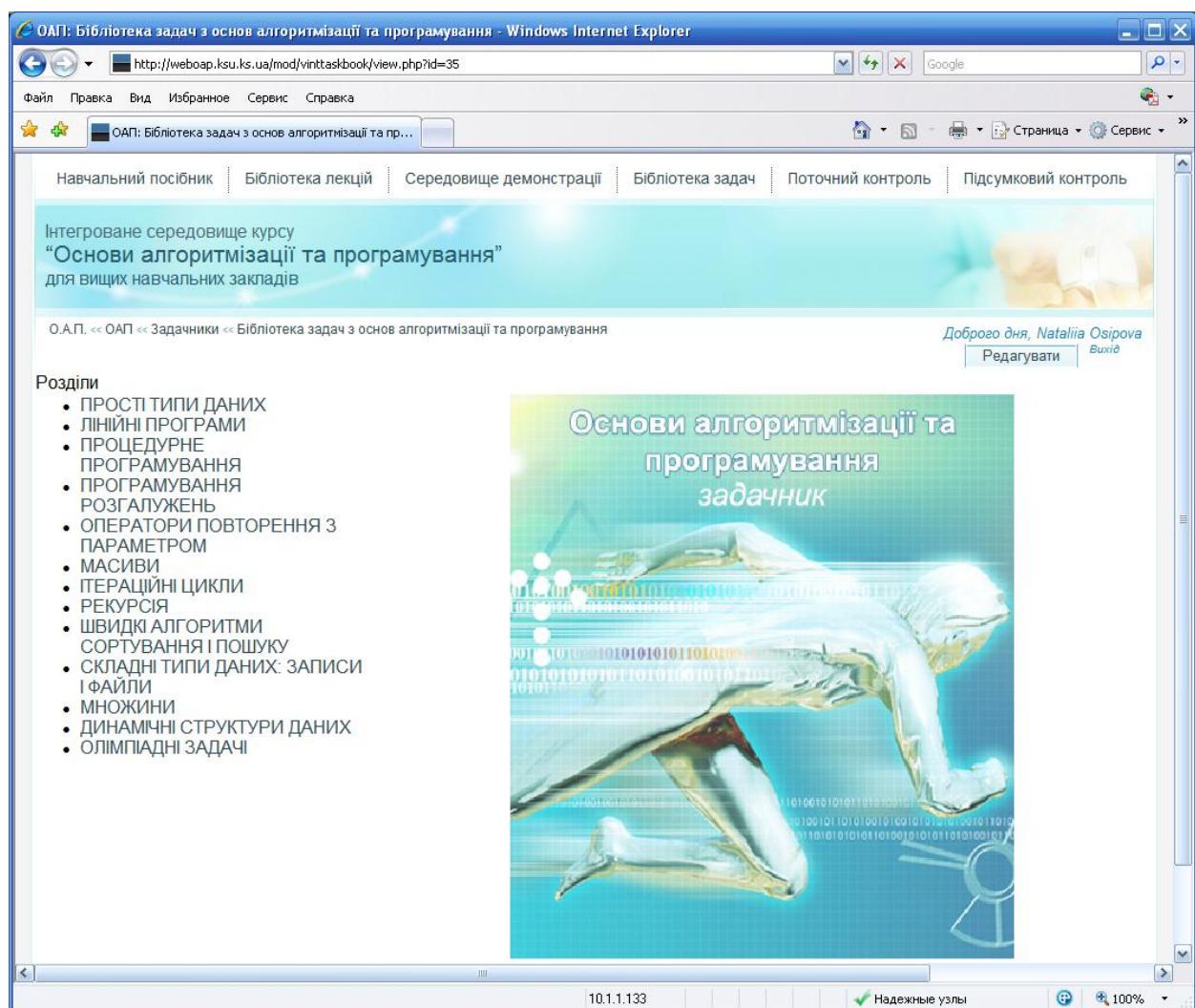
Готово

6. МОДУЛЬ «БІБЛІОТЕКА ЗАДАЧ»

6.1 «Бібліотека задач»

Бібліотека задач інтегрованого середовища представляє собою систему завдань, структуровану у відповідності з тематичним планом навчального курсу та змістом електронного посібника. Кожен розділ задачника відповідає теоретичному матеріалу посібника.

