HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
BBM204
PROGRAMMING ASSIGNMENT #1

Subject	: Analysis of Algorithms
Submission Date	: 11.03.2019
Deadline	: 25.03.2019
Programming Language	: Java

1) Introduction

In this assignment, you will analyze different algorithms and compare their asymptotic complexity with experimental running times. This will allow you to observe under what conditions the theoretical complexities are valid. You are expected to design an experiment for validating the asymptotical complexities of different algorithms through empirical data collected by executing their implementations.

2) Background

Analysis of algorithms is the area of computer science that provides tools to analyze the efficiency of different methods of solutions. Analysis of algorithm is the process of analyzing the problem-solving capability of the algorithm in terms of the time and size required (the size of memory for storage while implementation). However, the main concern of the analysis of algorithms is the required time or performance. Generally, we perform the following types of analysis

- **Worst-case** − The maximum number of steps taken on any instance of size **a**.
- **Best-case** − The minimum number of steps taken on any instance of size **a**.
- **Average case** − An average number of steps taken on any instance of size **a**.

The efficiency of an algorithm depends on these parameters; i) how much time, ii) memory space, iii) disk space it requires. Analysis of algorithms is mainly used to predict performance and compare algorithms that are developed for the same task. Also, it provides guarantees for performance and helps to understand the theoretical basis. We rely on Asymptotic Complexities to represent the growth rate of execution time, building a theoretical framework for classifying the complexity of algorithms. This framework however is built with simplifying assumptions, ignoring details like compiler optimizations, caching etc.

3) Problem Definition

The main objective of this assignment is to show the relationship between the running time of implementations with the theoretical asymptotic complexities. You are expected to design an experiment showing that the empirical data follows the corresponding asymptotic growth function. To do so, you will have to consider how to reduce the noise in your measurements and plot the running times to reveal the asymptotic complexities. A typical method can be;

i. Generate n random integer numbers,
ii. Run the algorithms on randomly generated integer numbers.
iii. Record the execution times
iv. Carry out steps i-iii for different input size (n)
v. You should plot a graph that will show the relation between n and the execution time

For the algorithms given below you may have different average-case, best-case and worst-case complexities. For instance, Insertion Sort's average case and best case complexities are different. Design an experiment to show both the average and the best case complexities of this algorithm, by generating the inputs accordingly.

- **Insertion Sort algorithm pseudocode code**

```
for i from 1 to N
key = a[i]
j = i - 1
while j >= 0 and a[j] > key
   a[j+1] = a[j]
   j = j - 1
a[j+1] = key
```

- **Radix sort algorithm pseudocode code**

```
Radix-Sort(A, d)
//It works same as counting sort for d number of passes.
//Each key in A[1..n] is a d-digit integer.
//(Digits are numbered 1 to d from right to left.)
    for j = 1 to d do
        //A[]-- Initial Array to Sort
        int count[10] = {0};
        //Store the count of "keys" in count[]
        //key- it is number at digit place j
        for i = 0 to n do
         count[key of(A[i]) in pass j]++

        for k = 1 to 10 do
         count[k] = count[k] + count[k-1]

        //Build the resulting array by checking
        //new position of A[i] from count[k]
        for i = n-1 downto 0 do
         result[ count[key of(A[i])] ] = A[j]
         count[key of(A[i])]--

        //Now main array A[] contains sorted numbers
        //according to current digit place
        for i=0 to n do
          A[i] = result[i]

    end for(j)
end func
```

- **Merge sort algorithm pseudocode code**

```
procedure mergesort( var a as array )
   if ( n == 1 ) return a
   var l1 as array = a[0] ... a[n/2]
   var l2 as array = a[n/2+1] ... a[n]
   l1 = mergesort( l1 )
   l2 = mergesort( l2 )
   return merge( l1, l2 )
end procedure

procedure merge( var a as array, var b as array )
   var c as array
   while ( a and b have elements )
      if ( a[0] > b[0] )
         add b[0] to the end of c
         remove b[0] from b
      else
         add a[0] to the end of c
         remove a[0] from a
      end if
   end while
   while ( a has elements )
      add a[0] to the end of c
      remove a[0] from a
   end while
   while ( b has elements )
      add b[0] to the end of c
      remove b[0] from b
   end while
   return c
end procedure
```

