Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут атомної і теплової енергетики

Кафедра цифрових технологій в енергетиці

**Проектування та аналіз обчислювальних алгоритмів**

ЗВІТ ДО

КОМП’ЮТЕРНОГО ПРАКТИКУМУ № 1

«Методи сортування масивів»

*(Тема роботи)*

Варіант № 16

Виконав: студент 2-го курсу

гр. ТР-12

Каркушевський В.Л.

*(П.І.Б.)*

Оцінка \_\_\_\_\_\_\_\_\_ Перевірив: Крячок О.С.

Дата « 30 » вересня 2022 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(П.І.Б., підпис)*

Київ – 2022

**Завдання**

Ознайомитись з алгоритмами сортування масивів та способами їхньої

реалізації. У якості індивідуального завдання необхідно написати програмний код, у якому реалізується сортування масивів методами бульбашки, вставок, вибору, сортуванням Шелла, Гоара та швидкого сортування. Виконати порівняння ефективності вказаних методів сортування.

Звернення до елементів масиву реалізувати за допомогою вказівника на масив (С++).

Вихідні дані по варіантах у Додатку В-1.

РЕЗУЛЬТАТ РОБОТИ ПОТРІБНО:

1. Роздрукувати (вивести на екран) початковий масив та масиви після

виконання сортування різними методами. Показати кількість операцій

для виконання сортування різними методами.

2. Відкритий для редагування програмний код розмістити на сайті

https://replit.com/ (посилання через кнопку «+ Invite»).

3. Звіт до комп’ютерного практикуму № 1 додати в свій Клас на ресурсі <https://classroom.google.com/>.

Індивідуальне завдання

16. Протабулювати функцію y = cos (x^2 + 2) на відрізку [-π, π] з кроком h = π / 10 і впорядкувати значення функції за зростанням.

Після виконання лабораторної роботи, очікується визначити ефективнісь кожного алгоритму, та його переваги і недоілки.

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

**Сортування бульбашкою (bubble sort)**

Сутність методу полягає в багаторазовому проході по масиву. На кожному кроці послідовно порівнюються пари сусідніх елементів, і якщо порядок в такій парі невірний, то елементи в парі міняються місцями. При проході алгоритму, елемент, що стоїть не на своїй позиції, «спливає» до потрібної позиції як бульбашка, звідки і назва алгоритму.