Бібліотека задач інтегрованого середовища призначена для зберігання системи завдань, що підтримуються модулем алгоритмічних тестів. Передбачається, що користувач може відкрити бібліотеку задач, переглянути її, вибрати задачу та розв'язати її, перевіривши за допомогою модуля алгоритмічних тестів.



6.2 Загальний опис інтерфейсу бібліотеки задач

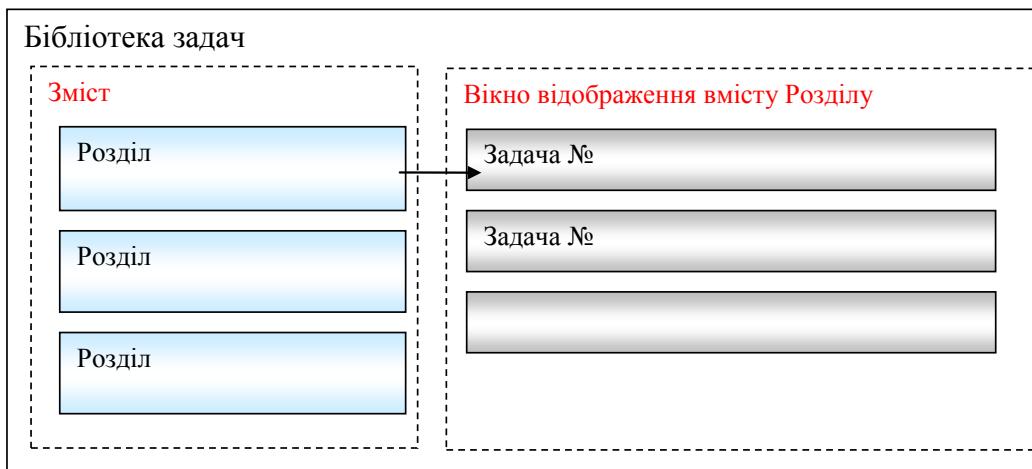
Бібліотека задач побудована за технологією гіпертексту з описами завдань із залученням засобів мультимедіа.

Зручність і наочність навігації по електронним навчальним ресурсам, простота й оперативність переходів до необхідних розділів, об'єктів і засобів навчання є невід'ємною частиною інтерфейсу бібліотеки задач.

Підтримується відображення графічних зображень, а також формул, що дають можливість формулювати задачі з різних предметних областей.

Кожна задача бібліотеки задач має унікальний номер, умову й рівень складності.

6.3 Структура головного вікна бібліотеки задач



Навчальні задачі згруповано в кількох розділах. Розділи містять задачі для розв'язання під час практичних, лабораторних занять, самостійної роботи та задачі, що можуть бути використані при поточному та підсумковому контролі.

Інтегроване середовище курсу
“Основи алгоритмізації та програмування”
для вищих навчальних закладів

О.А.П. << ОАП << Задачники << Бібліотека задач з основ алгоритмізації та програмування

Добого дня, Natalia Osipova
Вихід Редактувати

Розділи

- ПРОСТИ ТИПИ ДАНИХ
- ЛІНЙІНІ ПРОГРАМИ
- ПРОЦЕДУРНЕ ПРОГРАМУВАННЯ
- ПРОГРАМУВАННЯ РОЗГАЛУЖЕНЬ
- ОПЕРАТОРИ ПОВТОРЕННЯ З ПАРАМЕТРОМ
- МАСИВИ
- ІТЕРАЦІЙНІ ЦИКЛИ
- РЕКУРСІЯ
- Швидкі АЛГОРИТМИ СОРТУВАННЯ І ПОШУКУ
- СКЛАДНІ ТИПИ ДАНИХ: ЗАПИСИ І ФАЙЛИ
- МНОЖИНЫ
- ДИНАМІЧНІ СТРУКТУРИ ДАНИХ
- ОЛІМПІАДНІ ЗАДАЧІ

