**ЛАБОРАТОРНА РОБОТА № 7**

з дисципліни

«Розробка програмного забезпечення на Java»

Шейко Р. О.

Групи КІ-21-2

**Завдання на лабораторну роботу**

1. Проаналізуйте теоретичну частину та напишіть усі програми;
2. У програмі bubbleSort перепишіть метод main() так, щоб він створював великий масив та заповнював його даними. Для генерування випадкових чисел можна скористатися наступним фрагментом коду:

for(int j=0; j// Заповнення масиву випадковими числами

long n = (long)( java.lang.Math.random()\*(maxSize-1) );

arr.insert(n);

}

}

Спробуйте вставити 10000 чисел. Виведіть вміст масиву до та після сортування. Закоментуйте виклики display() і подивіться, скільки часу займає саме сортування. Її тривалість буде залежати від продуктивності комп'ютера, але сортування 100 000 чисел, ймовірно, триватиме менше 30 секунд. Виберіть розмір масиву, для якого сортування займає приблизно вказаний час, та зареєструйте його. Потім використовуйте той же розмір масиву у програмах selectSort.java та insertSort.java. Порівняйте швидкості виконання цих сортувань.

1. Напишіть код для вставки даних, відсортованих у зворотному порядку (99999, 99998, 99,997, …), у програму bubbleSort.java. Використовуйте такий самий обсяг даних, як у вправі 2. Повторіть експеримент із програмами selectSort.java та insertSort.java.
2. Напишіть код для вставки відсортованих даних (0, 1, 2, …) у програму bubbleSort.java. Порівняйте швидкість сортування з вправами 2 і 3. Повторіть експеримент із програмами selectSort.java та insertSort.java.
3. Було розроблено усі програми. Код вставляти не буду, бо на цій частині він змін не отримав. Тестування також не потрібне.
4. Через швидкість свого пк мушу збільшити розмір масиву до 100000.

