**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**імені ІГОРЯ СІКОРСЬКОГО”**

**ЗВІТ**

**з лабораторної роботи №3**

**з навчальної дисципліни “****Рекурсивні алгоритми”**

**Тема:**

**Методи сортування.**

**Виконав студент групи ТР–12**

Каркушевський Владислав

Лабораторну роботу захищено

з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Перевірив доцент кафедри**

\_\_\_\_\_\_\_\_\_\_\_ Андрій ОНИСЬКО

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 року

**Київ 2022**

**Мета:** Метою лабораторної роботи є набуття практичних навичок з

використання простих методів сортування.

**Теоретична частина**

Сортування – процес переставлення об’єктів множини у визначеному порядку з метою полегшення наступного пошуку елементів у відсортованій множині. Елементи сортування є практично в усіх задачах (телефонні книги, відомості, звіти). Найчастіше сортування використовують для подальшого пошуку даних. Методи сортування поділяють на такі категорії: сортування масивів та сортування файлів.

Практично кожен алгоритм сортування можна розбити на три частини:

- порівняння, що визначає впорядкованість пари елементів;

- перестановку, що змінює місцями пару елементів;

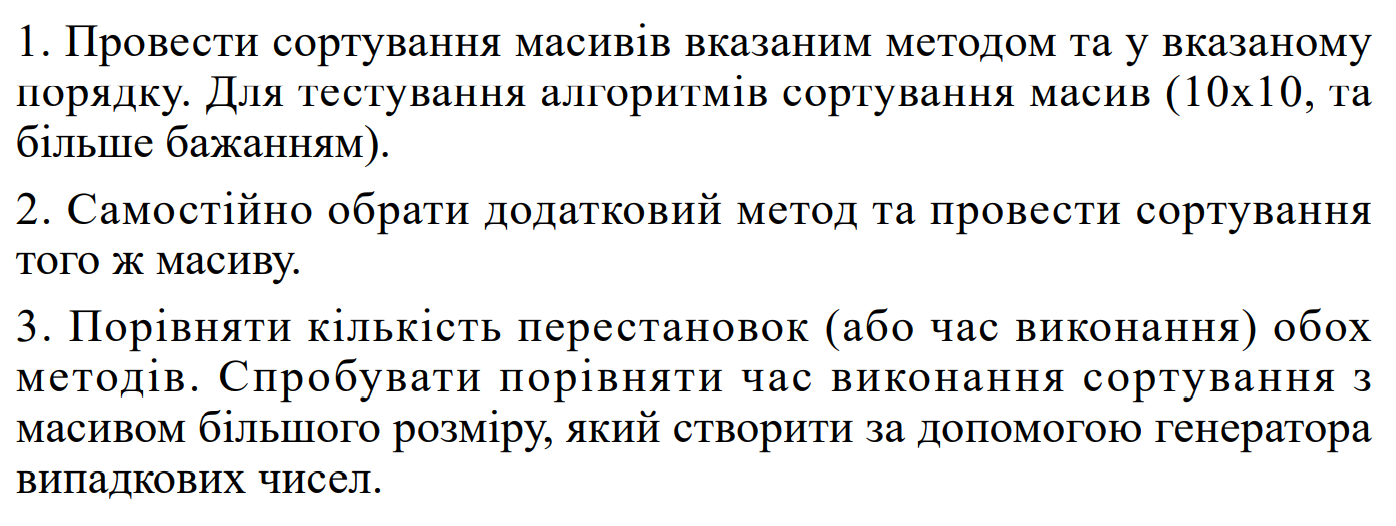
- власне сортуючий алгоритм, що здійснює порівняння і перестановку елементів доти, поки всі елементи множини не будуть впорядковані

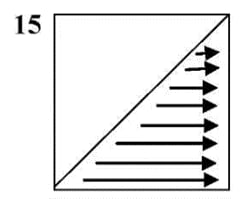
Методи сортування які будуть використовуватися у лабораторній:

* сортування вставкою (insertoin sort)
* сортування бульбашкою(bubble sort)
* сортування гребінцем(combSort)
* сортування вибором(selection sort)
* швидке сортування(quickSort)
* Сортування методом Шелла(shellSort)