Задачі

- 1. Паліндром
- 2. Числа Фібоначчі
- 3. Кількість цифр
- 4. Сума цифр
- 5. Пошук найменшого елемента масиву
- 6. Факторіал
- 7. Ханойська башта
- 8. Рекурсивний пошук найбільшої цифри числа
- 9. Рекурсивний пошук найменшої цифри числа
- 10. Кількість непарних цифр
- 11. Біноміальний коефіцієнт
- 12. Обхід шахматної дошки конем
- 13. Функція Аккермана
- 14. Сума факторіалів
- 15. Кількість комбінацій
- 16. Найбільший спільний дільник
- 17. Знайти F(n)
- 18. Знайти F(n)

Бібліотека задач є гіпертекстом, який структурований за змістом. Зміст задачника представлений у лівій частині вікна Задачника. Щоб відкрити потрібний розділ Задачника, треба натиснути мишкою на його назві. У вікні відобразиться список задач обраного розділу. Щоб відкрити задачу в бібліотеці задач, треба натиснути на назві задачі.

Кожна задача у бібліотеці має наступну структуру:

- умова задачі;
- позначення вхідних та вихідних даних;
- пояснення вхідних та вихідних даних;
- приклад вхідних та вихідних даних.

ОАП: Бібліотека задач з основ алгоритмізації та програмування - Windows Internet Explorer

Файл Правка Вид Избранное Сервис Справка

ОАП: Бібліотека задач з основ алгоритмізації та пр...

СОРГУВАННЯ І ПОШУКУ

- СКЛАДНІ ТИПИ ДАНИХ: ЗАПИСИ
- ФАЙЛИ
- МНОЖИНЫ
- ДИНАМІЧНІ СТРУКТУРИ ДАНИХ
- ОЛІМПІАДНІ ЗАДАЧІ

• 13. Функція Аккермана ■

Умова

Обчислити значення функції Аккермана для двох невід'ємних цілих чисел n та m , де:

$$A(n, m) = \begin{cases} n + 1, & \text{якщо } n = 0 \\ A(n-1, 1) & \text{якщо } n \neq 0, m = 0 \\ A(n-1, A(n, m-1)) & \text{якщо } n > 0, m > 0 \end{cases}$$

Вхідні данні	Вихідні данні
$n\ m$	A

$$A(n, m) = \begin{cases} n + 1, & \text{якщо } n = 0 \\ A(n-1, 1) & \text{якщо } n \neq 0, m = 0 \\ A(n-1, A(n, m-1)) & \text{якщо } n > 0, m > 0 \end{cases}$$

Приклад:

1 3	5
-----	---

Розв'язати задачу

- 14. Сума факторіалів ■
- 15. Кількість комбінацій ■

10.1.1.133 Надежные узлы 100%

Після того, як користувач прочитає формулювання задачі та складе алгоритм розв'язку, він може відправити знайдений розв'язок для перевірки в систему алгоритмічних тестів.

ОАП: Бібліотека задач з основ алгоритмізації та програмування - Windows Internet Explorer

Файл Правка Вид Избранное Сервис Справка

ОАП: Бібліотека задач з основ алгоритмізації та пр...

Розв'язок задачі

Напишіть код у цьому ПОПІ

```
Program z13;
var c,m,n:longint;
function akker
  (n,m:longint):longint;
begin
  if n=0 then akker:=m+1
  else if (n>0) and (m=0)
    then akker:=akker(n-1,1)
  else if (m>0) and (n>0)
    then akker:=akker(n-1,Akk(n,m-1));
end;
```

або відправте файлом з кодом
(Максимальний розмір: 8Мб)

