COMP 301 Analysis of Algorithms, Fall 2021
Instructor: Zafer Aydın
Lab Assignment 2

Introduction

In this lab you will implement two sorting algorithms: insertion sort and merge 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 in a separate method called `insertion_sort`. 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) Re-implement insertion sort as a separate method called `insertion_sort_reverse`, which sorts the numbers in descending order.

(c) Choose an integer array of size 10000. Initialize your array by random numbers from 0 to 99. First sort the numbers in this array using insertion sort in ascending order. Then send this sorted array as input to insertion sort again. Measure the time it takes to perform this second sorting, which is equal to the best-case running time of insertion sort for the given set of numbers. Include this time in your report.

(d) Use the same array as in part (c). Sort the numbers in this array in descending order using `insertion_sort_reverse`. Send the output of this method as input to `insertion_sort` so that the numbers are sorted in ascending order again. Measure the time it takes to perform this second sorting, which is equal to the worst-case running time of insertion sort for the given set of numbers. Include this time in your report.

(e) Increase the array size to 100000 and repeat parts (c) and (d). What happens to the best-case and worst-case running times as compared to using an array of 10000 integers.

COMP 301 Analysis of Algorithms, Fall 2021
Instructor: Zafer Aydın
Lab Assignment 2

2. Implement the merge sort procedure given below as a separate method in Java.

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

(a) Rewrite the MERGE procedure so that it does not use sentinels (i.e. ∞), instead stopping once either array L or R has had all its elements copied back to A and then copying the remainder of the other array back into A.

(b) Choose an integer array of size 16. Initialize your array by random numbers. Sort the numbers in this array using merge sort. Include the original array and the sorted array to your report. Make sure merge sort works correctly.

3. (a) Choose an integer array of size 1,048,576. Initialize your array by random numbers. Compute the time it takes to sort these integers using insertion sort and

merge sort. Include these times into your report. Does merge sort run significantly faster then insertion sort for this input size?

(b) For which values of input size *n* (i.e. the number of integers in input array) does the insertion sort runs faster than merge sort? Write a simple Java code that searches these values for *n*. Include these values to your report, which could be specified as a range of integers.

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;
```