Trường Đại Học Quốc Tế - ĐHQG TP.HCM

LAB REPORT

Course: Algorithms & Data Structures LAB 2

Full Name: Trần Minh Phúc

Student's ID: ITCSIU24070

1 Problem 1

```
Items before sorting: 77 99 44 55 22 88 11 0 66 33
Items after sorting: 0 11 22 33 44 55 66 77 88 99
The number of swaps = 31
The number of comparison = 44
```

```
// demonstrates bubble sort
// to run this program: C>java BubbleSortApp
////////////////////////////////////////////////////////////////
class ArrayBub {
 private long[] a; // ref to array a
 private int nElems; // number of data items
 private int nSwaps; // number of swaps
 private int nComparison;
 // --------------------------------------------------------------

 public ArrayBub(int max) // constructor
 {
  a = new long[max]; // create the array
  nElems = 0; // no items yet
  nSwaps = 0; // no swaps yet
  nComparison = 0;
 }

 // --------------------------------------------------------------
 public void insert(long value) // put element into array
 {
  a[nElems] = value; // insert it
  nElems++; // increment size
 }

 // --------------------------------------------------------------
 public void display() // displays array contents
 {
  for (int j = 0; j < nElems; j++) // for each element,
```

```
   System.out.print(a[j] + " "); // display it
   System.out.println("");
  }

  // ------------------------------------------------------------
  public void bubbleSort() {
   int out, in;

   for (out = nElems - 1; out > 1; out--) // outer loop (backward)
   for (in = 0; in < out; in++) {// inner loop (forward)
    nComparison++;
    if (a[in] > a[in + 1]) // out of order?
    swap(in, in + 1); // swap them
   }
  } // end bubbleSort()
  // ------------------------------------------------------------

  private void swap(int one, int two) {
   long temp = a[one];
   a[one] = a[two];
   a[two] = temp;

   nSwaps++; // increase number of swap by 1
  }

  public int getSwapNumber() {
   return nSwaps;
  }

  public int getnComparison() {
   return nComparison;
  }
  // ------------------------------------------------------------
 } // end class ArrayBub
 ////////////////////////////////////////////////////////////////

 class BubbleSortApp {
```

```
  public static void main(String[] args) {
   int maxSize = 100; // array size
   ArrayBub arr; // reference to array
   arr = new ArrayBub(maxSize); // create the array

   arr.insert(77); // insert 10 items
   arr.insert(99);
   arr.insert(44);
   arr.insert(55);
   arr.insert(22);
   arr.insert(88);
   arr.insert(11);
   arr.insert(00);
   arr.insert(66);
   arr.insert(33);
   System.out.print("Items before sorting: ");
   arr.display(); // display items
   System.out.print("Items after sorting: ");
   arr.bubbleSort(); // bubble sort them

   arr.display(); // display them again

   // display the number of swaps
   System.out.println("The number of swaps = " + arr.getSwapNumber());
   System.out.println("The number of comparison = " + arr.getnComparison());
   System.out.println("The number of comparison after each loop in this sort algorithm is (n
   System.out.println("If we modified the sort method to have the check sorted array conditi

  } // end main()
 } // end class BubbleSortApp
 ////////////////////////////////////////////////////////////////
```