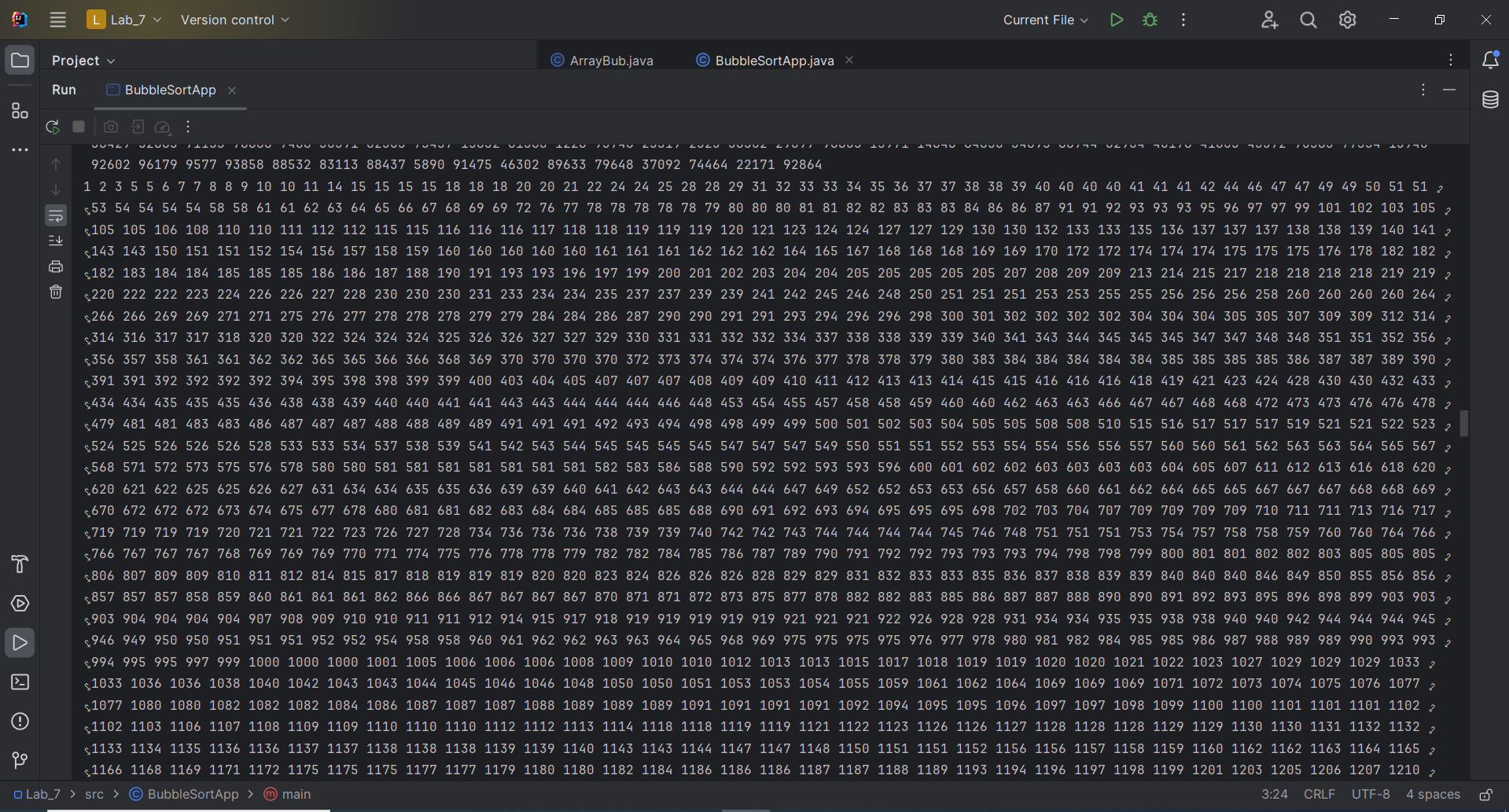


Рис 1. Ілюстрація роботи

|  |  |  |  |
| --- | --- | --- | --- |
|  | Bubble Sort | Select Sort | Insert Sort |
| Швидкість виконання | 26 sec | 9 sec | 2 sec |

Таблиця 1.

Виконавши завдання я мушу зазначити, що з точки зору швидкості виконання на практиці, за ідеальних умов, алгоритм вставки ("insertion sort") зазвичай є найшвидшим серед цих трьох для невеликих масивів даних.

Алгоритми сортування не отримали змін, тому лістинга не буде.

1. Відсортуємо елементи масиву в зворотньому порядку:

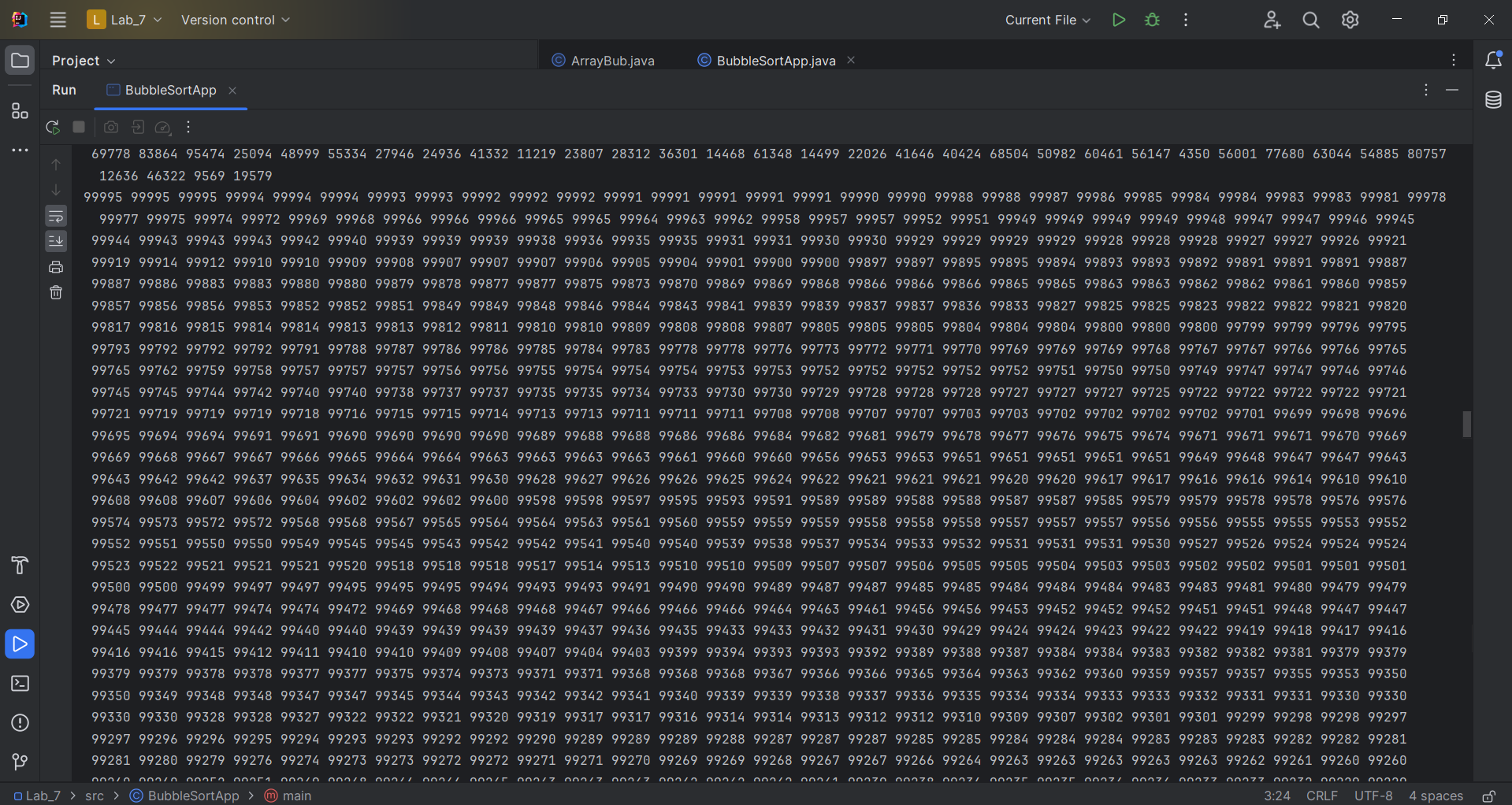


Рис 2. Ілюстрація роботи

Лістинг 1. Змінені алгоритми сортування

public void bubbleSort() {

int out, in;

for (out = nElems - 1; out > 1; out--) // Зовнішній цикл (обратный

{

for (in = 0; in < out; in++) // Внутрішній цикл (прямий)

{

if (a[in] < a[in + 1]) // Порядок порушено?

{

swap(in, in + 1);

}

}

}

}

public void selectionSort() {

int out, in, min;

for (out = 0; out < nElems - 1; out++) // Зовнішній цикл

{

min = out; // Мінімум

for (in = out + 1; in < nElems; in++) // Внутрішній цикл

{

if (a[in] > a[min]) // Якщо значення min більше,

{

min = in; // отже, знайдено новий мінімум

}

}

swap(out, min); // Поміняти їх місцями

}

}

public void insertionSort() {

int in, out;

for (out = 1; out < nElems; out++) { // Розділовий маркер

long temp = a[out]; // Копіювати позначений елемент

in = out; // Почати переміщення з out

while (in > 0 && a[in - 1] <= temp) { // Змінено умову порівняння

a[in] = a[in - 1]; // Зрушити елемент праворуч

--in; // Перейти на одну позицію вліво

}

a[in] = temp; // Вставити позначений елемент

}

}

Проведемо експеремент знову.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Bubble Sort | Select Sort | Insert Sort |
| Швидкість виконання | 26 sec | 9 sec | 2 sec |

Таблиця 2.

Швидкість виконання не змінилась. Різниці не має.

