# ASSIGNMENT 1

**Due on March 24, 2017 (23:59:59)**

**Instructions.** In this assignment, you will analyze different algorithms and compare their running times. You are expected to measure running times of the algorithms which are listed below (in Section 1), and comment about the results. Also you will implement "Finding $N^{th}$ Nearest Touristic Places" algorithm described in Section 2.

## Background

Analysis of algorithms is the area of computer science that provides tools and methodologies to analyze the efficiency of different methods of solutions. Efficiency of an algorithm depends on different parameters; how much time, memory space, disk space etc. it requires. Analysis of algorithms is necessary for lots of reasons but 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 theoretical basis.

A complete analysis of the running time of an algorithm involves the following steps:

- Implement the algorithm correctly.
- Determine the time required for each basic operation.
- Identify unknown quantities that can be used to describe the frequency of execution of the basic operations.
- Develop a realistic model for the input to the program.
- Analyze the unknown quantities, assuming the modelled input.
- Calculate the total running time by multiplying the time by the frequency for each operation, then adding all the products.

**On these experiments, you will measure time requirements of the algorithms and compare their time complexities.** A time complexity analysis should focus on gross differences in the efficiency of algorithms that are likely to dominate the overall cost of a solution. You can analyze the example below:

```
                          Unit Cost       Times
 i:=1                        c1             1
 sum = 0                     c2             1
 while (i <= n)              c3             n+1
 begin
   j:=1;                     c4             n
   while (j <= n)            c5           n*(n+1)
   begin
     sum = sum + i           c6             n*n
```

```
      j:= j + 1              c7            n*n
    end
  i:= i + 1                  c8            n
  end
```

Total Cost = c1 + c2 + (n+1)*c3 + n*c4 + n*(n+1)*c5+n*n*c6 + n*n*c7 + n*c8. The time required for this algorithm is proportional to $n^2$ which is determined as growth rate and it is usually denoted as $O(n^2)$.

**The details of the notations are not given in this assignment paper, you should research and use it for your comments.**

## Part I: Running Time Analysis

The solutions of the problems mentioned in this section should be explained in the form of a report.

1 Analyze the algorithm below and show the total cost of this algorithm. Also find tilde notation.

```
                          Unit Cost        Times
    j:=n                     c1
    while j >= do            c2
    begin
      i:=j                   c3
      while i>=1 do          c4
      begin
        x:=x + 1             c5
        i:= floor (i/2)      c6
      end
    j:= floor (j/2)          c7
    end
```

2 You are given 5 different algorithms for different purposes and their pseudocodes that are listed below.

(a) Finding second largest element in an array at most n+lgn-2 comparisons

(b) Stooge sort algorithm : An optimized version of merge sort algorithm's pseudocode

```
func stoogesort(array L, i = 0, j = length(L)-1)
     if L[j] < L[i] then
        Exchange L[i] ? L[j]
     if j − i > 1 then
         t := (j − i + 1)/3
         stoogesort(L, i  , j-t)
         stoogesort(L, i+t, j  )
         stoogesort(L, i  , j-t)
     return L
end func
```

(c) Radix sort algorithm

```
func RadixSort(A, d)
    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 func
```

(d) Shaker sort algorithm
(for the details, visit <https://en.wikipedia.org/wiki/Cocktail_shaker_sort> )

```
func cocktailShakerSort( A : list of sortable items ) defined as:
  do
    swapped := false
    for each i in 0 to length( A ) − 2 do:
      // test whether the two elements are in the wrong order
      if A[ i ] > A[ i + 1 ] then
        // let the two elements change places
        swap( A[ i ], A[ i + 1 ] )
        swapped := true
      end if
    end for
    if not swapped then
      // we can exit the outer loop here if no swaps occurred.
      break do−while loop
    end if
    swapped := false
    for each i in length( A ) − 2 to 0 do:
      if A[ i ] > A[ i + 1 ] then
        swap( A[ i ], A[ i + 1 ] )
        swapped := true
      end if
    end for
  // if no elements have been swapped, then the list is sorted
  while swapped
end func
```

(e) Maximum subarray algorithm : divide and conquer

```
func maxsub(int[] S; low, high: int)

        if low = high then
            return (low, high, S(low))
        else
            mid = (low + high) / 2
            (llow, lhigh, lsum) = maxsub(S, low, mid)
            (rlow, rhigh, rsum) = maxsub(S, mid+1, high)
            (mlow, mhigh, msum) = middlemaxsub(S, low, mid, high)
        end if;
        return triple with highest sum
end maxsub

func middlemaxsub(int[] S; low, high, int)
        start at mid and find bestleft and leftsum
        start at mid and find bestright and rightsum
        return (bestleft, bestright, rightsum+leftsum)
end middlemaxsub
```

You must execute this algorithms on randomly generated integer numbers. For this purpose, you must carry out the following main steps.