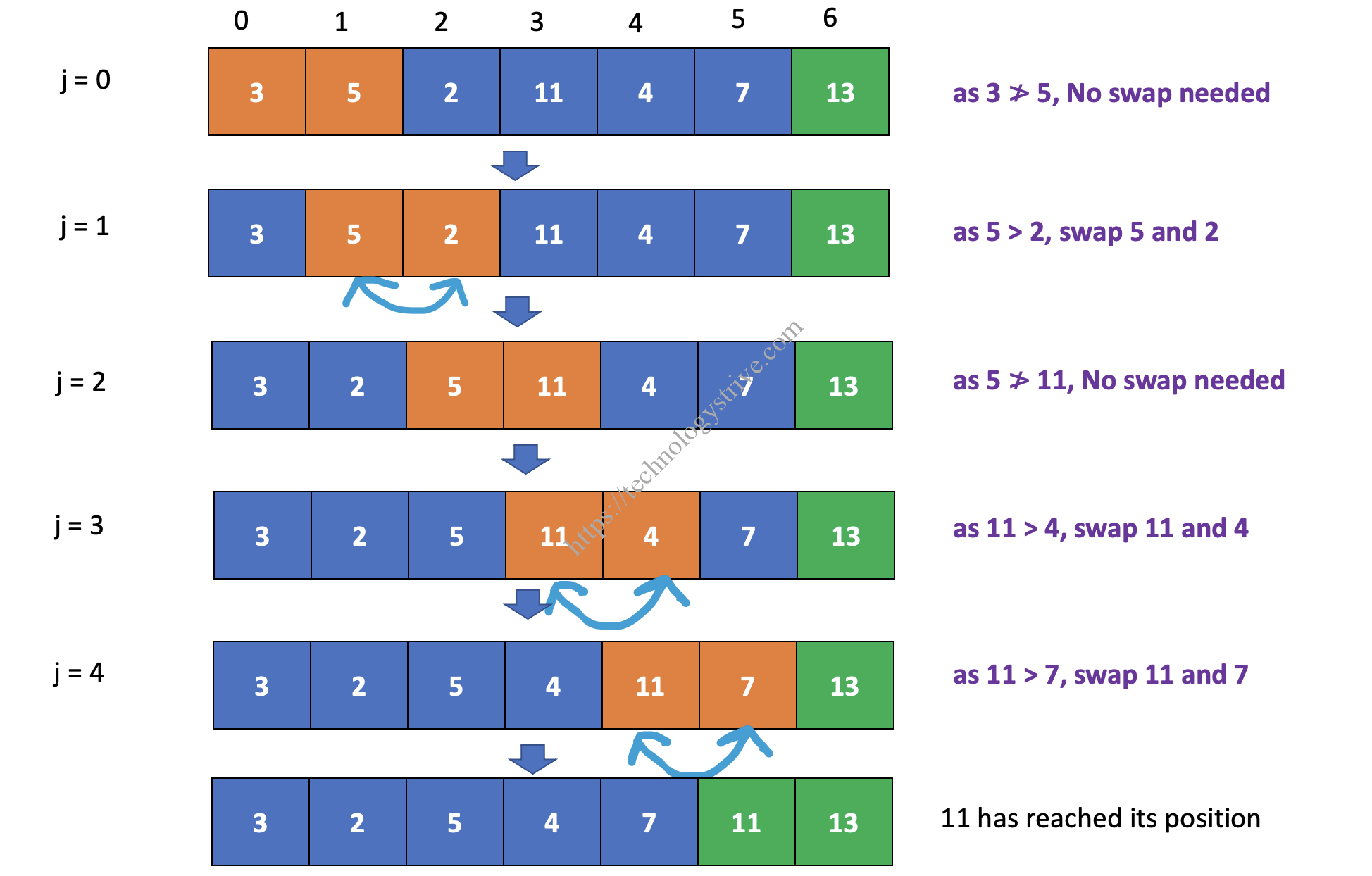


Рис. Принцип дії алгоритму

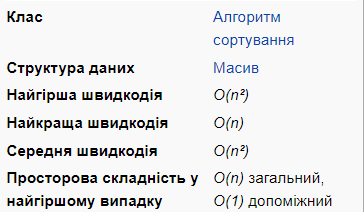


Рис. Часова складність

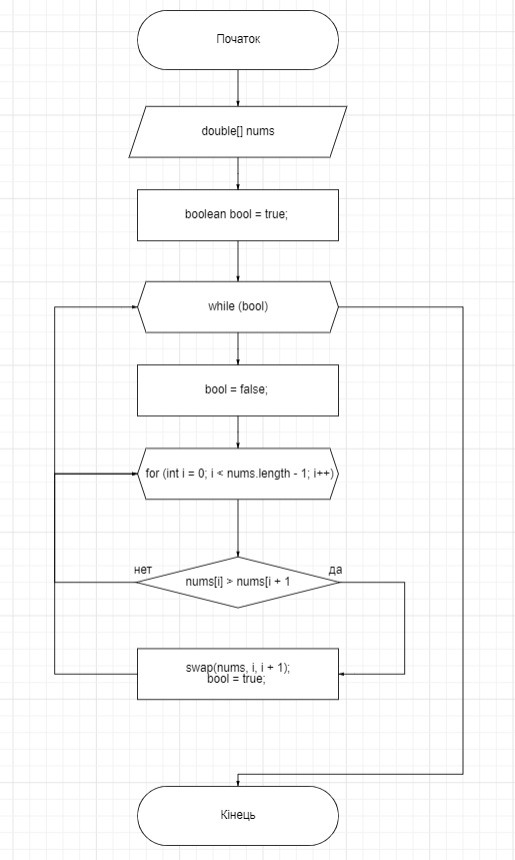
****

Рис. Блок схема до алгоритму сортування бульбашкою

**Cортування** **вставкою (insertoin sort)**

Сортований масив переглядається в порядку зростання номерів і кожен елемент вставляється в уже переглянуту частина масиву так, щоб зберегти порядок.



Рис. Принцип дії алгоритму

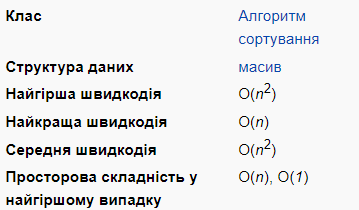


Рис. Часова складність

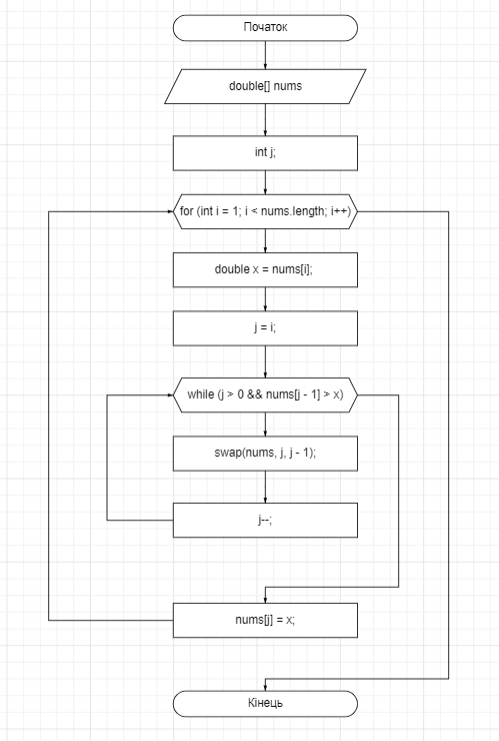
****

Рис. Блок схема до алгоритму сортування вставкою

**Сортування вибором(selection sort)**

Спочатку відшукується найменший елемент масиву, потім він міняється місцями з елементом, що стоїть першим у сортованому масиві. Далі, знаходиться другий найменший елемент і міняється місцями з елементом, що стоїть другим у вихідному масиві. Цей процес триває до тих пір, поки весь масив не буде відсортований.