Оберіть мову Pascal

Зберегти зміни

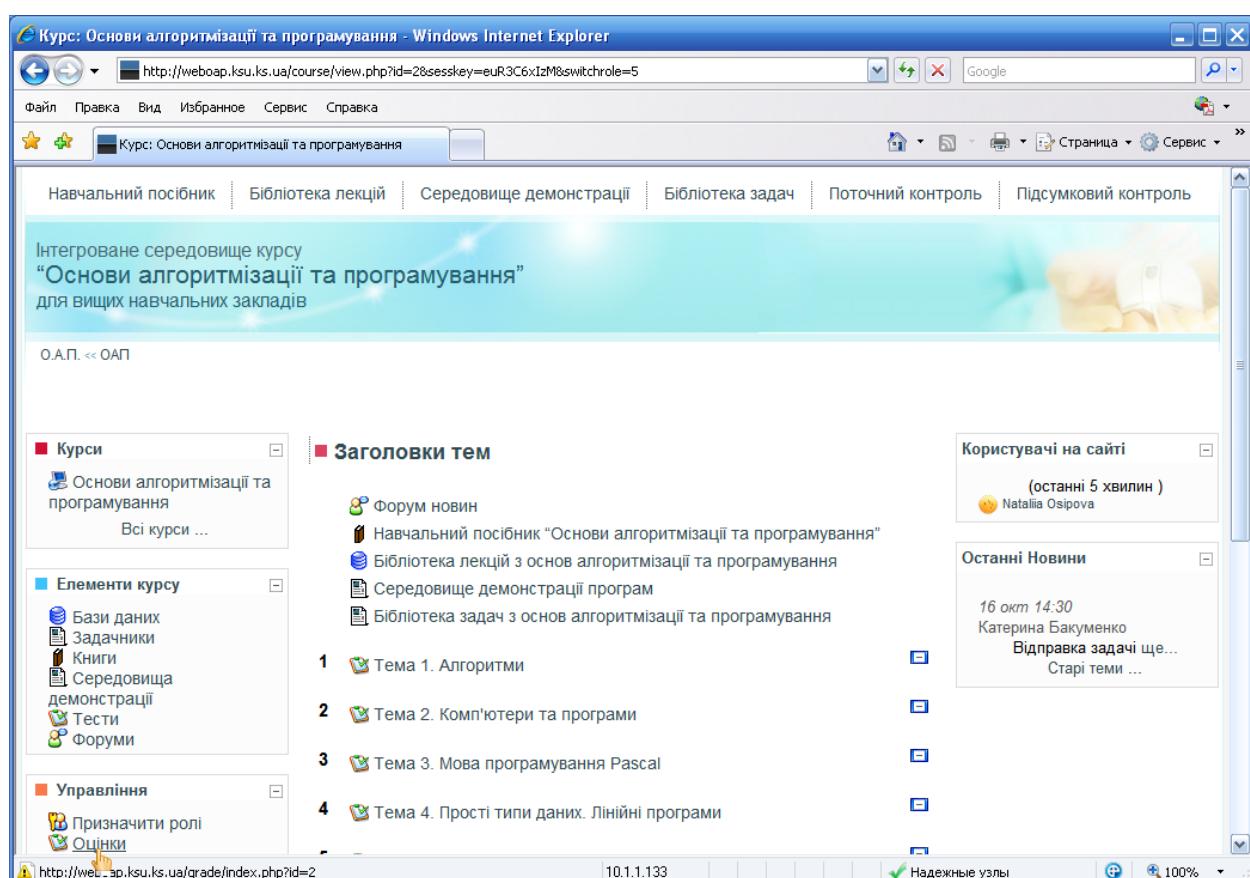
tasksubmit.php 10.1.1.133 Надежные узлы 100%

Для відправки розв'язку необхідно:

1. Натиснути на кнопку **Розв'язати задачу**, яка розташована під умовою відповідної задачі.
2. У вікні Розв'язок задачі обрати мову програмування Pascal або C.
3. Ввести (скопіювати) програмний код у поле середовища або завантажити файл з програмним кодом (максимальний розмір файла не повинен перевищувати 8 Мб).

4. Натиснути кнопку ***Зберегти зміни***.
 5. Переглянувши результати перевірки (кількість пройдених програмою тестів з загальної кількості тестів, що передбачені для даної задачі), користувач може підтвердити або відмінити відправку задачі.

За кожну з відправлених задач в журналі виставляється оцінка. Для перегляду своїх оцінок студент може на сторінці курсу у розділі Управління вибрати *Оцінки*.