- **Selection Sort Algorithm Pseudocode**

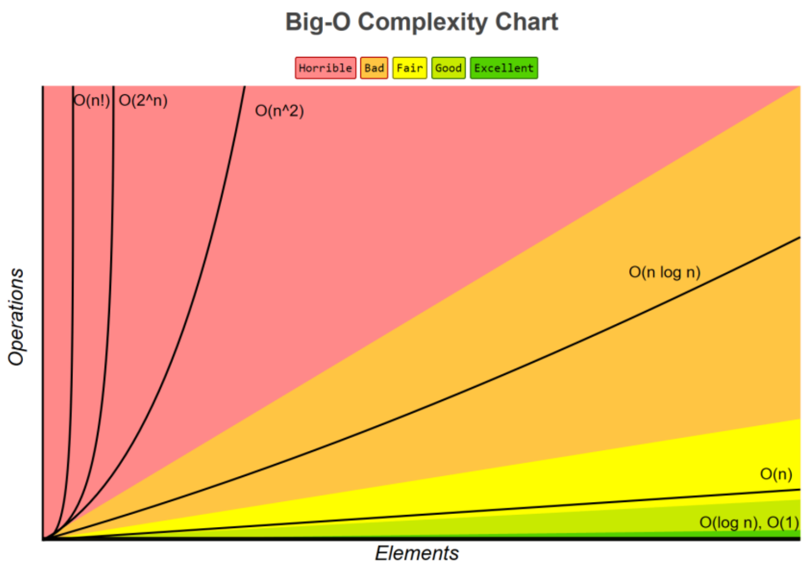
```
procedure selection sort
   list  : array of items
   n     : size of list

   for i = 1 to n - 1
   /* set current element as minimum*/
      min = i

      /* check the element to be minimum */

      for j = i+1 to n
         if list[j] < list[min] then
            min = j;
         end if
      end for

      /* swap the minimum element with the current element*/
      if indexMin != i  then
         swap list[min] and list[i]
      end if
   end for

end procedure
```

- **Binary Search Algorithm Pseudocode**

```
function binary_search(A, n, T):
    L := 0
    R := n − 1
    while L <= R:
        m := floor((L + R) / 2)
        if A[m] < T:
            L := m + 1
        else if A[m] > T:
            R := m - 1
        else:
            return m
          return unsuccessful
```

4) Report

- In your report, you should plot a graph showing the relation between the input size and execution times. You are expected to reveal the Asymptotic complexity of the algorithm with you experiments. For example, for an algorithm with an average case complexity of O(n), you should be able to get the execution times in roughly a line. Some common growth functions in algorithm analysis is shown below. Plot the execution time graph for each method separately indicating the theoretical complexity of the algorithm.
- **Hint:** Executing your algorithm once for an input size might be very noisy leading to plots with lots of spikes. Consider repeating the experiments and averaging the results
- **Hint:** You will require enough number of executions to have a clear plot, consider different number of input sizes to experiment with. If your input sizes are too small, the difference might be too small.
- Your report should also include a plot comparing the best and average case of the algorithms if there is a difference between them. Indicate your data generation strategy in your reports to reveal the different best/worst case complexities.



Grading and Submission

- Your work will be graded over a maximum of 100 points according to the grading policy stated below.
- You must submit both your code and report and you will get 50 points each. You report should clearly cover the following points:
    - **Problem definition**. You should clearly express the idea behind this assignment.
    - **Findings**. You should demonstrate the results with plots, discuss your random number generation strategies for each. Report how you designed your experiments clearly.
    - **Discussion** about how to measure the complexities of algorithms with experiments, what were the challenges and why do you think it was the case.