**Завдання**

**Варіант 15**

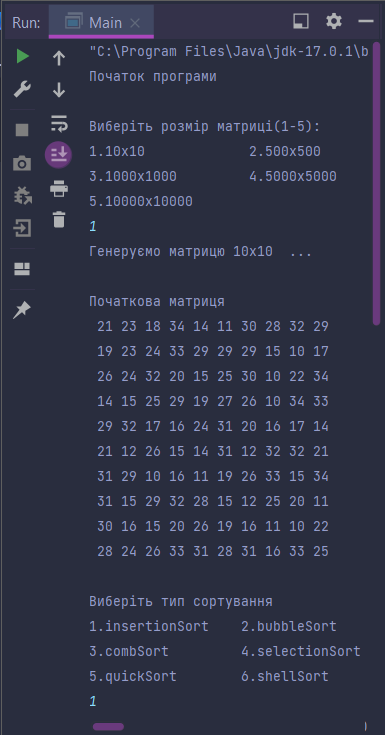
****

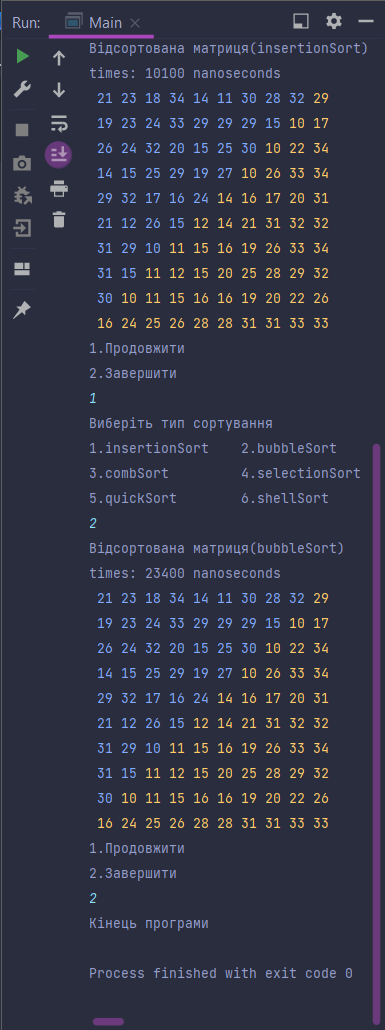


**Код програми:**

*import* java.util.Scanner;*//імпортуємо сканер  
  
public class* Main {  
 *public static void* main(String[] args) {  
  
 System.***out***.println("Початок програми\n");  
  
  
 System.***out***.println("Виберіть розмір матриці(1-5):\n1.10x10 2.500x500\n3.1000x1000 4.5000x5000\n5.10000х10000");  
  
 Scanner sc = *new* Scanner(System.***in***);*//створюємо сканер  
 int* size = sc.nextInt();*//за допомогою сканера вибираємо розмір матриці* size = *info*(size);  
  
 *int*[][] matrixStart = *makeMatrix*(size);*//викликаємо метод для створення матриці* System.***out***.println("Початкова матриця");  
 *outputMatrix*(matrixStart);*//виводимо початкову матрицю  
  
 boolean* choise = *true* ;  
 *while*(choise) {*//створюємо міні-програму* System.***out***.println("Виберіть тип сортування\n1.insertionSort 2.bubbleSort\n3.combSort 4.selectionSort\n5.quickSort 6.shellSort");  
 *int* par = sc.nextInt();*//викликаємо сканер  
  
 if*(par == 1 ){  
 System.***out***.println("Відсортована матриця(insertionSort)");  
 *int* [][] matrixForInsertionSort = *task*(matrixStart, par);*//викликаємо метод task з вибраним типом мортування  
 print*(matrixForInsertionSort);  
  
 }*else if*(par == 2 ){  
 System.***out***.println("Відсортована матриця(bubbleSort)");  
  
 *int* [][] matrixForBubbleSort = *task*(matrixStart, par);*//викликаємо метод task з вибраним типом мортування  
 print*(matrixForBubbleSort);*//виводимо матрицю різнокольоровим коліром* }*else if*(par == 3 ){  
 System.***out***.println("Відсортована матриця(combSort)");  
  
 *int* [][] matrixForCombSort = *task*(matrixStart, par);*//викликаємо метод task з вибраним типом мортування  
 print*(matrixForCombSort);*//виводимо матрицю різнокольоровим коліром* } *else if*(par == 4 ){  
 System.***out***.println("Відсортована матриця(selectionSort)");  
  