2 Problem 2

```
// selectSort.java
// demonstrates selection sort
// to run this program: C>java SelectSortApp
////////////////////////////////////////////////////////////////
class ArraySel
{
 private long[] a;                 // ref to array a
 private int nElems;               // number of data items
 private int nComparison;
 //--------------------------------------------------------------
 public ArraySel(int max)          // constructor
 {
  a = new long[max];                 // create the array
  nElems = 0;                        // no items yet
 }
 //--------------------------------------------------------------
 public void insert(long value)    // put element into array
 {
  a[nElems] = value;             // insert it
  nElems++;                      // increment size
 }
 //--------------------------------------------------------------
 public void display()             // displays array contents
 {
  for(int j=0; j<nElems; j++)       // for each element,
  System.out.print(a[j] + " ");  // display it
  System.out.println("");
 }
 //--------------------------------------------------------------
 public void selectionSort()
 {
  int out, in, min;

  for(out=0; out<nElems-1; out++)   // outer loop
  {
```

```
    min = out;                      // minimum
    for(in=out+1; in<nElems; in++) {// inner loop
     nComparison++;
     if(a[in] < a[min] ) {        // if min greater,
      min = in;               // we have a new min
     }
     swap(out, min);                // swap them
    }
   }  // end for(out)
   System.out.println("-----------------------------------------");
  }  // end selectionSort()
  //-------------------------------------------------------------
  private void swap(int one, int two)
  {
   long temp = a[one];
   a[one] = a[two];
   a[two] = temp;
  }
  //-------------------------------------------------------------
  public int getnComparison() {
   return nComparison;
  }
 }  // end class ArraySel
 ////////////////////////////////////////////////////////////////
 class SelectSortApp
 {
  public static void main(String[] args)
  {
   int maxSize = 100;            // array size
   ArraySel arr;                 // reference to array
   arr = new ArraySel(maxSize);  // create the array

   arr.insert(77);               // insert 10 items
   arr.insert(99);
   arr.insert(44);
   arr.insert(55);
   arr.insert(22);
```

```
    arr.insert(88);
    arr.insert(11);
    arr.insert(00);
    arr.insert(66);
    arr.insert(33);

    arr.display();                // display items
    System.out.println("-----------------------------------------");
    System.out.println("The items that are swapped are: ");
    arr.selectionSort();          // selection-sort them

    arr.display();                // display them again
    System.out.println("We don't always need to swap elements. Consider a case where the mini
    System.out.println("The number of comparison = " + arr.getnComparison());
    System.out.println("The number of comparison after each loop in this sort algorithm is (n
  }  // end main()
}  // end class SelectSortApp
////////////////////////////////////////////////////////////////
```

```
77 99 44 55 22 88 11 0 66 33
----------------------------------------
The items that are swapped are:
77 and 77. Positions: 0 and 0, respectively;
44 and 77. Positions: 0 and 2, respectively;
55 and 44. Positions: 0 and 3, respectively;
22 and 55. Positions: 0 and 4, respectively;
55 and 22. Positions: 0 and 4, respectively;
11 and 55. Positions: 0 and 6, respectively;
0 and 11. Positions: 0 and 7, respectively;
11 and 0. Positions: 0 and 7, respectively;
0 and 11. Positions: 0 and 7, respectively;
77 and 99. Positions: 1 and 2, respectively;
44 and 77. Positions: 1 and 3, respectively;
22 and 44. Positions: 1 and 4, respectively;
44 and 22. Positions: 1 and 4, respectively;
22 and 44. Positions: 1 and 4, respectively;
11 and 22. Positions: 1 and 7, respectively;
22 and 11. Positions: 1 and 7, respectively;
11 and 22. Positions: 1 and 7, respectively;
77 and 99. Positions: 2 and 3, respectively;
44 and 77. Positions: 2 and 4, respectively;
77 and 44. Positions: 2 and 4, respectively;
44 and 77. Positions: 2 and 4, respectively;
22 and 44. Positions: 2 and 7, respectively;
44 and 22. Positions: 2 and 7, respectively;
22 and 44. Positions: 2 and 7, respectively;
77 and 99. Positions: 3 and 4, respectively;
88 and 77. Positions: 3 and 5, respectively;
```

```
88 and 77. Positions: 3 and 5, respectively;
55 and 88. Positions: 3 and 6, respectively;
44 and 55. Positions: 3 and 7, respectively;
55 and 44. Positions: 3 and 7, respectively;
33 and 55. Positions: 3 and 9, respectively;
77 and 99. Positions: 4 and 5, respectively;
88 and 77. Positions: 4 and 6, respectively;
44 and 88. Positions: 4 and 7, respectively;
66 and 44. Positions: 4 and 8, respectively;
44 and 66. Positions: 4 and 8, respectively;
77 and 99. Positions: 5 and 6, respectively;
88 and 77. Positions: 5 and 7, respectively;
66 and 88. Positions: 5 and 8, respectively;
55 and 66. Positions: 5 and 9, respectively;
77 and 99. Positions: 6 and 7, respectively;
88 and 77. Positions: 6 and 8, respectively;
66 and 88. Positions: 6 and 9, respectively;
77 and 99. Positions: 7 and 8, respectively;
88 and 77. Positions: 7 and 9, respectively;
77 and 99. Positions: 8 and 9, respectively;
----------------------------------------
0 11 22 33 44 55 66 88 77 99
```

```
The number of comparison = 45
```