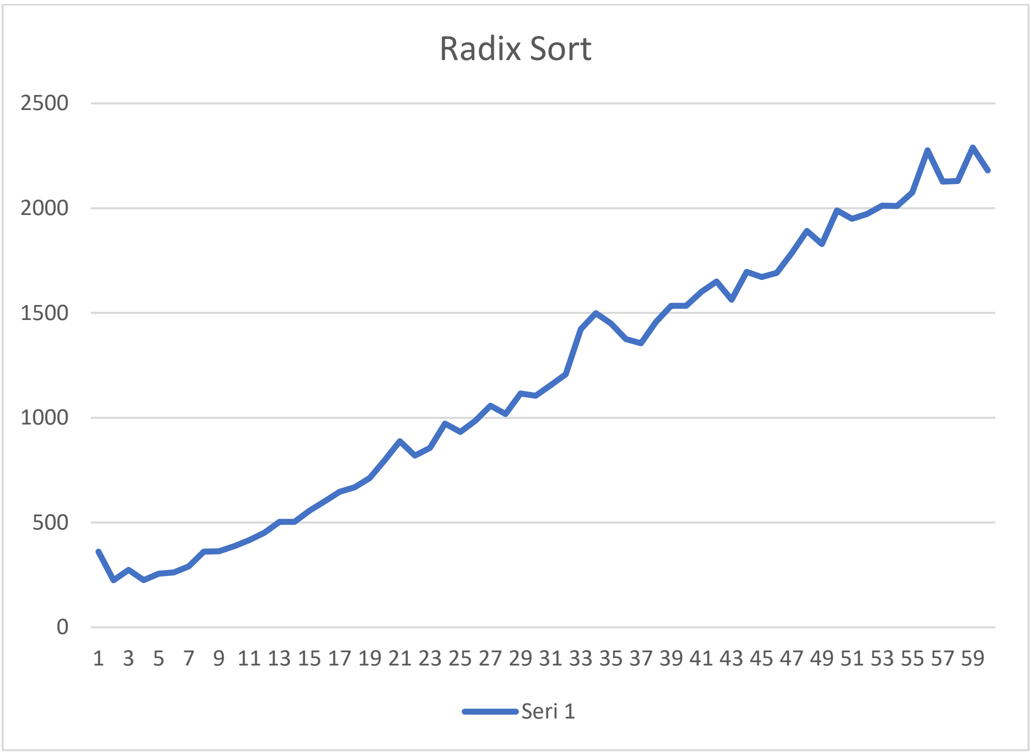
- You will submit your work from https://submit.cs.hacettepe.edu.tr/index.php with the file hierarchy as below:

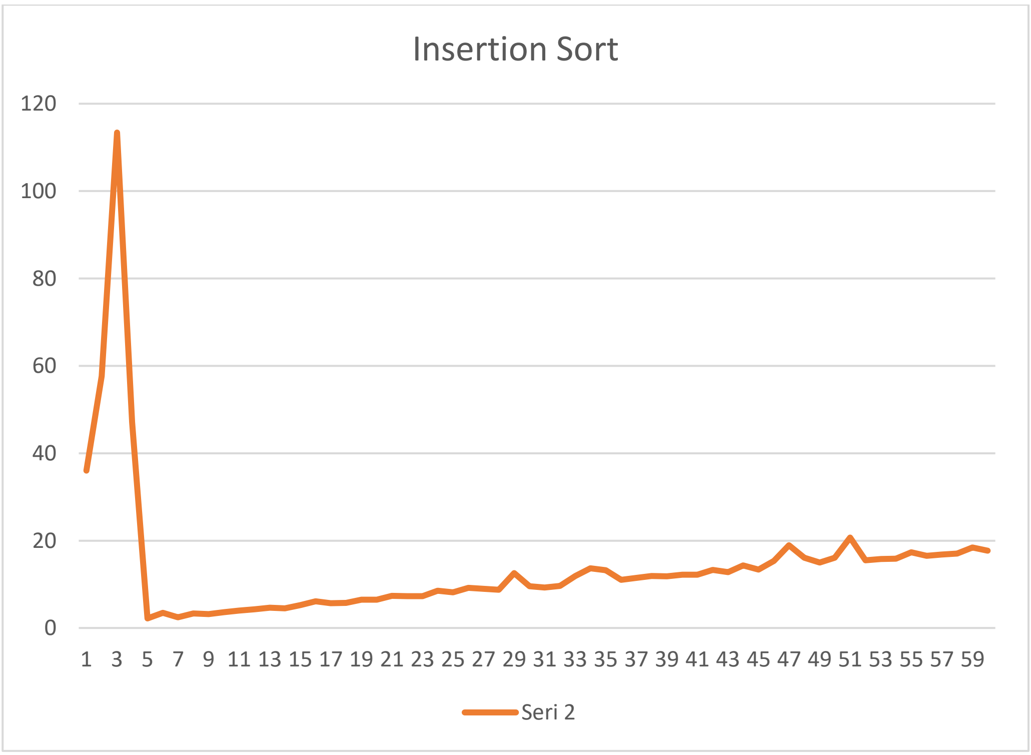
```
<student id.zip>
   → report <DIR>
        → report.pdf <FILE>
   → src <DIR>
        ->Assignment1.java(Required)
        ->*.java(optional)
```