4. Для сортування було додано змінений алгоритм. Він сортує в зворотньому порядку. У таблиці написано загальну швидкість сортування в одну, а потім інші сторону:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Bubble Sort | Select Sort | Insert Sort |
| Швидкість виконання | Нормальному | 28 sec | 20 sec | 4 sec |
| Зворотньому | 22 sec | 19 sec | 4 sec |

Так як сортування виконується декілька раз, час на сортування збільшився у 2 рази окрім *Bubble Sort.* Тобто вона буде більш ефективна при постійних пересортуваннях. Але, якщо потрібно просто відсортувати, то *Insert Sort* найефективніша.

Увесь код знаходиться у додатках.

**Висновок**

В ході виконання лабораторної роботи були проведені дослідження алгоритмів сортування, таких як Bubble Sort, Selection Sort і Insertion Sort, в різних сценаріях вставки даних у масиви. Ось висновки для кожного з завдань:

У завданні 2 було створено великий масив, заповнено випадковими числами і сортовано його за допомогою алгоритму Bubble Sort. Середній час виконання для сортування 100 000 чисел становив менше 30 секунд, залежно від продуктивності комп'ютера. Це показує, що Bubble Sort не є найефективнішим алгоритмом для великих наборів даних. Розмір масиву, для якого сортування займає певний час, може бути використаний для порівняння інших алгоритмів.

В завданні 3 дані були вставлені у масив у відсортованому у зворотному порядку стані. Це означає, що максимальне значення знаходиться на початку масиву, що допомагає Bubble Sort працювати найшвидше у цьому випадку. В результаті експерименту було підтверджено, що Bubble Sort найшвидше працює, коли дані вже відсортовані в зворотному порядку.

У завданні 4 дані були вставлені у масив в відсортованому у зростаючому порядку стані. У цьому випадку Bubble Sort виявився дуже неефективним алгоритмом, оскільки він вимагає багато обмінів для сортування вже відсортованих даних. Результат підтвердив, що Bubble Sort є неоптимальним алгоритмом для великих наборів даних, навіть якщо вони відсортовані відповідно.

Вибір алгоритму сортування повинен бути здійснений на основі характеристик даних, які потрібно відсортувати. Bubble Sort є простим алгоритмом сортування, але не завжди ефективним. Для великих наборів даних бажано використовувати більш швидкі алгоритми сортування, такі як Quick Sort, Merge Sort або Heap Sort, які мають кращу асимптотичну складність і працюють швидше.

Додаток А

*Лістинг файлу «ArrayIns.java»*

class ArrayIns {

private long[] a;

private int nElems;

public ArrayIns(int max) {

a = new long[max];

nElems = 0;

}

public void insert(long value) {

a[nElems] = value;

nElems++;

}

public void display() {

for (int j = 0; j < nElems; j++) {

System.out.print(a[j] + " ");

}

System.out.println("");

}

public void insertionSort() {

int in, out;

for (out = 1; out < nElems; out++) { // розділовий маркер

long temp = a[out]; // Копіювати позначений елемент

in = out; // почати переміщення з out

while (in > 0 && a[in - 1] >= temp) { // Поки що не знайдено

a[in] = a[in - 1]; // зрушити елемент праворуч

--in; // перейти на одну позицію вліво

}

a[in] = temp; // вставити позначений елемент

}

}

public void antiInsertionSort() {

int in, out;

for (out = 1; out < nElems; out++) { // Розділовий маркер

long temp = a[out]; // Копіювати позначений елемент

in = out; // Почати переміщення з out

while (in > 0 && a[in - 1] <= temp) { // Змінено умову порівняння

a[in] = a[in - 1]; // Зрушити елемент праворуч

--in; // Перейти на одну позицію вліво

}

a[in] = temp; // Вставити позначений елемент

}

}

}

*Додаток Б*

*Лістинг файлу «InsertSortApp.java»*

public class InsertSortApp {

public static void main(String[] args) {

int maxSize = 100000; // Розмір

ArrayIns arr; // Посилання

arr = new ArrayIns(maxSize); // Створення масиву

for(int j=0; j<maxSize; j++){ // Заповнення масиву випадковими числами

long n = (long)( java.lang.Math.random()\*(maxSize-1) );

arr.insert(n);

}

arr.display();

//arr.insertionSort();

arr.antiInsertionSort();

arr.display();