3 Problem 3

```
// insertSort.java
// demonstrates insertion sort
// to run this program: C>java InsertSortApp
//--------------------------------------------------------------
class ArrayIns
{
 private long[] a;                 // ref to array a
 private int nElems;               // number of data items
 private int nPass = 0, totalPass = 0;
 //--------------------------------------------------------------
 public ArrayIns(int max)          // constructor
```

```
{
 a = new long[max];                 // create the array
 nElems = 0;                        // no items yet
}
//--------------------------------------------------------------
public void insert(long value)    // put element into array
{
 a[nElems] = value;             // insert it
 nElems++;                      // increment size
}
//--------------------------------------------------------------
public void display()             // displays array contents
{
 for(int j=0; j<nElems; j++)       // for each element,
 System.out.print(a[j] + " ");  // display it
 System.out.println("");
}
//--------------------------------------------------------------
public void insertionSort()
{
 int in, out;

 for(out=1; out<nElems; out++)     // out is dividing line
 {
  long temp = a[out];            // remove marked item
  in = out;
  totalPass++;                   // start shifts at out
  while(in>0 && a[in-1] >= temp) // until one is smaller,
  {
   totalPass++;
   nPass++;
   a[in] = a[in-1];            // shift item to right
   --in;                       // go left one position
  }
  a[in] = temp;                  // insert marked item

 }  // end for
```

```
  }  // end insertionSort()
  public int getnPass() {
   return nPass;
  }
  public int getTotalPass() {
   return totalPass;
  }
  //--------------------------------------------------------------
 }  // end class ArrayIns
////////////////////////////////////////////////////////////////
class InsertSortApp
{
 public static void main(String[] args)
 {
  int maxSize = 100;            // array size
  ArrayIns arr;                 // reference to array
  arr = new ArrayIns(maxSize);  // create the array

  arr.insert(77);               // insert 10 items
  arr.insert(99);
  arr.insert(44);
  arr.insert(55);
  arr.insert(22);
  arr.insert(88);
  arr.insert(11);
  arr.insert(00);
  arr.insert(66);
  arr.insert(33);

  arr.display();                // display items

  arr.insertionSort();          // insertion-sort them
  System.out.println("Number of pass in the inner loops (Number of copies) = " + arr.getnPa
  System.out.println("Total number of pass (Both inner and outer loops) = " + arr.getTotall
  System.out.println("The number of pass in the inner loop is 1 + 2 + 3 + ... + n - 1 = n(
 }  // end main()
}  // end class InsertSortApp
```

```
Number of pass in the inner loops (Number of copies) = 31
Total number of pass (Both inner and outer loops) = 40
```

4 Problem 4

- For 10000 items

```
Number of elements: 10000
------------------------------------------------------------
Number of comparison using bubble Sort: 49994999
Number of swap using bubble Sort: 24961883
Number of comparison using selection Sort: 49995000
Number of copy using insertion Sort: 24962856
```

- For 15000 items

```
Number of elements: 15000
------------------------------------------------------------
Number of comparison using bubble Sort: 112492499
Number of swap using bubble Sort: 56097115
Number of comparison using selection Sort: 112492500
Number of copy using insertion Sort: 56099247
```

- For 20000 items

```
Number of elements: 20000
------------------------------------------------------------
Number of comparison using bubble Sort: 199989999
Number of swap using bubble Sort: 99794635
Number of comparison using selection Sort: 199990000
Number of copy using insertion Sort: 99798638
```

- For 25000 items

```
Number of elements: 25000
------------------------------------------------------------
Number of comparison using bubble Sort: 312487499
Number of swap using bubble Sort: 157482166
Number of comparison using selection Sort: 312487500
Number of copy using insertion Sort: 157488393
```

- For 30000 items

```
Number of elements: 30000
------------------------------------------------------------
Number of comparison using bubble Sort: 449984999
Number of swap using bubble Sort: 225755474
Number of comparison using selection Sort: 449985000
Number of copy using insertion Sort: 225764173
```