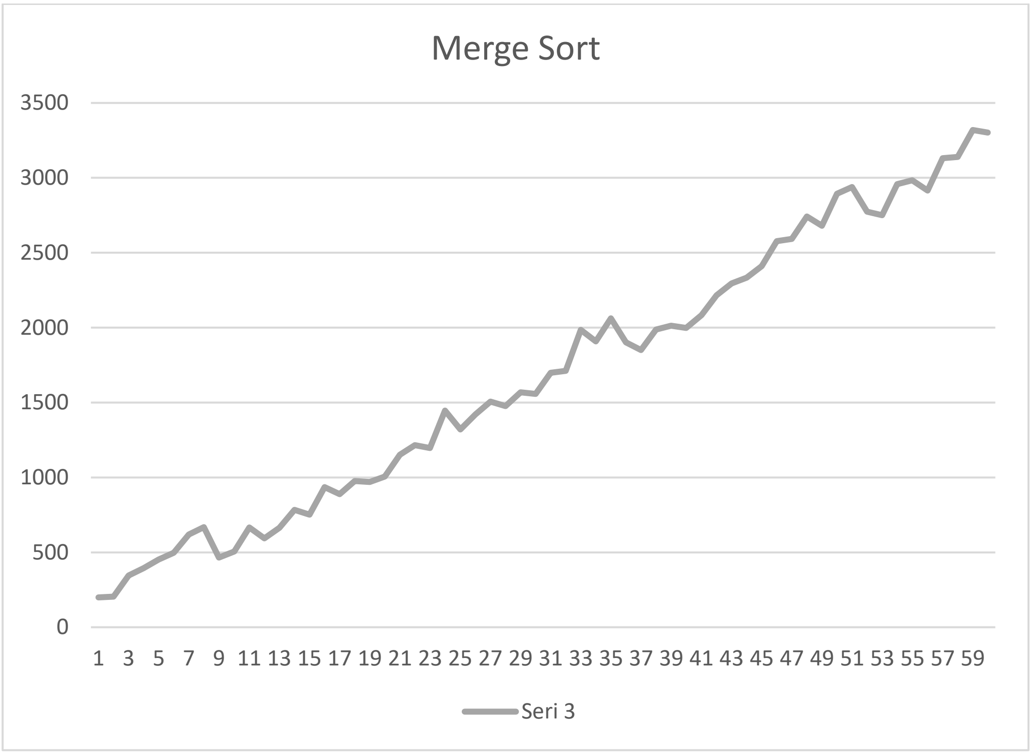
- The class name in which main method belongs should be Assignment1.java. All classes should be placed in src directory. Feel free to create subdirectories, corresponding the package(s), but each should be in src directory.
- This file hierarchy must be zipped before submitted (Not .rar, only .zip file are supported by the system)

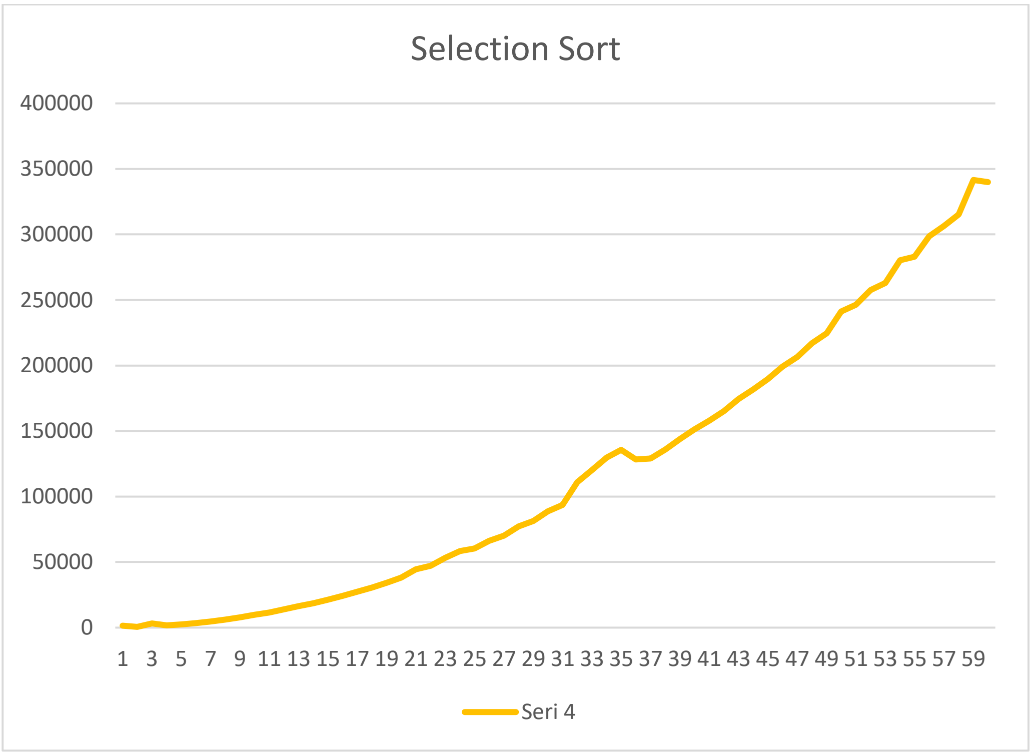
NOTES AND RESTRICTIONS:

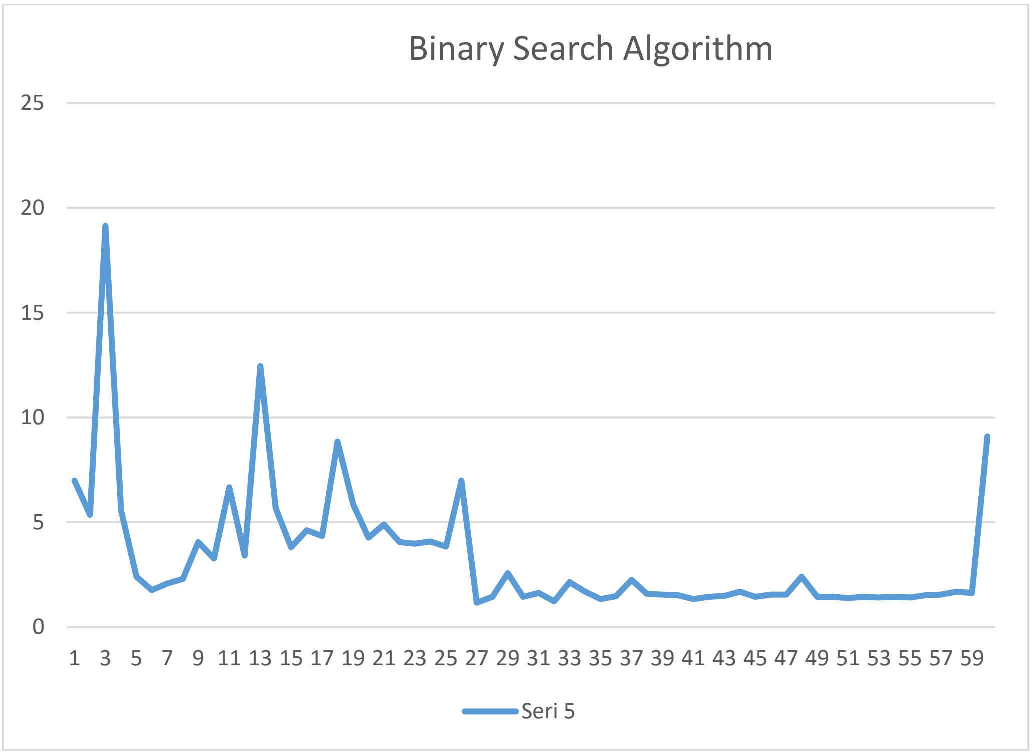
- Your experiment should be submitted before the due date. Late submissions will be penalized.
- All assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudo code) will not be tolerated.
- In short, turning in someone else's work, in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.


Radix Sort
2500
2000
1500
1000
500
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59
Seri 1


Insertion Sort
120
100
80
60
40
20
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59
Seri 2


Merge Sort
3500
3000
2500
2000
1500
1000
500
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59
Seri 3


Selection Sort
400000
350000
300000
250000
200000
150000
100000
50000
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59
Seri 4


Binary Search Algorithm
25
20
15
10
5
0
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59
Seri 5