Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

**Лабораторна робота №** 4

по дисципліні «Теорія алгоритмів»

на тему: Алгоритми сортування

|  |  |
| --- | --- |
| Виконав:  студент групи ІА-з91  Москаленко В.В.  Дата здачі \_\_\_\_\_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_\_\_\_\_ | Перевірив:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Київ 2020

1. **Завдання.**

Необхідно реалізувати 3 алгоритми сортування:

Бульбашкове сортування

Швидке сортування

Будь-який алгоритм на вибір

1. **Постановка проблеми.**

Ми розглядаємо три алгоритми сортування: бульбашкою, швидке та сортування вибором. Потрібно створити масив великого розміру та заповнити його випадковими числами, клонувати даний масив для кожного алгоритму сортування, зробити заміри часу для порівняння швидкості роботи різних алгоритмів сортування і виконати задані алгоритми сортування. Є готові розв’язки в стандартних бібліотеках.

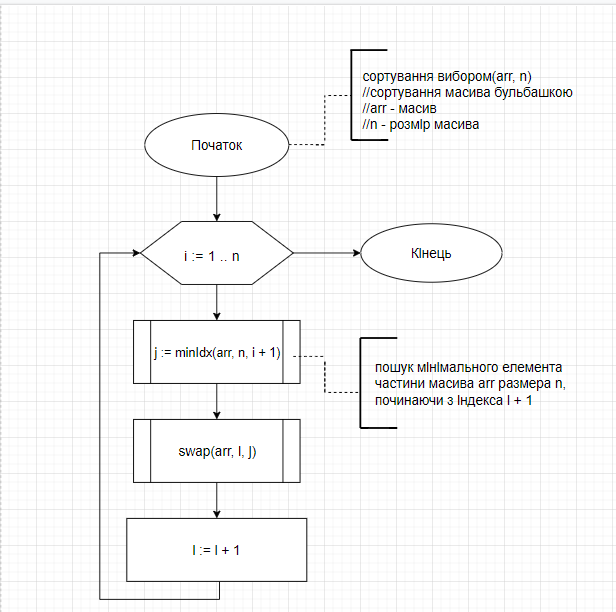
1. **Побудова моделі.**

Для даної моделі ми використаємо стандартний масив 2тис. од., який ран домно наповнимо, клонуємо для 3ох алгоритмів сортування і через загальний інтерфейс виключено кожен з алгоритмів сортування. Кожен алгоритм сортування буде виконаний в окремій функції.

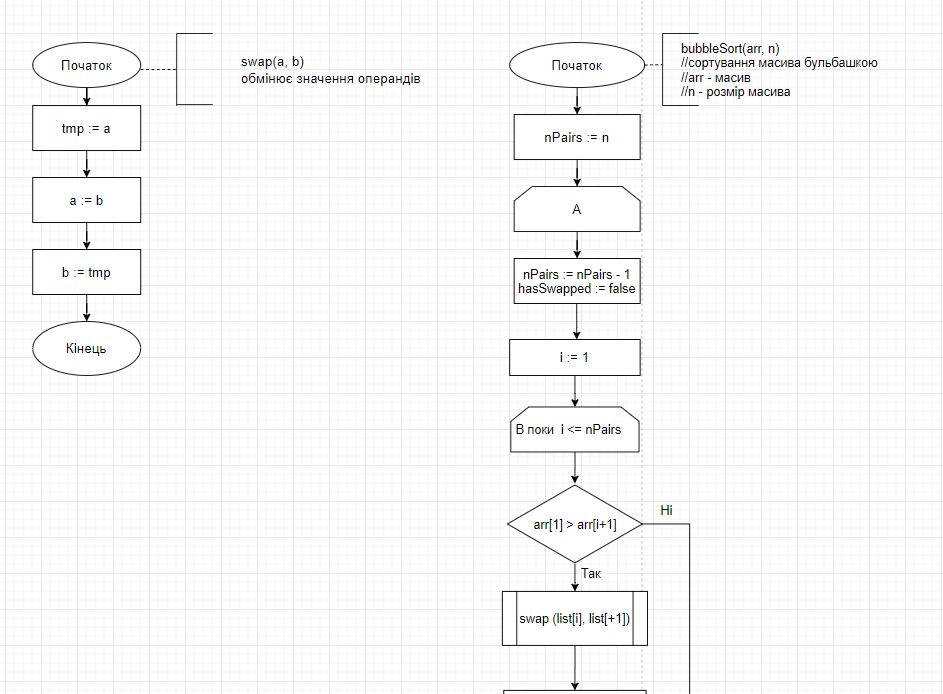
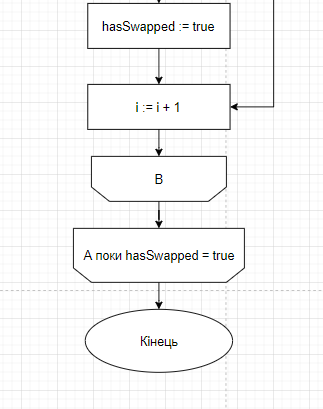
1. **Розроблення алгоритму.**