- For 35000 items

```
Number of elements: 35000
------------------------------------------------------------
Number of comparison using bubble Sort: 612482499
Number of swap using bubble Sort: 305948700
Number of comparison using selection Sort: 612482500
Number of copy using insertion Sort: 305961019
```

- For 40000 items

```
Number of elements: 40000
------------------------------------------------------------
Number of comparison using bubble Sort: 799979999
Number of swap using bubble Sort: 400284601
Number of comparison using selection Sort: 799980000
Number of copy using insertion Sort: 400300655
```

- For 45000 items

```
Number of elements: 45000
------------------------------------------------------------
Number of comparison using bubble Sort: 1012477499
Number of swap using bubble Sort: 504471958
Number of comparison using selection Sort: 1012477500
Number of copy using insertion Sort: 504492457
```

- For 50000 items

```
Number of elements: 50000
------------------------------------------------------------
Number of comparison using bubble Sort: 1249974999
Number of swap using bubble Sort: 627585796
Number of comparison using selection Sort: 1249975000
Number of copy using insertion Sort: 627610770
```

Table for analyzing:

	COPIES/ COMPARISONS/ SWAPS		
	Bubble Sort	**Selection Sort**	**Insertion Sort**
10000	0/49994999/24961883	0/49995000/0	24962856/0/0
15000	0/112492499/56097115	0/112492500/0	56099247/0/0
20000	0/199989999/99794635	0/199990000/0	99798638/0/0
25000	0/312487499/157482166	0/312487500/0	157488393/0/0
30000	0/449984999/225755474	0/449985000/0	225764173/0/0
35000	0/612482499/305948700	0/612482500/0	305961019/0/0
40000	0/799979999/400284601	0/799980000/0	400300655/0/0
45000	0/1012477499/504471958	0/1012477500/0	504492457/0/0
50000	0/1249974999/627585796	0/1249975000/0	627610770/0/0

This table shows the number of COPIES/COMPARISONS/SWAPS with respect to number of elements in the arrays. As illustrated on the table, bubble sort take the most number of operators among the others, in contrast to insertion sort.

5 Problem 5

```
public class Student implements Comparable<Student>
{
 private String fname, lname;
 private int grade;

 public Student(String fname, String lname, int grade)
 {
  this.fname = fname;
  this.lname = lname;
  this.grade = grade;
 }

 public String toString()
 {
  return fname + " " + lname + "\t" + grade;
 }
```

```
// compare by grade (lower grade => negative), then by last name, then first name
@Override
public int compareTo(Student other)
{
 if (other == null) return 1;
 int gradeCompare = Integer.compare(this.grade, other.grade);
 if (gradeCompare != 0) return gradeCompare;
 int lnameCompare = this.lname.compareTo(other.lname);
 if (lnameCompare != 0) return lnameCompare;
 return this.fname.compareTo(other.fname);
}

public static void insertionSortStudent(Student[] a, int nElems)
{
 int in, out;

 for(out=1; out<nElems; out++)     // out is dividing line
 {
  Student temp = a[out];            // remove marked item
  in = out;                       // start shifts at out
  while(in>0 && a[in-1].compareTo(temp)>0) // until one is smaller,
  {
   a[in] = a[in-1];            // shift item to right
   --in;                       // go left one position
  }
  a[in] = temp;                  // insert marked item
 }  // end for
}  // end insertionSort()
public static void displayStudents(Student[] a, int nElems)
{
 for(int j=0; j<nElems; j++)       // for each element,
 System.out.println(a[j]);  // display it
 System.out.println("");
}

public static void main(String[] args)
```

```
 {
  Student[] students = new Student[10];
  students[0] = new Student("John", "Doe", 90);
  students[1] = new Student("Jane", "Smith", 85);
  students[2] = new Student("Alice", "Johnson", 92);
  students[3] = new Student("Bob", "Brown", 78);
  students[4] = new Student("Charlie", "Davis", 88);
  students[5] = new Student("Eve", "Miller", 95);
  students[6] = new Student("Frank", "Wilson", 80);
  students[7] = new Student("Grace", "Moore", 89);
  students[8] = new Student("Henry", "Taylor", 91);
  students[9] = new Student("Ivy", "Anderson", 87);
  Student.displayStudents(students, 10);
  Student.insertionSortStudent(students, 10);
  Student.displayStudents(students, 10);
 }
}
```

This is the end of the report