 *int* [][] matrixForSelectionSort = *task*(matrixStart, par);*//викликаємо метод task з вибраним типом мортування  
 print*(matrixForSelectionSort);*//виводимо матрицю різнокольоровим коліром* }*else if*(par == 5 ){  
 System.***out***.println("Відсортована матриця(quickSort)");  
  
 *int* [][] matrixForQuickSort = *task*(matrixStart, par);*//викликаємо метод task з вибраним типом мортування  
 print*(matrixForQuickSort);*//виводимо матрицю різнокольоровим коліром* }*else if*(par == 6 ){  
 System.***out***.println("Відсортована матриця(shellSort)");  
  
 *int* [][] matrixForShellSort = *task*(matrixStart, par);*//викликаємо метод task з вибраним типом мортування  
 print*(matrixForShellSort);*//виводимо матрицю різнокольоровим коліром* }*else*{  
 System.***out***.print("\u001B[31m");  
 System.***out***.println("error");  
 System.*exit*(0);  
 }  
  
  
 System.***out***.println("1.Продовжити\n2.Завершити");  
 *int* end = sc.nextInt();  
 *if*(end == 2 ){  
 choise = *false*;  
 }*else*{  
 System.***out***.print("\u001B[31m");  
 System.***out***.println("error");  
 System.*exit*(0);  
 }  
 }  
  
 System.***out***.println("Кінець програми");  
  
  
 }  
  
  
 *public static int*[][] task(*int*[][] matrix, *int* parametr){*//створюємо метод завдання згідно варіанту  
  
  
 int*[][] finalMatrix = *new int*[matrix.length][matrix.length];*//створюємо нову матрицю  
 for*(*int* i = 0 ; i < matrix.length;i++){  
 *for*(*int* j = 0 ; j < matrix[0].length;j++){  
 finalMatrix[i][j] = matrix[i][j];*//заповнюємо тими самими значеннями що і початокву матрицю* }  
 }  
 *long* start = System.*nanoTime*();*//розпочинаємо вимір часу  
  
 int* index\_A = 2;  
 *int* index\_B = 0 ;  
 *for*(*int* i = 1; i< finalMatrix.length ; i++){  
 *int*[] res = *new int*[index\_A];*//створємо одновимірний масив  
 for*(*int* j = finalMatrix.length- index\_A; j< finalMatrix[0].length; j++ ){  
 res[index\_B] = finalMatrix[i][j];*//заповняємо значеннями відповідного рядка* index\_B++;  
 }  
 index\_B =0;  
  
  
 *if*(parametr == 1 ){*//сортуємо за вибраним методом  
 insertionSort*(res);  
 }*else if*(parametr == 2 ){  
 *bubbleSort*(res);  
 }*else if*(parametr == 3 ){  
 *combSort*(res);  
 }*else if*(parametr == 4 ){  
 *selectionSort*(res);  
 }*else if*(parametr == 5 ){  
 *quickSort*(res,0, res.length-1);  
 }*else if*(parametr == 6 ){  
 *shellSort*(res);  
 }  
 *for*(*int* k = finalMatrix.length- index\_A; k< finalMatrix.length; k++){  
 finalMatrix[i][k] = res[index\_B];*//повертаємо його у матрицю* index\_B++;  
 }  
 index\_B =0;  
 index\_A++;  
 }  
  
 *long* finish = System.*nanoTime*();*//закінчуємо вимір часу* System.***out***.printf( "times: %d nanoseconds\n", finish-start);*//виведення часу перестановки матриці  
 return* finalMatrix;  
 }  
  
  
 *public static void* swap(*int*[] nums, *int* x, *int* y) {*//створюємо метод для зміни елементів масиву місцями  
 int* temp = nums[x];  
 nums[x] = nums[y];  
 nums[y] = temp;  
 }  
  
  
 *public static void* insertionSort(*int*[]nums){*//створюємо метод для сортування вставками  
 int* j ;  
 *for*(*int* i = 1;i<nums.length;i++){  
 *int* x = nums[i];  
 j=i;  
 *while* (j>0 && nums[j-1]>x){  
 *swap*(nums,j,j-1);  
 j--;  
 }  
 nums[j]=x;  
 }  
 }  
  
  
 *public static void* bubbleSort(*int*[]nums){*//створюємо метод для сортування бульбашкою  
 boolean* bool = *true*;  
 *while*(bool){  
 bool = *false*;  
 *for*(*int* i = 0 ; i < nums.length-1 ; i++){  
 *if*(nums[i]>nums[i+1]){  
 *swap*(nums,i,i+1);  
 bool = *true*;  
 }  
 }  
 }  
 }  
  
