# МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Національний університет “Львівська політехніка”**



**Інститут післядипломної освіти**

**ЗВІТ**

**Про виконання лабораторної роботи №2**

**«Метод сортування вибором»**

**з дисципліни «Алгоритми і структури даних»**

Виконав:

слухач групи ПЗС-11

Гринчук Тарас

Прийняла:

доц. Кортєєва Т.О.

« »\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2014 р.

∑ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЛЬВІВ – 2014

**Тема роботи**: Метод сортування вибором.

**Мета роботи**: Вивчити алгоритм сортування вибором. Здійснити програмну реалізацію алгоритму сортування вибором. Дослідити швидкодію алгоритму сортування вибором.

## 1. Теоретичні відомості

Сортування вибором (англійською «Selection Sort») — простий алгоритм сортування лінійного масиву, на основі вставок. Має ефективність O(n2), що робить його неефективним при сортування велеких масивів, і в цілому, менш ефективним за подібний алогоримт сортування включенням. Сортування вибором вирізняється більшою простотою, ніж cортування включенням, і в деяких випадках вищою продуктивністю.

Алгоритм працює наступним чином:

1. Знаходить у списку найменше значення.

2. Міняє його місцями із першим значеннями у списку.

3. Повторює два попередніх кроки, доки список не завершиться (починаючи з другої позиції).

Фактично, таким чином ми поділили список на дві частини: перша (ліва) — повністю відсортована, а друга (права) — ні.

Сортування вибором не є складним в аналізі та порівнянні його з іншими алгоритмами, оскільки жоден з циклів не залежить від даних у списку. Знаходження найменшого елементу вимагає перегляду усіх n елементів (у даному випадку (n − 1) порівняння), і після цього, переситановки його до першої позиції. Знаходження наступного найменшого елементу вимагає перегляду (n − 1) елементів, і так далі, для (n − 1) + (n − 2) + ... + 2 + 1 = n(n − 1) / 2 Î Θ(n2) порівнянь. Кожне сканування вимагає однієї перестановки для (n − 1) елементів (останній елемент знаходитиметься на своєму місці).

Реалізація алгоритму ***/алгоритм S /*:**

Задана таблиця елементів *R1 ,R2 ,…,Rn.*

Даний алгоритм реорганізує таблицю у висхідному порядку, тобто для її ключів буде мати місце співвідношення k*1 <k2 <…<kn.*

**Алгоритм S.**

S1. Цикл по індексу проходження. Повторювати кроки S2 - *S* 4 при i=1,2,…,n-1.

S2. Ініціалізація мінімального індексу пошуку: встановити r=i.

S3. Відшукання найменшого ключа. Повторювати при j=i+1*,*i+2,…,n:якщо k*j <kr* то встановити r = j*.*

S4. Перестановка записів. Якщо r !=j, то R*i<->Rr.*

S5. Кінець, Вихід.

**2. Текст програми на мові програмування JAVA**

package \_asd\_lab2;

import java.util.\*;

/\*\*

\* Л/р № 2

\* Метод сортування вибором

\* @author Taras

\*/

public class \_ASD\_Lab2 {

//чи показувати проміжні результати

static final boolean showIntermediateResults = true;

//функція сортування

static void sort(int arr[]) {

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

// Знайдемо мінімальний елемент на

// проміжку індексів [i; a.length);

// спочатку його індекс рівний i

int minValueIndex = i;

// Перебираємо елементи, що залишилися на проміжку

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

// Якщо елемент в позиції j менший

// елемента в позиції minValueIndex, то

// необхідно обновити значення індекса

if (arr[j] < arr[minValueIndex]) {

minValueIndex = j;

}

}

// Міняємо поточний елемент з мінімальним

int temp = arr[i];

arr[i] = arr[minValueIndex];

arr[minValueIndex] = temp;

//виведемо проміжний результат, якщо це потрібно

if(showIntermediateResults) show(arr);

}

}

//вивід одновимірного масиву на екран

static void show(int[] arr) {

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

System.out.print("" + arr[i] + " ");

System.out.println();

}

//перевірка чи масив відсортований у порядку зростяння значень елементів

static boolean checkResults(int[] arr) {

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

if(arr[i] < arr[i-1]) return false;

return true;

}

//основна функція програми

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Array size: ");

if(!sc.hasNextInt()) {

// Якщо користувач ввів не ціле число виводимо відповідне повідомлення

System.out.println("It's not integer value! Program terminate...");

return;

}

// зчитуємо ціле число і створюємо масив

int[] a = new int[sc.nextInt()];

Calendar c = Calendar.getInstance();

Random rand = new Random(c.getTimeInMillis());

//ініціалізація едлементів масиву випадковими числами від 0 до 99

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

a[i] = rand.nextInt(100);

System.out.println("\nBefore sorting:");

show(a);

//засічемо час початку сортування

c = Calendar.getInstance();

long timeStart = c.getTimeInMillis();

//функція сортування (рекурсивна)

sort(a);

//засічемо час завершення сортування

c = Calendar.getInstance();

long timeStop = c.getTimeInMillis();

System.out.println("\nAfter sorting:");

show(a);

System.out.println("\nThe result is " + checkResults(a) +

". Time elapsed: " + (timeStop - timeStart) + " msec.");

}

}

**3. Результат виконання програми**

Запустимо програму на виконання без виведення проміжних результатів на екран: для масиву розміром 20 елементів (рис. 3.1) та 1000 елементів (рис. 3.2).