В main функції створимо користувацький інтерфейс для вибору кожного окремого методу. Так само там створимо масив з рандомним заповненням і блок switch для вибору кожного алгоритму. У блоку switch так само зробимо заміри часу. У кожній функції сортування проведемо відповідне сортування беручи до уваги класичний алгоритм.

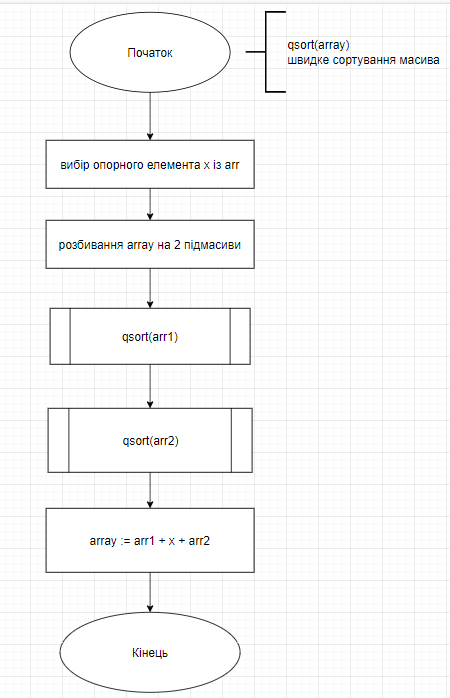
**Алгоритм сортування вибором**



**Алгоритм сортування бульбашкою**



**Алгоритм швидкого сортування**



1. **Перевірка правильності алгоритму.**

Алгоритм працює для всіх int значень, закінчує роботу за скінчений час, який вимірюється. За допомогою функції print візуально перевірений правильний результат роботи. За допомогою функції clone ми копіюємо масив і не псуємо існуючі дані. У кожній з функцій ми використовуємо класичний алгоритм з двома вкладеними циклами і наступні перестановки. Вихід з циклу забезпечується після усіх перестановок. Після цього алгоритм зупиняється.

**5.Реалізація алгоритму.**

import java.io.\*;

import java.lang.\*;

import java.util.Random;

import java.util.concurrent.TimeUnit ;

class Lab4

{

public static void main(String args[])throws IOException

{

int ch;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

do

{

System.out.println("\n\n\n1.Bubble Sort\n2.Selection Sort\n3.Quick Sort.\n4.Exit.");

ch=Integer.parseInt(br.readLine());

if(ch==4)

return;

Random r = new Random();

int n = 2000;

int [] a = new int[n];

int last = 0;

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

last = last + r.nextInt(10) + 1;

a[i] = last;

}

int[] bubbleArr = a.clone();

int[] selectArr = a.clone();

int[] quickArr = a.clone();

switch(ch)

{

case 1:

long timeBub = System.nanoTime();

// Код, который нужно померить

BubbleSort(bubbleArr,n);

timeBub = System.nanoTime() - timeBub;

System.out.printf("Bubble sort timing: %,9.3f ms\n", timeBub/1\_000\_000.0);

long timeBub\_dubl = System.nanoTime();