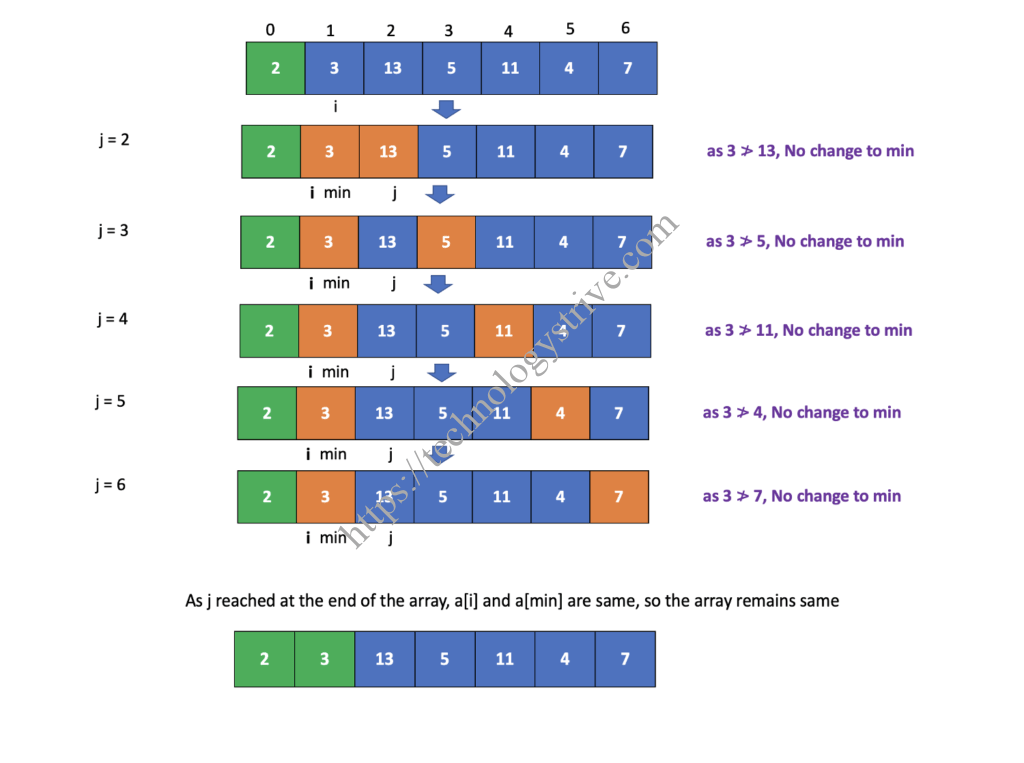


Рис. Принцип дії алгоритму

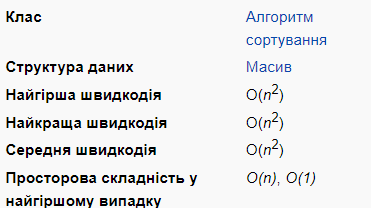


Рис. Часова складність

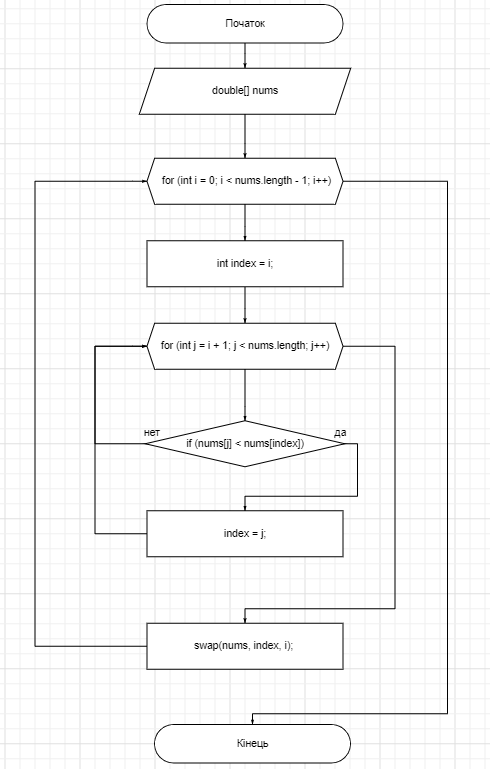
****

Рис. Блок схема до алгоритму сортування вибором

**Сортування** **методом Шелла(shellSort)**

Цей метод полягає в порівнянні елементів масиву, розділених однаковою відстанню таким чином, щоб елементи на цій відстані були впорядковані. Потім ця відстань ділиться навпіл і процес триває. В кінці відстань рівна 1 і якщо змін немає, то масив відсортований.

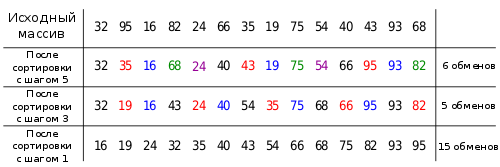


Рис. Принцип дії алгоритму

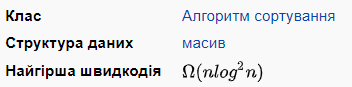


Рис. Часова складність

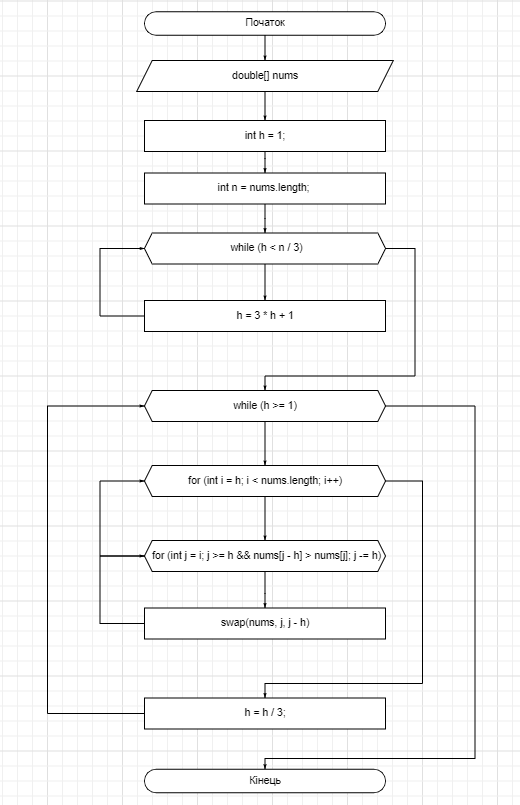
****

Рис. Блок схема до алгоритму сортування методом Шелла

**Швидке сортування(quickSort)**

Цей метод розглядає масив, як список значень. Спочатку виділяється середнє значення як сепаратор (фактор розбиття) списку. Список розбивається на два: в одному з них значення менше сепаратора, а в іншому - більше або рівні. Далі процедура сортування рекурсивно викликає саму себе для кожного з двох списків. Кожен раз при виклику сортування список елементів розбивається на два менших.



Рис. Принцип дії алгоритму

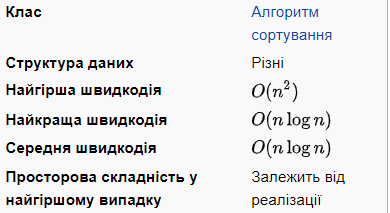
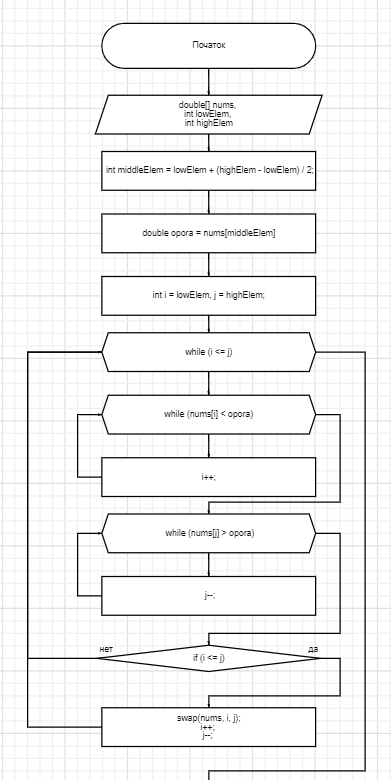


Рис. Часова складність

****

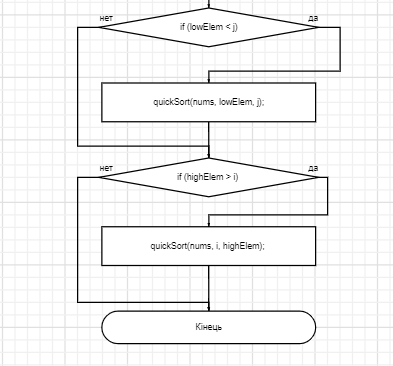
****

Рис. Блок схема до алгоритму сортування методом Шелла

**Швидке сортування Гоара(quickGoaraSort)**

Швидке сортування Гоара. Цей метод ґрунтується на послідовному поділі набору даних на блоки меншого розміру таким чином, що між значеннями різних блоків забезпечується відношення впорядкованості (для будь-якої пари блоків всі значення одного з цих блоків не перевищують значень іншого блоку).