  
 *public static void* combSort(*int*[] nums){*//створюємо метод для сортування гребінцем  
 int* gap\_length = nums.length;  
 *float* shrinkFactor = 1.3f;  
 *boolean* swapped = *false*;  
  
 *while* (gap\_length > 1 || swapped) {  
 *if* (gap\_length > 1) {  
 gap\_length = (*int*)(gap\_length / shrinkFactor);  
 }  
  
 swapped = *false*;  
  
 *for* (*int* i = 0; gap\_length + i < nums.length; i++) {  
 *if* (nums[i] > nums[i + gap\_length]) {  
 *swap*(nums, i, i + gap\_length);  
 swapped = *true*;  
 }  
 }  
 }  
  
 }  
  
  
 *public static void* selectionSort(*int*[] nums){*//створюємо метод для сортування вибором  
 for* (*int* i = 0; i < nums.length - 1; i++)  
 {  
 *int* index = i;  
 *for* (*int* j = i + 1; j < nums.length; j++){  
 *if* (nums[j] < nums[index]){  
 index = j;  
 }  
 }  
 *swap*(nums,index,i);  
 }  
  
 }  
  
  
 *public static void* quickSort(*int*[] nums, *int* low, *int* high) {*//створюємо метод для швидкого сортування  
  
 // вибираємо опорний елемент  
 int* middle = low + (high - low) / 2;  
 *int* opora = nums[middle];  
  
 *// розділимо на підмасиви, які більші і менші опорного елемента  
 int* i = low, j = high;  
 *while* (i <= j) {  
 *while* (nums[i] < opora) {  
 i++;  
 }  
  
 *while* (nums[j] > opora) {  
 j--;  
 }  
  
 *if* (i <= j) {*//міняємо місцями  
 swap*(nums,i,j);  
 i++;  
 j--;  
 }  
 }  
 *// викликаємо рекурсію для сортування лівої і правої частини  
 if* (low < j) {  
 *quickSort*(nums, low, j);  
 }  
 *if* (high > i) {  
 *quickSort*(nums, i, high);  
 }  
 }  
  
 *public static void* shellSort(*int* [] nums){*//створюємо метод для сортування методом Шелла  
  
 int* h = 1;  
 *int* n = nums.length;  
 *while* (h < n / 3) {  
 h = 3 \* h + 1;  
 }  
 *while* (h >= 1) {  
 *for* (*int* i = h; i < nums.length; i++) {  
 *for* (*int* j = i; j >= h && nums[j - h] > nums[j]; j -= h) {  
 *int* temp = nums[j];  
 nums[j] = nums[j - h];  
 nums[j - h] = temp;  
 }  
 }  
 h = h / 3;  
 }  
 }  
  
  
 *public static int* info(*int* size ){*//створюємо метод для вибору розміру матриці  
  
 if*(size == 1){  
 size = 10;  
 System.***out***.println("Генеруємо матрицю 10x10 ...\n");  
 }*else if*(size == 2 ){  
 size = 500;  
 System.***out***.println("\nГенеруємо матрицю 500x500 ...\n");  
 }*else if*(size == 3 ){  
 size = 1000;  
 System.***out***.println("\nГенеруємо матрицю 1000x1000 ...\n");  
 }*else if*(size == 4 ){  
 size = 5000;  
 System.***out***.println("\nГенеруємо матрицюg 5000x5000 ...\n");  
 }*else if*(size == 5 ){  
 size = 10000;  
 System.***out***.println("\nГенеруємо матрицю 10000x10000 ...\n");  
 }*else*{  
 System.***out***.print("\u001B[31m");  
 System.***out***.println("error");  
 System. *exit*(0);  
 }  
  
 *return* size;  
 }  
  
  
 *public static int*[][] makeMatrix( *int* size ){*//створюємо метод для створення матриці  
  
 int*[][] a = *new int* [size][size];  
  
 *for*(*int* i = 0 ; i< a.length; i++){  
 *for*(*int* j = 0 ; j< a[0].length;j++){  
 a[i][j] = (*int*) (Math.*random*()\*25+10);*//заповнюємо матрицю числами в діапазоні [10;35]* }  
 }  
 *return* a;  
 }  
  