// Код, который нужно померить

BubbleSort(bubbleArr,n);

timeBub\_dubl = System.nanoTime() - timeBub\_dubl;

System.out.printf("Bubble sort timing: %,9.3f ms\n", timeBub\_dubl/1\_000\_000.0);

break;

case 2:

long timeSel = System.nanoTime();

// Код, который нужно померить

SelectionSort(selectArr,n);

timeSel = System.nanoTime() - timeSel;

System.out.printf("Selection sort timing: %,9.3f ms\n", timeSel/1\_000\_000.0);

long timeSel\_dubl = System.nanoTime();

// Код, который нужно померить

SelectionSort(selectArr,n);

timeSel\_dubl = System.nanoTime() - timeSel\_dubl;

System.out.printf("Selection sort timing: %,9.3f ms\n", timeSel\_dubl/1\_000\_000.0);

break;

case 3:

int start=0;

int end=n-1;

long timeQuick = System.nanoTime();

// Код, который нужно померить

QuickSort(quickArr,start,end);

timeQuick = System.nanoTime() - timeQuick;

System.out.printf("Quick sort timing: %,9.3f ms\n", timeQuick/1\_000\_000.0);

long timeQuick\_dubl = System.nanoTime();

// Код, который нужно померить

QuickSort(quickArr,start,end);

timeQuick\_dubl = System.nanoTime() - timeQuick\_dubl;

System.out.printf("Quick sort timing: %,9.3f ms\n", timeQuick\_dubl/1\_000\_000.0);

//print(a,n);

break;

}

}

while(ch!=4);

}

public static void BubbleSort(int a[],int n)

{

int temp;

boolean swap;

for(int i=0;i<n-1;i++)

{

swap=false;

for(int j=0;j<n-1;j++)

{

if(a[j]>a[(j+1)])

{

temp=a[j];

a[j]=a[(j+1)];

a[(j+1)]=temp;

swap=true;

}

//System.out.print(a[j]);

}

//System.out.println();

if(swap==false)

break;

}

//print(a,n);

}

public static void SelectionSort(int a[], int n)

{

for (int i=0;i<n-1;i++)

{

int imin=i;

int temp;

for(int j=i+1;j<n;j++)

{

if(a[j]<a[imin])

imin=j;

}

temp=a[i];

a[i]=a[imin];

a[imin]=temp;

}

//print(a,n);

}

public static void InsertionSort(int a[],int n)

{

for(int i=1;i<n;i++)

{

int val=a[i];

int hole=i;

while(hole>0&&a[hole-1]>val)

{

a[hole]=a[hole-1];

hole=hole-1;

}

a[hole]=val;

}

print(a,n);

}

public static void QuickSort(int a[],int start,int end)

{

if(start<end)

{

int pIndex=QuickPartition(a,start,end);

QuickSort(a,start,pIndex-1);

QuickSort(a,pIndex+1,end);

}

else

return;

}

public static int QuickPartition(int a[],int start,int end)

{

int temp;

int pivot=a[end];

int pIndex=start;

for(int i=start;i<end;i++)

{

if(a[i]<=pivot)

{

//swap a[i],apindex

temp=a[i];

a[i]=a[pIndex];

a[pIndex]=temp;

pIndex++;

}

}

temp=a[pIndex];

a[pIndex]=a[end];

a[end]=temp;

return pIndex;

}

public static void print(int a[],int n)

{

System.out.println();

for(int i=0;i<n;i++)

System.out.print(a[i]+"\t");

}

private static long timer(Runnable method, TimeUnit timeUnit) {

long time = System.nanoTime();

method.run();

time = System.nanoTime() - time;

System.out.println(TimeUnit.NANOSECONDS.convert(time, timeUnit));

return TimeUnit.NANOSECONDS.convert(time, timeUnit);

}

}

1. **Перевірка програми.**

Дана програма перевірялась на масивах даних до ста одиниць та двух тисяч одиниць. Візуальний контроль був проведений за допомогою виводу елементів масиву у консоль, до і після сортування. Швидкість роботи даного алгоритму залежить від системного планування конкретної ОС.

1. **Вивід програми**