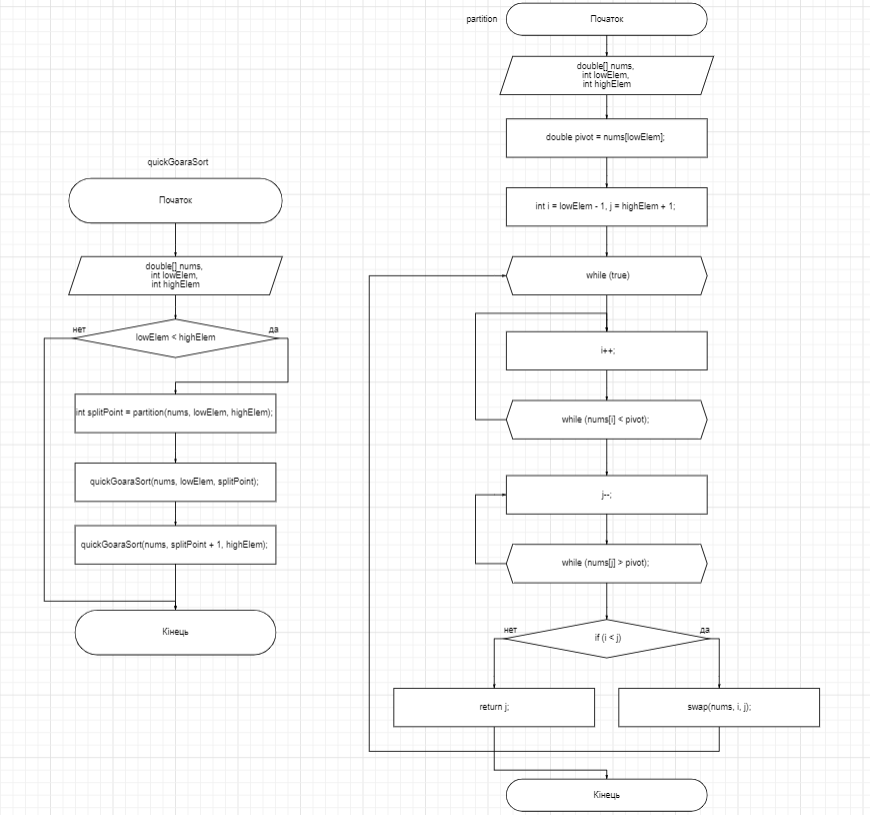
****

Рис. Блок схема до алгоритму швидке сортування Гоара

**Сортування гребінцем(combSort)**

В сортуванні гребінцем порівнюються елементи, які відстоять один від одного на якійсь відстані. Воно не надто велике, щоб сильно не відкидати елементи та повертати потім більшість назад, але й не надто маленьке, щоб можна було відправляти не на сусіднє місце, а трохи далі. На першому етапі ми знаходимо довжину масиву і ділимо її на 1,3. Тепер ми проходимо перший цикл сортування, порівнюючи 1 результатом ділення, 2 з результатом ділення плюс один і тд. Це відправить найбільші числа, якщо вони є на початку, насамкінець. Тепер ми знову ділимо минулий результат ділення на 1,3 і проходимо весь масив і порівнюємо. Так ми зменшимо кількість перестановок, а з ними заощадимо і процесорний час, необхідний для їх обробки.

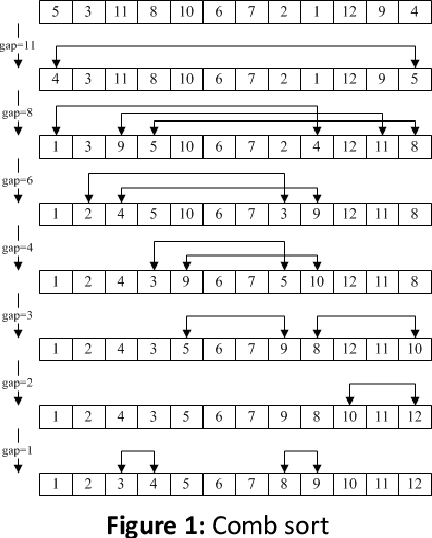


Рис. Принцип дії алгоритму

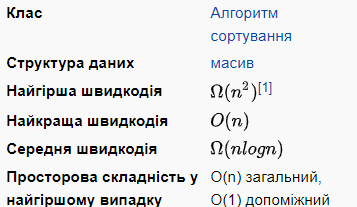


Рис. Часова складність

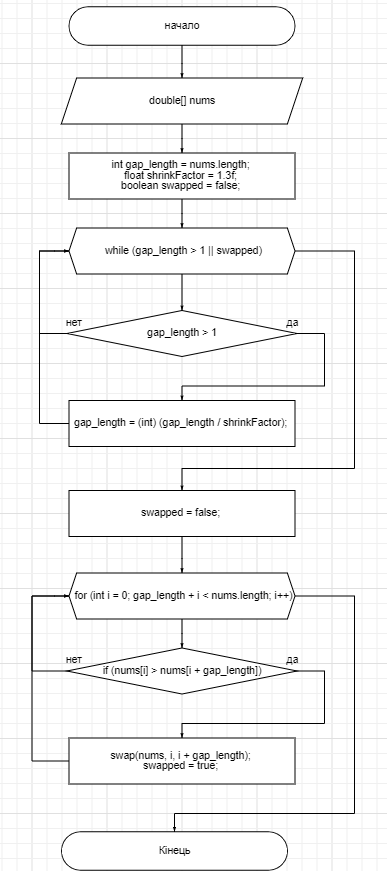
****

Рис. Блок схема до алгоритму сортування гребінцем

**Описання програми**

У лабораторній роботі розроблено сім методів різного сортування масивів, такі як: сортування бульбашкою(bubbleSort), сортування вставкою (insertoin sort), сортування вибором(selection sort), сортування методом Шелла(shellSort), швидке сортування(quickSort), швидке сортування Гоара(quickGoaraSort) та сортування гребінцем(combSort).