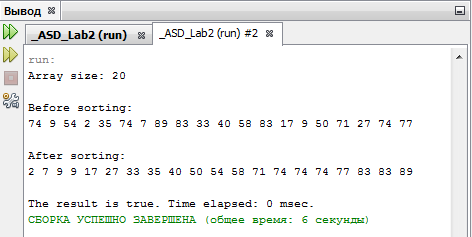


Рис. 3.1. Результат виконання програми

для масиву розміром 20 елементів

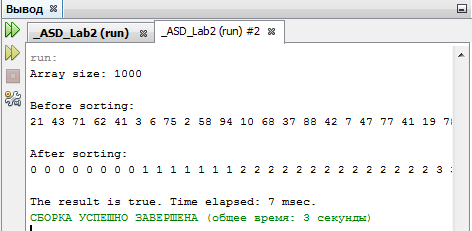


Рис. 3.2. Результат виконання програми

для масиву розміром 1000 елементів

Як бачимо на рисунках, програма виконує сортування в обох випадках правильно, про що свідчить повідомлення про перевірку результату. В першому випадку виконання алгоритму сортування займає час менше одної мілісекунд, а у другому – 7 мілісекунд.

Запустимо програму на виконання в режимі виведення проміжних результатів на екран: для масиву розміром 20 елементів (рис. 3.3) та 1000 елементів (рис. 3.4).

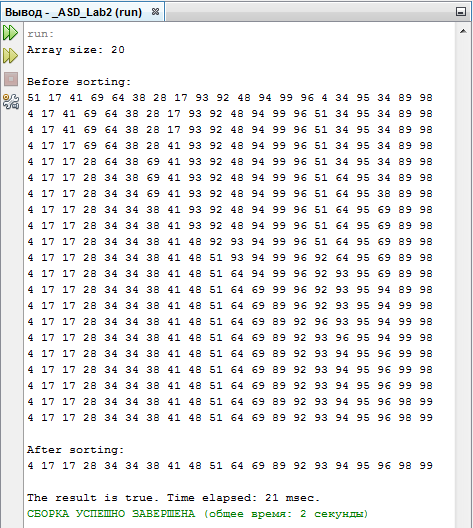


Рис. 3.3. Результат виконання програми

для масиву розміром 20 елементів

(в режимі виводу проміжних даних)

Як бачимо, час виконання програми для сортування 20-ти чисел, суттєво збільшився і становить 21 мілісекунда. Таку різницю можна пояснити значними затратами виводу більшого обсягу інформації на дисплей. Спробуємо вивести всі проміжні дані для масиву, розміром 1000 елементів (рис. 3.4):

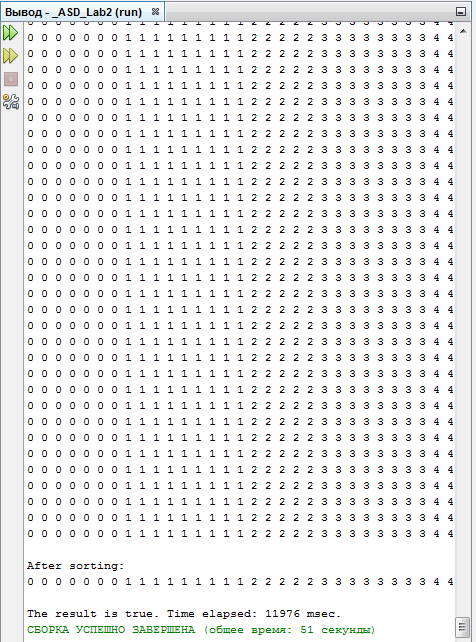


Рис. 3.4. Результат виконання програми

для масиву розміром 1000 елементів

(в режимі виводу проміжних даних)

В даному прикладі різниця в часі виконання – ще більша і становить 11976 мілісекунд.

## ВИСНОВКИ

На даній лабораторній роботі я навчився програмно реалізовувати алгоритм сортування вибором, удосконалив свої навички у написанні підпрограм для роботи з масивами, а також оволодів основними функціями роботи з часом та генератором випадкових чисел у JAVA.