Algorithms and Data Structures (ADS2)

Laboratory Sheet 1 – Sample Solutions

Part 1

a) Following the pseudocode for INSERTION-SORT introduced in Lecture 1 (slide 13), implement the `InsertionSort` algorithm in Java.

```
public class InsertionSort{

  public static void sort(int a[]){
    int n = a.length;
    for (int j = 1; j < n; j++){
      int key = a[j];
      int i = j-1;
      while ((i >= 0) && (a[i] > key)){
        a[i+1] = a[i];
        i--;
      }
      a[i+1] = key;
    }
  }
}
```

b) Write a test program, `TestSortingAlgorithms`, to check that your implementation is correct, namely that the output array is in *ascending order*.

```
public class TestSortingAlgorithms{

  public static boolean isSorted(int a[]){
    int n = a.length;
    for (int i = 0; i < n-1; i++){
      if (a[i] > a[i+1]){
        return false;
      }
    }
    return true;
  }
}
```

c) What is the complexity of `TestSortingAlgorithms`?

O(n)

d) Implement `InsertionSortDescending` to sort arrays in *descending order*.

```
public class InsertionSortDescending {

  public static void sort(int a[]) {
    int n = a.length;
    for (int j = 1; j < n; j++) {
      int key = a[j];
      int i = j-1;
      while ((i >= 0) && (a[i] < key)) {
        a[i+1] = a[i];
        i--;
      }
      a[i+1] = key;
    }
  }
}
```

Part 2 SELECTION-SORT is a sorting algorithm informally described as follows:

Input: an array A of integers (with indices between 0 and n-1)
Output: a permutation of the input such that $A[0] \leq A[1] \leq \ldots \leq A[n-1]$

Algorithm: Array A is imaginary divided into two parts - sorted one and unsorted one. At the beginning, the sorted part is empty, while unsorted one contains the whole array. The algorithm sorts A by repeatedly picking the minimum element from the unsorted subarray and moving it to the end of the sorted subarray.

a) Write pseudocode for SELECTION-SORT corresponding to the natural language description above.

```
SELECTION-SORT(A)
  for i = 0 to n-2
    index := i
    for j = i+1 to n-1
      if A[j] < A[index] then
        index := j
    min := A[index]
    A[index] := A[i]
    A[i] := min
```

b) What is the running time of SELECTION-SORT in the worst case? And in the best case? How does it compare to INSERTION-SORT?

SELECTION-SORT has quadratic running time in both cases while INSERTION-SORT has linear running time in the best case.

c) Is it a stable sorting algorithm?

It is not stable (show that the relative order of elements with equal keys is changed).

d) Does it sort in-place?

Yes, as the memory requirement is O(1).

e) Implement the `SelectionSort` algorithm in Java following your pseudocode for SELECTION-SORT. Use `TestSortingAlgorithms` to check that your implementation is correct.

```
public class SelectionSort{

  public static void sort(int a[]){
    int n = a.length;
    for (int i = 0; i < n - 1; i++){
      int index = i;
      //find minimum
      for (int j = i + 1; j < n; j++){
        if (a[j] < a[index]){
          index = j;
        }
      }
      //swap
      int min = a[index];
      a[index] = a[i];
      a[i] = min;
```

```
        }
    }
```

Part 3 You have been provided with a suite of test text files `int10.txt`, `int50.txt`, `int100.txt` and `int1000.txt` where each `int`*n*`.txt` contains *n* integers in randomly sorted order. Write a program `TimeSortingAlgorithms.java` to generate timing runs for `InsertionSort` and `SelectionSort` outputting the time taken to sort (an array read from) each of the text files above. Example output might be something like:

```
-------------------------------------------------------
Time taken to sort int10.txt:


InsertionSort: 310 milliseconds

SelectionSort: 530 milliseconds


-------------------------------------------------------
Time taken to sort int50.txt:
(etc.)
```

Only a snippet of the timing method. Method `Scanner.nextInt()` or similar can be used to implement `readArray`.

```
System.out.print("\nInsertionSort: ");
readArray(arr,longName);
time1=System.currentTimeMillis();
InsertionSort.sort(arr);
time2=System.currentTimeMillis();
timeTaken=time2-time1;
System.out.print(timeTaken + " milliseconds\n");
```