Також розроблено методи:

swap() – який змінює елементи масива місцями.

print\_array\_num() – який виводить елементи масива.

all\_print() – який виводить усю інформацію, таку як початковий масив, відсортований масив, час роботи алгоритму та кількість ітерацій.

У методі main, створено масив з довжиною в 21 елемент і заповнено його значеннями протабульованими функцією y = cos (x:2 + 2) на відрізку [-π, π] з кроком h = π / 10.

Після заповнення масива елементами , його виведено у консоль, за допомогою команди print\_array\_num(). Далі створено цикл while, у якому ми вибираємо тип сортування , ввівши відповідне число у консоль, після вибору виводиться його загальний результат, тобто результат дії метода all\_print(). Після чого потрібно вибрати, чи продовжувати сортування, чи закінчити. У разі введеня некоректного значення, ми виводимо повідомлення у консоль ‘error’ та виходимо із програми. У разі продовження, знову надається вибір типу сортування. У разі закінчення ми виходиму з циклу.

**Результати роботи**

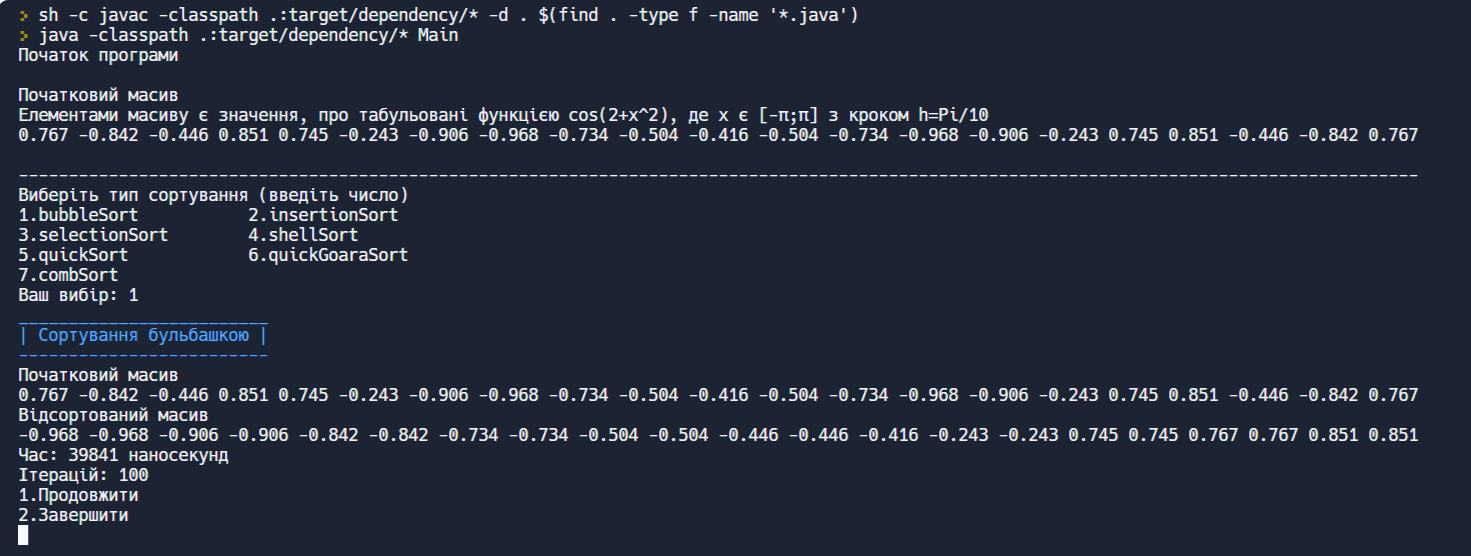


Рис. Виконання сортування бульбашкою