  
 *public static void* outputMatrix(*int*[][] b){*//створюємо метод для виведення матриці  
  
 for*(*int* i = 0 ; i< b.length;i++){  
 *for*(*int* j = 0 ; j< b[0].length;j++ ){  
 System.***out***.printf("%3d", b[i][j]);  
 }  
 System.***out***.println();  
 }  
 System.***out***.println();  
 }  
  
  
 *public static void* print(*int* [][] matrix){*//створюємо метод для різнобарвного виведення матриці  
 int* size = matrix[0].length-1;  
  
 *for* (*int* i = 0 ; i < matrix.length; i++) {  
 *for* (*int* j = 0; j < size; j++) {  
 System.***out***.print("\u001B[34m");  
 System.***out***.printf("%3d",matrix[i][j]);  
 }  
 *for*(*int* k = size; k <matrix[0].length;k++ ) {  
 System.***out***.print("\u001B[33m");  
 System.***out***.printf("%3d", matrix[i][k]);  
 }  
 System.***out***.println();  
 size--;  
 }  
  
 System.***out***.print("\u001B[0m");  
 }  
  
  
}

**Результат виконання програми:**

****

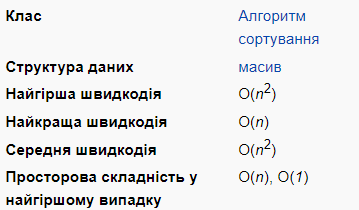
****

**Складність алгоритму**

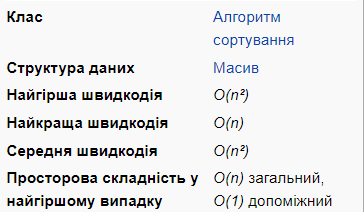
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **10** | **500** | **1 000** | **5 000** | **10 000** |
| **insertionSort** | **9 800 ns** | **27 223 600 ns** | **58 087 500 ns** | **5 004 400 600 ns** | **45 920 222 500 ns** |
| **bubbleSort** | **24 400 ns** | **83 511 200 ns** | **317 105 300 ns** | **38 051 814 100 ns** | **376 880 210 200 ns** |
| **combSort** | **11 700 ns** | **47 434 100 ns** | **61 661 200 ns** | **363 622 800 ns** | **1 669 991 600 ns** |
| **selectionSort** | **18 700 ns** | **18 232 600 ns** | **169 200 300 ns** | **10 221 302 000 ns** | **102 949 534 000 ns** |
| **quickSort** | **17 800 ns** | **7 638 500 ns** | **23 304 400 ns** | **436 820 600 ns** | **2 194 576 600 ns** |
| **shellSort** | **12 400 ns** | **8 689 700 ns** | **30 932 500 ns** | **508 780 900 ns** | **2 056 523 000 ns** |

Як можемо побачити по таблиці часу виконання, що при малих розмірах матриці різниця незначна, але коли розмір великий то час таких методів як : **combSort, quickSort, shellSort** значно менший ніж **selectionSort, insertionSort**. Самим довшим є метод **bubbleSort**.

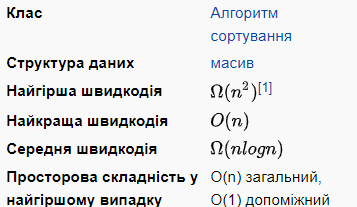
**insertionSort**



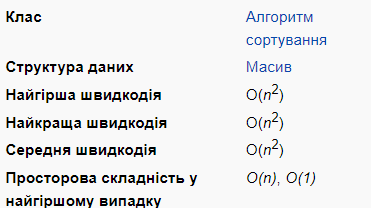
**bubbleSort**



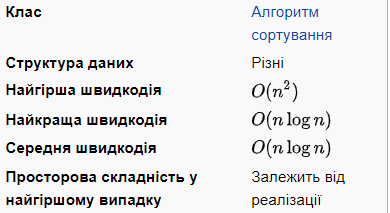
**combSort**



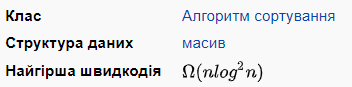
**selectionSort**



**quickSort**



**shellSort**



**Висновок:**

На цій лабораторній роботи були набутті практичні навички з використання простих методів сортування, таких як: insertionSort, bubbleSort, combSort, selectionSort, quickSort, shellSort. Було оцінену часову складність.