COMP 301 Analysis of Algorithms,
Instructur: Zafer Aydın
Lab Assignment 1

Introduction

In this lab you will implement two sorting algorithms: insertion sort and selection sort. Submit your answers to the questions below in a text file (e.g. Word document). Name your file in name_surname.docx format. Submit your solution document and Java codes as a compressed folder (.zip, .rar) in name_surname format to Canvas.

Assignment

1. (a) Implement the ***insertion sort*** algorithm given below in Java. Implement the algorithm as a separate method. You can use the code template `insertion.java`.

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

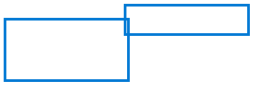
where A is an aray of numbers and the array indices start from 1.

(b) Test your algorithm by choosing an integer array of size 10. Initialize your array by random numbers. Make sure your program sorts arrays correctly. Include the output of your program for this sample input in your report.

(c) Compute and compare the time it takes to sort 1000 integers. Repeat this experiment 1000 times (i.e. in each repetition a new random array should be generated and used as input). Repeat these steps by sorting 10000 integers. Fill the table below for minimum, average and maximum running times. You can complete implementing the `statistics` method in `insertion.java`.

	Minimum Time	Average Time	Maximum Time
Sorting 1000 integers	7400	13023.2	41000
Sorting 10000 integers	12600	26626	33300

(d) How much does the average running time increase by changing the input size from 1000 to 10000? For instance, does it increase by 10-fold or 100 fold, etc? 13,602.80



COMP 301 Analysis of Algorithms,
Instructur: Zafer Aydın
Lab Assignment 1

2. Consider sorting n numbers stored in array A by first finding the smallest element of A and exchanging it with the element in A[1] . Then find the second smallest element of A, and exchange it with A[2]. Continue in this manner for the first n-1 elements of A. This is the ***selection sort*** algorithm.

(a) Implement this algorithm in Java. Implement the algorithm in a separate method. You can use the code template `selection.java`.

(b) Test your algorithm by choosing an integer array of size 10. Initialize your array by random numbers. Make sure your program sorts arrays correctly. Include the output of your program for this sample input in your report.

(c) Compute and compare the time it takes to sort 1000 integers. Repeat this experiment 1000 times (i.e. in each repetition a new random array should be generated and used as input). Repeat these steps by sorting 10000 integers. Fill the table below for minimum, average and maximum running times. You can complete implementing the `statistics` method in `selection.java` or use the `statistics` method you implemented for `insertion.java`.

	Minimum Time	Average Time	Maximum Time
Sorting 1000 integers	108700	239004	2836100
Sorting 10000 integers	10004700	19874437.40	15331200

(d) How much does the average running time increase by changing the input size from 1000 to 10000? For instance, does it increase by 10-fold or 100 fold, etc? 19635433,40

3. Compare the average running times of insertion sort and selection sort. Which algorithm performs better? Can you give any reason?

insertion is better.

COMP 301 Analysis of Algorithms,
Instructur: Zafer Aydın
Lab Assignment 1

You can find the following Java commands useful in this lab

To import class libraries

```
import java.util.Arrays;

import java.util.Random;
```

To generate and store random numbers in an array

```
int data[ ] = new int[10];

Random rand = new Random();

rand.setSeed(System.currentTimeMillis());

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

data[i] = rand.nextInt(100);
```

To measure the running time put the following code before and after your selection sort block

```
long startTime = System.currentTimeMillis();

/* Your selection sort block will go here */

long endTime = System.currentTimeMillis();

long elapsed = endTime – startTime;
```

or you can also use the nanoTime which could be more convenient:

```
long startTime = System.nanoTime();

/* Your sorting block will go here */

long endTime = System.nanoTime();

long elapsed = endTime – startTime;
```