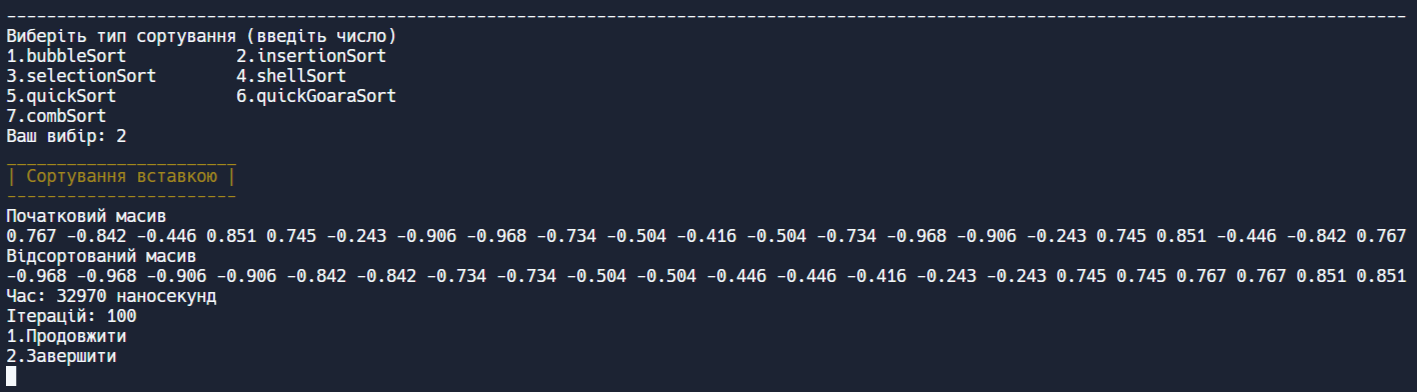


Рис. Виконання сортування вставкою

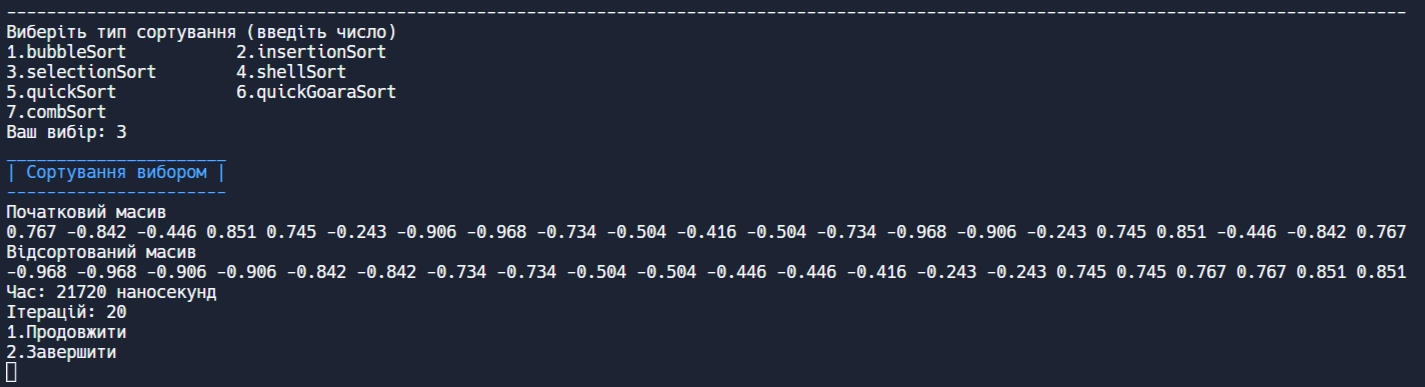


Рис. Виконання сортування вибором

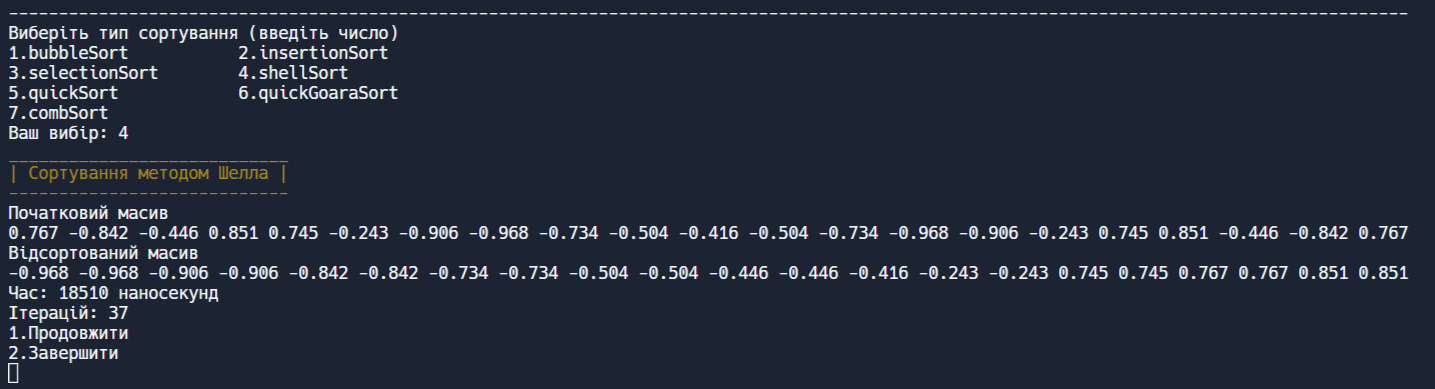


Рис. Виконання сортування вибором

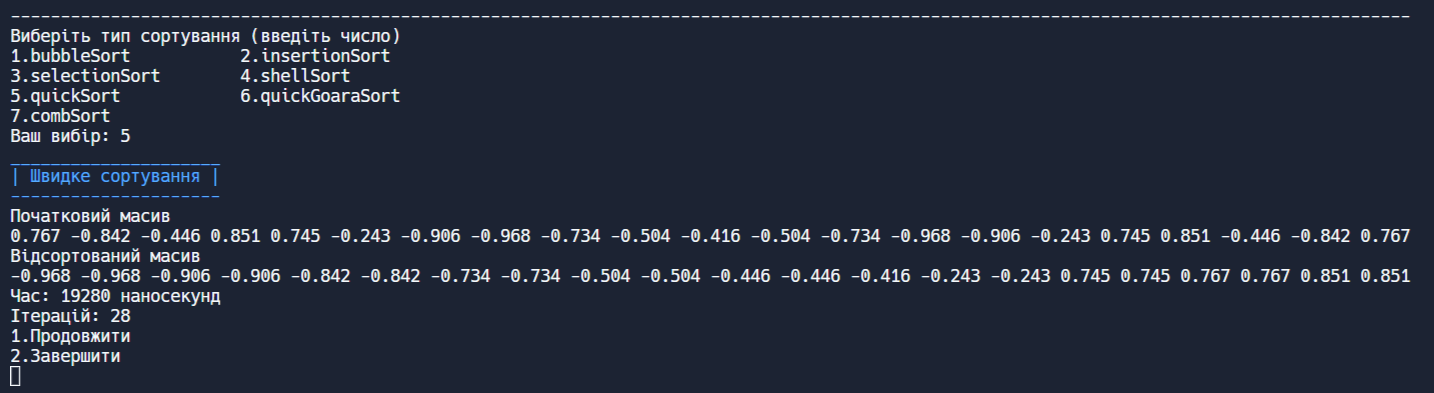


Рис. Виконання швидкого сортування

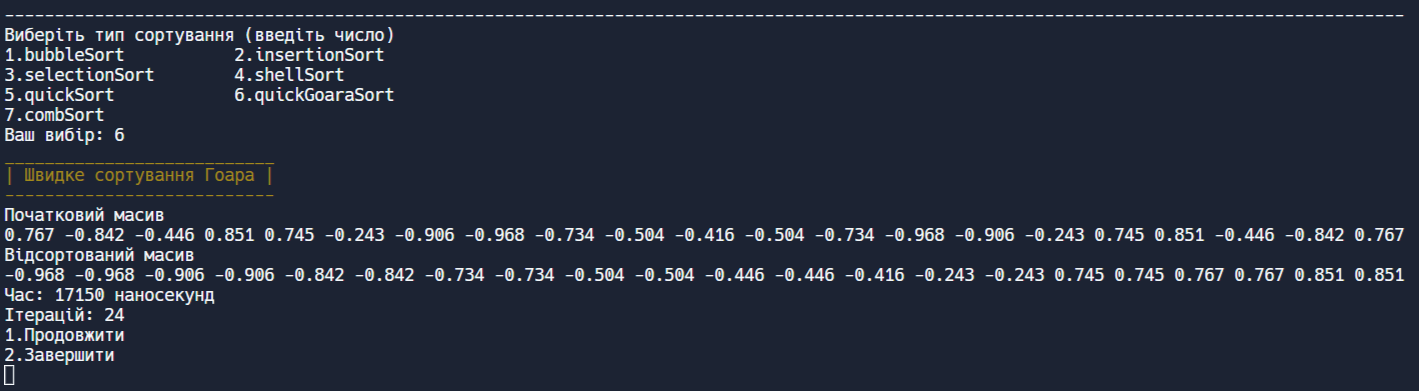


Рис. Виконання швидкого сортування Гоара

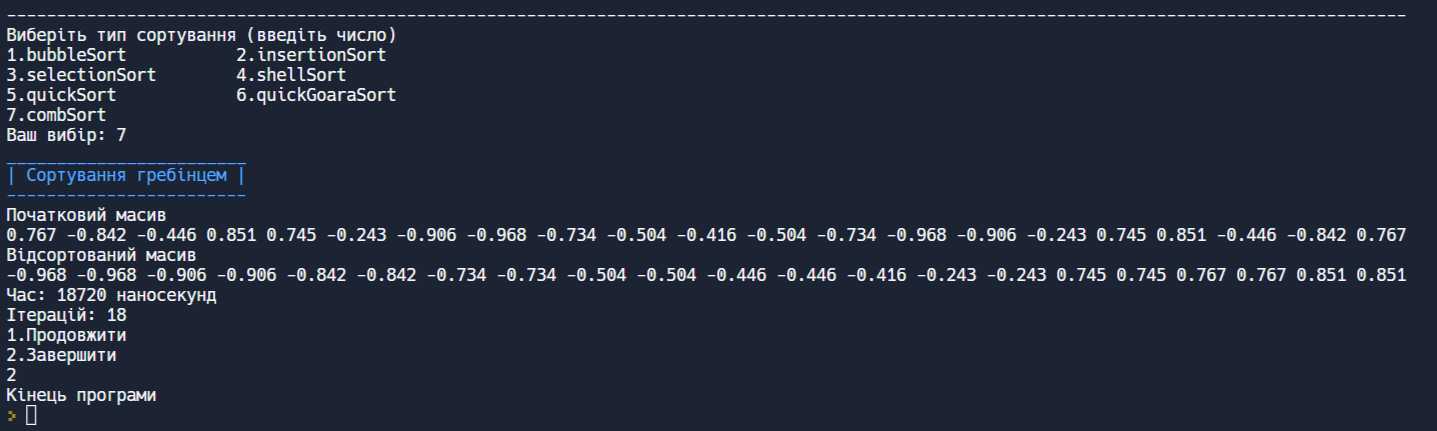


Рис. Виконання сортування гребінцем

**Порівняння ефективності**

|  |  |  |
| --- | --- | --- |
| Тип сортування | Час(наносекунди) | Кількість ітерацій |
| bubbleSort | 39841 | 100 |
| insertoin sort | 32970 | 100 |
| selection sort | 21720 | 20 |
| shellSort | 18510 | 37 |
| quickSort | 19280 | 28 |
| quickGoaraSort | 17150 | 24 |
| combSort | 18720 | 18 |

**Висновки**

На цій лабораторній роботі було ознайомлено із алгоритмами сортування масивів та способами їх реалізації. У якості індивідуального завдання написано програмний код, у якому реалізується сортування масивів методами бульбашки, вставок, вибору, сортуванням Шелла, Гоара, швидкого сортування та сортування гребінцем. Виконано порівняння ефективності вказаних методів сортування. Порівнювалися час сортування та кількість ітерацій. У результаті визначено, що при малих розмірах матриці різниця незначна, але коли розмір великий то час таких методів як : combSort, quickSort, quickGoaraSort, shellSort значно менший ніж selectionSort, insertionSort. Самим довшим є метод bubbleSort.

**Додатки**

**Посилання на Replit.com**

[**https://replit.com/@TR-12-Karkushie/Lab1#Main.java**](https://replit.com/@TR-12-Karkushie/Lab1#Main.java)

**Код програми**

/\*

\* Лабораторна робота №1

\* Завдання: Ознайомитись з алгоритмами сортування масивів та способами їхньої реалізації.

\* У якості індивідуального завдання необхідно написати програмний код, у

\* якому реалізується сортування масивів методами бульбашки, вставок, вибору,

\* сортуванням Шелла, Гоара, швидкого сортування та сортування гребінцем. Виконати порівняння

\* ефективності вказаних методів сортування.

\* Індивідуальне завдання:

\* Протабулювати функцію y = cos (x^2 + 2) на відрізку [-π, π] з кроком

\* h = π / 10 і впорядкувати значення функції за зростанням.