7 МОДУЛЬ «СИСТЕМА ПОТОЧНОГО ТА ПІДСУМКОВОГО КОНТРОЛЮ ЗНАНЬ»

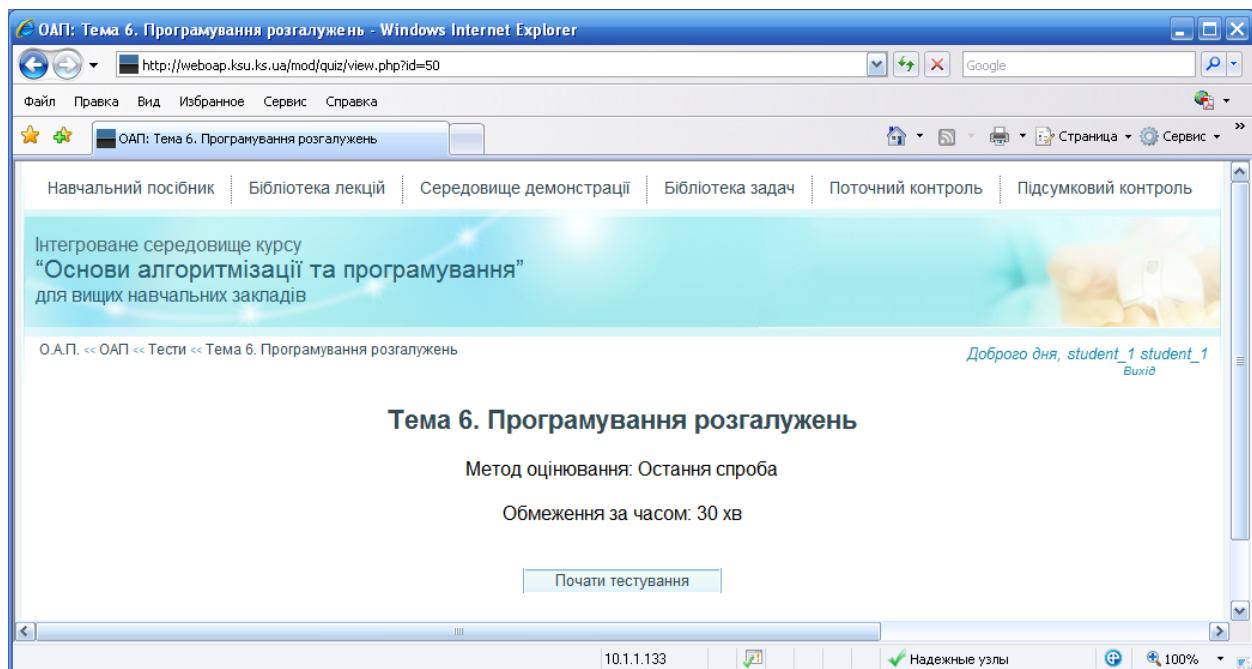
Модуль «Поточний контроль» має на меті перевірку рівня підготовленості студента до виконання лабораторної роботи. Тести згруповані за темами курсу ОАП.

The screenshot shows a Windows Internet Explorer window with the title 'ОАП: Тести - Windows Internet Explorer'. The URL in the address bar is <http://weboap.ksu.ks.ua/mod/quiz/index.php?id=2>. The menu bar includes 'Файл', 'Правка', 'Вид', 'Избранное', 'Сервис', and 'Справка'. The toolbar includes icons for back, forward, search, and print. The main content area has tabs at the top: 'Навчальний посібник', 'Бібліотека лекцій', 'Середовище демонстрації', 'Бібліотека задач', 'Поточний контроль', and 'Підсумковий контроль'. Below the tabs, a banner reads 'Інтегроване середовище курсу "Основи алгоритмізації та програмування" для вищих навчальних закладів'. A breadcrumb trail at the bottom left says 'О.А.П. << ОАП << Тести'. The main content is a table with 13 rows, each containing a number and a topic name. The columns are labeled 'Тема', 'Ім'я', 'Тест закривається', 'Кращий результат', and 'Оцінка'. The table data is as follows:

Тема	Ім'я	Тест закривається	Кращий результат	Оцінка
1	Тема 1. Алгоритми			
2	Тема 2. Комп'ютери та програми			
3	Тема 3. Мова програмування Pascal			
4	Тема 4. Прості типи даних. Лінійні програми			
5	Тема 5. Процедурне програмування			
6	Тема 6. Програмування розгалужень			
7	Тема 7. Оператори повторення з параметром. Масиви			
8	Тема 8. Ітераційні цикли			
9	Тема 9. Рекурсія			
10	Тема 10. Швидкі алгоритми сортування і пошуку			
11	Тема 11. Складні типи даних: записи і файли			
12	Тема 12. Множини			
13	Тема 13. Динамічні структури даних			

At the bottom of the browser window, there is an error message 'Ошибка на странице.' and a status bar showing '10.1.1.133' and 'Надежные узлы'.

Щоб вибрати тест, необхідно клацнути на його назві. Відкриється сторінка з описом тесту: тема, за якою проводиться тестування; метод оцінювання тесту (найвища, найнижча, середня оцінка, або оцінка за останню спробу); обмеження тесту за часом.



Натиснувши кнопку **Почати тестування**, студент починає спробу.

Кожне питання тесту може бути одного з наступних видів: обчислюваний, опис, есе, на відповідність, вкладені відповіді, у закритій формі (множинний вибір), коротка відповідь, числовий, випадкове питання на відповідність, альтернатива.

The screenshot shows a test page with the following details:

- Title:** Тема 6. Програмування розгалужень - Спроба 1
- Page Number:** Сторінка: (Попередній) 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 (Наступний)
- Question 1:** Чому буде дорівнювати $a+b$ після виконання оператору розгалуження?
Code:
a:=1; b:=1;
if a>0 then a:=a+1 else if b>0 then b:=b+1;
Select one answer:
 - a. 2
 - b. 1
 - c. 0
 - d. 3
- Question 24:** Які з цих стандартних визначених функцій приймають булеві значення?
Балів: 1
Select one answer:
 - a. Odd(X)
 - b. EoIn(F)
 - c. EoF(F)
 - d. Ord(x)
- Buttons at the bottom:** Зберегти, але не відправляти, Відправити все і завершити тест
- Page Number:** Сторінка: (Попередній) 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 (Наступний)

Тестування проходить у захищенному вікні. Обравши відповіді на питання на поточній сторінці, студент може натиснути кнопку **Зберегти, але не відправляти** або перейти на наступну сторінку з питаннями. Після відповіді на всі питання тесту студент повинен натиснути кнопку **Відправити все і завершити тест.**

ОАП: Тема 6. Програмування розгалужень - Microsoft Internet Explorer

Файл Правка Вид Избранное Сервис Справка

Назад Назад вперед Поиск Извещение Помощь Адрес: http://weboap.ksu.ks.ua/mod/quiz/view.php?id=50 Перехід

Навчальний посібник Бібліотека лекцій Середовище демонстрації Бібліотека задач Поточний контроль Підсумковий контроль

Інтегроване середовище курсу
“Основи алгоритмізації та програмування”
для вищих навчальних закладів

О.А.П. << ОАП << Тести << Тема 6. Програмування розгалужень

Доброого дня, student_2 student_2
Вихід

Тема 6. Програмування розгалужень

Метод оцінювання: Остання спроба

Обмеження за часом: 30 хв

Спроба	Завершено	Балів / 50	Оцінка
#1	четверг 25 Март 2010, 12:47	47.67	5 (A)

Остання спроба: 9.53 / 10.

Після проходження тесту студент може переглянути сторінку результатів, на якій відображено номер спроби, дата та час завершення тесту, кількість набраних балів та оцінку.

Для більш детального ознайомлення з результатами тесту необхідно натиснути на кількості балів. Відкриється сторінка, на якій відображено відповіді на кожне питання тесту:

- Правильні відповіді позначені зеленим кольором;
- Неправильні – сірим кольором;
- Частково правильні – жовтим кольором.

Вказано кількість балів, набраних за кожне питання тесту.

Оцінка за проходження тесту автоматично заноситься до журналу.