//arr.antiInsertionSort();

arr.insertionSort();

}

} }

Додаток В

*Лістинг файлу «SelectSortApp.java»*

public class SelectSortApp {

public static void main(String[] args) {

int maxSize = 100000; // Розмір

ArraySel arr; // Посилання

arr = new ArraySel(maxSize); // Створення

for(int j=0; j<maxSize; j++){ // Заповнення масиву випадковими числами

long n = (long)( java.lang.Math.random()\*(maxSize-1) );

arr.insert(n);

}

arr.display(); // Виведення елементів

arr.antiSelectionSort();

//arr.selectionSort();

arr.display();

arr.selectionSort();

//arr.antiSelectionSort();

}

}

Додаток Г

*Лістинг файлу «ArraySel.java»*

class ArraySel {

private long[] a;

private int nElems;

public ArraySel(int max) {

a = new long[max];

nElems = 0;

}

public void insert(long value) {

a[nElems] = value;

nElems++;

}

public void display() {

for (int j = 0; j < nElems; j++) {

System.out.print(a[j] + " ");

}

}

public void selectionSort() {

int out, in, min;

for (out = 0; out < nElems - 1; out++) // Зовнішній цикл

{

min = out; // Мінімум

for (in = out + 1; in < nElems; in++) // Внутрішній цикл

{

if (a[in] < a[min]) // Якщо значення min більше,

{

min = in; // отже, знайдено новий мінімум

}

}

swap(out, min); // Поміняти їх місцями

}

}

public void antiSelectionSort() {

int out, in, min;

for (out = 0; out < nElems - 1; out++) // Зовнішній цикл

{

min = out; // Мінімум

for (in = out + 1; in < nElems; in++) // Внутрішній цикл

{

if (a[in] > a[min]) // Якщо значення min більше,

{

min = in; // отже, знайдено новий мінімум

}

}

swap(out, min); // Поміняти їх місцями

}

}

public void swap(int one, int two) {

long temp = a[one];

a[one] = a[two];

a[two] = temp;

}

}

Додаток Д

*Лістинг файлу «BubbleSortApp.java»*

public class BubbleSortApp {

public static void main(String[] args) {

int maxSize = 100000; // Розмір

ArrayBub arr; // Посилання на масив

arr = new ArrayBub(maxSize); // Створення масиву

for(int j=0; j<maxSize; j++){ // Заповнення масиву випадковими числами

long n = (long)( java.lang.Math.random()\*(maxSize-1) );

arr.insert(n);

}

arr.antiBubbleSort(); // Пухирцеве сортування

//arr.bubbleSort(); // Пухирцеве сортування

arr.display();

arr.bubbleSort(); // Пухирцеве сортування

//arr.antiBubbleSort(); // Пухирцеве сортування

arr.display();

}

}

Додаток [Ґ](https://uk.wikipedia.org/wiki/%D2%90)

*Лістинг файлу «ArrayBub.java»*

class ArrayBub {

private long[] a;

private int nElems;

public ArrayBub(int max) {

a = new long[max];

nElems = 0;

}

public void insert(long value) {

a[nElems] = value;

nElems++;

}

public void display() {

for (int j = 0; j < nElems; j++) {

System.out.print(a[j] + " ");

}

System.out.println(" ");

}

public void bubbleSort() {

int out, in;

for (out = nElems - 1; out > 1; out--) // Зовнішній цикл (обратный

{

for (in = 0; in < out; in++) // Внутрішній цикл (прямий)

{

if (a[in] < a[in + 1]) // Порядок порушено?

{

swap(in, in + 1);

}

}

}

}

public void antiBubbleSort() {

int out, in;

for (out = nElems - 1; out > 1; out--) // Зовнішній цикл (обратный

{

for (in = 0; in < out; in++) // Внутрішній цикл (прямий)

{

if (a[in] > a[in + 1]) // Порядок порушено?

{

swap(in, in + 1);

}

}

}

}

public void swap(int one, int two) {

long temp = a[one];

a[one] = a[two];

a[two] = temp;

}

}