\*

\*

\* Виконав студент групи ТР-12

\* Каркушевський Владислав

\* Номер варіанту: 16

\* \*/

import java.util.Scanner;

public class Main {

public static int iteration = 0;

public static void main(String[] args) {

System.out.println("Початок програми\n");

Scanner sc = new Scanner(System.in);

double[] array = new double[21];

double x = -Math.PI;

double h = Math.PI / 10;

for (int i = 0; i < array.length; i++) {

double temp = Math.pow(x, 2) + 2;

array[i] = Math.cos(temp);

x += h;

}

System.out.println("Початковий масив");

System.out.println("Елементами масиву є значення, про табульовані функцією cos(2+x^2), де х є [-π;π] з кроком h=Pi/10");

print\_array\_num(array);

boolean choise = true;

while (choise) {

System.out.println("\n--------------------------------------------------------------------------------------------------------------------------------------------");

System.out.println("Виберіть тип сортування (введіть число)\n1.bubbleSort 2.insertionSort\n3.selectionSort 4.shellSort\n5.quickSort 6.quickGoaraSort\n7.combSort ");

System.out.print("Ваш вибір: ");

int par = sc.nextInt();

if (par == 1) {

System.out.print("\u001B[34m");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("| Сортування бульбашкою |");

System.out.println("-------------------------");

System.out.print("\u001B[0m");

double[] array\_bubble = new double[array.length];

System.arraycopy(array, 0, array\_bubble, 0, 21);

long start = System.nanoTime();

bubbleSort(array\_bubble);

long finish = System.nanoTime();

all\_print(array, array\_bubble, start, finish);

} else if (par == 2) {

System.out.print("\u001B[33m");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("| Сортування вставкою |");

System.out.println("-----------------------");

System.out.print("\u001B[0m");

double[] array\_insertion = new double[array.length];

System.arraycopy(array, 0, array\_insertion, 0, 21);

long start = System.nanoTime();

insertionSort(array\_insertion);

long finish = System.nanoTime();

all\_print(array, array\_insertion, start, finish);

} else if (par == 3) {

System.out.print("\u001B[34m");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("| Сортування вибором |");

System.out.println("----------------------");

System.out.print("\u001B[0m");

double[] array\_selection = new double[array.length];

System.arraycopy(array, 0, array\_selection, 0, 21);

long start = System.nanoTime();

selectionSort(array\_selection);

long finish = System.nanoTime();

all\_print(array, array\_selection, start, finish);

} else if (par == 4) {

System.out.print("\u001B[33m");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("| Сортування методом Шелла |");

System.out.println("----------------------------");

System.out.print("\u001B[0m");

double[] array\_shell = new double[array.length];

System.arraycopy(array, 0, array\_shell, 0, 21);

long start = System.nanoTime();

shellSort(array\_shell);

long finish = System.nanoTime();

all\_print(array, array\_shell, start, finish);

} else if (par == 5) {

System.out.print("\u001B[34m");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("| Швидке сортування |");

System.out.println("---------------------");

System.out.print("\u001B[0m");

double[] array\_quick = new double[array.length];

System.arraycopy(array, 0, array\_quick, 0, 21);

long start = System.nanoTime();

quickSort(array\_quick, 0, array\_quick.length - 1);

long finish = System.nanoTime();

all\_print(array, array\_quick, start, finish);

} else if (par == 6) {

System.out.print("\u001B[33m");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("| Швидке сортування Гоара |");

System.out.println("---------------------------");

System.out.print("\u001B[0m");

double[] array\_goara = new double[array.length];

System.arraycopy(array, 0, array\_goara, 0, 21);

long start = System.nanoTime();

quickGoaraSort(array\_goara, 0, array\_goara.length - 1);

long finish = System.nanoTime();

all\_print(array, array\_goara, start, finish);

} else if (par == 7) {

System.out.print("\u001B[34m");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("| Сортування гребінцем |");

System.out.println("------------------------");

System.out.print("\u001B[0m");

double[] array\_comb = new double[array.length];

System.arraycopy(array, 0, array\_comb, 0, 21);

long start = System.nanoTime();

combSort(array\_comb);

long finish = System.nanoTime();

all\_print(array, array\_comb, start, finish);

} 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 if (end == 1) {

continue;

} else {

System.out.print("\u001B[31m");

System.out.println("error");

System.exit(0);

}

}

System.out.println("Кінець програми");

}

public static void all\_print(double[] start\_array, double[] sort\_array, long start, long finish) {

System.out.println("Початковий масив");

print\_array\_num(start\_array);

System.out.println("Відсортований масив");

print\_array\_num(sort\_array);

System.out.printf("Час: %d наносекунд\n", finish - start);

System.out.println("Ітерацій: " + iteration);

iteration = 0;

}

static void print\_array\_num(double[] num) {

for (int i = 0; i < num.length; i++) {

System.out.printf("%.3f ", num[i]);

}

System.out.println();

}

public static void swap(double[] nums, int x, int y) {//створюємо метод для зміни елементів масиву місцями

double temp = nums[x];

nums[x] = nums[y];

nums[y] = temp;

iteration++;

}

public static void bubbleSort(double[] 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 insertionSort(double[] nums) {//створюємо метод для сортування вставками

int j;

for (int i = 1; i < nums.length; i++) {

double x = nums[i];

j = i;

while (j > 0 && nums[j - 1] > x) {

swap(nums, j, j - 1);

j--;

}

nums[j] = x;

}

}

public static void selectionSort(double[] 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 shellSort(double[] 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) {

swap(nums, j, j - h);

}

}

h = h / 3;

}

}

public static void quickSort(double[] nums, int lowElem, int highElem) {//створюємо метод для швидкого сортування

int middleElem = lowElem + (highElem - lowElem) / 2;

double opora = nums[middleElem];

int i = lowElem, j = highElem;

while (i <= j) {

while (nums[i] < opora) {

i++;

}

while (nums[j] > opora) {

j--;

}

if (i <= j) {

swap(nums, i, j);

i++;

j--;

}

}

if (lowElem < j) {

quickSort(nums, lowElem, j);

}

if (highElem > i) {

quickSort(nums, i, highElem);

}

}

public static void quickGoaraSort(double[] nums, int lowElem, int highElem) {//створюємо метод для сортування методом Гоара

if (lowElem < highElem) {

int splitPoint = partition(nums, lowElem, highElem);

quickGoaraSort(nums, lowElem, splitPoint);

quickGoaraSort(nums, splitPoint + 1, highElem);

}

}

public static int partition(double[] nums, int lowElem, int highElem) {

double pivot = nums[lowElem];

int i = lowElem - 1, j = highElem + 1;

while (true) {

do {

i++;

} while (nums[i] < pivot);

do {

j--;

} while (nums[j] > pivot);

if (i < j) swap(nums, i, j);

else return j;

}

}

public static void combSort(double[] 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;

}

}

}

}

}