ПРЕДМЕТНИЙ ПОКАЖЧИК

Алгоритм.....	19, 22, 26, 27, 32, 34, 35, 37, 54, 55
Аналіз алгоритма	27, 62
Бінарний пошук.....	22
Ефективність....	4, 5, 15, 16, 22, 25, 26, 30, 34, 37, 40, 45, 49, 54, 64, 83
Лінійний пошук	15
Мажорованість.....	5, 42, 69
Обчислювальний експеримент	2, 5, 7
Пам'ять.....	5
Піраміdalне сортування	54
Пошук	15, 16, 17, 18, 19, 60, 80
Природність поведінки	5
Сортування.....	2, 5, 10, 11, 12, 25, 26, 27, 30, 32, 34, 37, 40, 41, 42, 45, 46, 49, 65
Сортування вибором	32
Сортування вставками	34
Сортування злиттям	45, 64
Сортування обмінами.....	25, 27, 30, 42, 43, 69
Сортування Хоара.....	49, 51, 68, 69
Сортування Шелла	37
Стійкість	5, 28, 42, 69
Час сортування	5

Науково-методичне видання

**Співаковський О.В.
Оsipova H.B.
Львов M.C.
Бакуменко K.B.**

**ОСНОВИ АЛГОРИТМІЗАЦІЇ
ТА ПРОГРАМУВАННЯ
ОБЧИСЛЮВАЛЬНИЙ ЕКСПЕРИМЕНТ**

РОЗВ'ЯЗАННЯ ПРОБЛЕМ ЕФЕКТИВНОСТІ
В АЛГОРИТМАХ ПОШУКУ ТА СОРТУВАННЯ

Навчальний посібник

ISBN 978-966-630-049-7

Підписано до друку 25.05.2011. Формат 60x84/16.
Папір офсетний. Друк цифровий. Гарнітура Times New Roman.
Умовн. друк. арк. 5,81. Наклад 300.

Видавець:
Видавництво «Айлант».
Свідоцтво ХС №1 від 20 серпня 2006 р.
73000, Україна, м. Херсон, пров. Пугачова, 5/20.
Тел.: (0552) 26-67-22.

Виготовник:
Видавництво ХДУ.
Свідоцтво серія ХС № 33 від 14.03.2003 р.
Видано Управлінням у справах преси та інформації облдержадміністрації.
73000, Україна, м. Херсон, вул. 40 років Жовтня, 4.
Тел. (0552) 32-67-95.

**Співаковський О.В.
Осипова Н.В.
Львов М.С.
Бакуменко К.В.**

**Співаковський
Олександр Володимирович**

Проректор з науково-педагогічної роботи, інформаційних технологій, міжнародних зв'язків, завідувач кафедри інформатики, кандидат фізико-математичних наук, доктор педагогічних наук, професор, заслужений працівник освіти України



**Львов
Михайло Сергійович**

кандидат фізико-математичних наук (математична кібернетика), професор кафедри інформатики, директор Центру інформаційно-комунікаційних технологій Херсонського державного університету.

**Осипова
Наталія Володимирівна**

кандидат технічних наук, доцент кафедри інформатики, керівник відділу інтегрованих середовищ навчання Херсонського державного університету.



**Бакуменко
Катерина Вікторівна**

співробітник компанії Postindustria IT.

Основи алгоритмізації та програмування

Обчислювальний експеримент

Розв'язання проблем ефективності в алгоритмах пошуку та сортування

У посібнику запропоновано модель вивчення основ алгоритмізації та програмування з використанням інтегрованого середовища, де на відміну від традиційного підходу головна увага приділяється задачі аналізу на всіх стадіях процесу проектування та реалізації алгоритмів. Описано новий підхід до вивчення поняття складності та вивчення властивостей алгоритмів і вибору оптимального алгоритма, що має величезне значення для порівняння різних алгоритмів і використання їх для розв'язування практичних задач. Головна увага приділяється проведенню обчислювального експерименту для вивчення складності та мажорованості алгоритмів сортування з використанням інтегрованого середовища WEBOAP (weboap.ksu.ks.ua), розробленого у НДІ ІТ Херсонського державного університету.

Посібник призначений для викладачів, аспірантів, студентів вищих навчальних закладів, вчителів з інформатики та учнів старших класів загальноосвітніх навчальних закладів.