- Generate $n$ random integer numbers.
- Construct a table and save execution time for each algorithms.(See Table 1)
- You must carry out these steps for different $n$ number.
- Finally, you will plot a graph that will show relation between $n$ and execution time

| Algorithms /$n$ | 100 | 300 | 500 | 700 | 1100 | 1300 | 1500 | 1700 | 1900 | 2100 | 2300 | 2500 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Finding second largest element | | | | | | | | | | | | |
| Stooge sort algorithm | | | | | | | | | | | | |
| Radix sort algorithm | | | | | | | | | | | | |
| Shaker sort algorithm | | | | | | | | | | | | |
| Maximum subarray algorithm | | | | | | | | | | | | |

Table 1: Exemplary Table

3 Suppose that the execution of a particular algorithm requires carrying out T(N) operations, where N is the number of inputs that must be processed and

$$T(N) = NlogN + 9N + 55 \tag{1}$$

Assume the algorithm will be executed on hardware capable of performing $10^6$ operations per second and an input of size 220. How long will it take (in seconds)? (Pick the closest answer.)
1) 1
2) 10
3) 20
4) 30
5) 40
6) 50
7) 60
8) 120
9) 1000
10) 10000

## Part II: Finding $N^{th}$ nearest touristic places

Alice visits Barceolana in Spain as a tourist. Since she will be there for a one day and her time is very restricted, she decided to look for only first the nearest "N" touristic places to herself. Places and Alice's location are situated 2-dimensional which are represented (x,y) and (a,b) respectively . In this problem, you are expected to answer the booster distance of $N^{th}$ nearest touristic places from Alice's coordinate. The coordinates of the places and Alice's location will randomly be generated in each execution. N will be entered from the keyboard. Also in each execution 100 touristic places' coordinate pairs and only one Alice's location will randomly be generated which is represented x,y and a,b respectively. But if the nearest place of visit fee is high, she decided to visit the second nearest place whose visit fee is cheaper than the other and so on. So low-cost touristic place has a prioity for Alice. Booster distance is calculated as

$$\sqrt{(x-a)^2 + (y-b)^2} \tag{2}$$

If the result of the distance exceeds 200, the visit fee of the place doesn't be considered. Write an algorithm for Alice to visit nearest and low-cost touristic place. Show Alice's prefered coordinates, booster distances and visit fees according to N number as an output. (Use priority queue to implement this algorithm and consider the exceptions ex. you generated 100 coordinates but you entered 180 for N, the program must give a warning and inform user)

**Constraints**

$-10^3 \leq$ a,b $\leq 10^3$

$-10^3 \leq$ x,y $\leq 10^3$

**Sample Input:**

- N=2 (entered from the keyboard)

- Generate Alice's location randomly. e.g. (-7,10)
- Generate 100 randomly touristic places' coordinates.. e.g. (10,10), (50,-160), (0,3), (-200,1000) etc.
- Generate 100 randomly fees for these places respectively. e.g. 30, 20, 40, 4 (Euro) etc.
- Calculate the Booster distances!

  $\sqrt{(-7-10)^2 + (10-10)^2} \cong 18$ – Second nearest distance

  $\sqrt{(-7-50)^2 + (10+160)^2} \cong 179$ – First nearest distance, because the location fee is less than the other calculated distances' fee, so it must be in priority queue!

  $\sqrt{(-7-0)^2 + (10-3)^2} \cong 10$ – Third nearest distance

  $\sqrt{(-7+200)^2 + (10-1000)^2} \cong 10008$ – ignore low-cost, because it is greater than 200

**Sample Output:**

The booster distances are found!
$1^{th}$ nearest distance calculated with the second generated coordinate is 179
Coordinates of touristic place is (50,-160), location fee is 20

$2^{th}$ nearest distance calculated with the first generated coordinate is 18
Coordinates of touristic place is (10,10), location fee is 30

## Reports

- You must argue the algorithms in terms of time complexity and memory requirements
- You must show your analysis in detail for each algorithm in your report
- Your report will include table,graph and your analysis for the experiment.
- There are no special requirements for the report. If you use a document, code, solution etc. you have to refer related documents.

## Notes

You are required to submit all your code (*all your code should be written in Java*) along with a report in PDF format (should be prepared using LaTeX). The codes you will submit should be well commented. Your report should be self-contained and should contain a brief overview of the problem and the details of your implemented solution. You can include pseudocode or figures to highlight or clarify certain aspects of your solution. Save all your work until the assignment is graded.You can ask your questions about the experiment on Piazza.

You will only submit your report file in the given format below

```
studentid.zip
— – <report>
— – — – report.pdf
— – <src>
— – — – main.java
— – — – *.java
```

Your assignment will not be marked if you do not prepare a table and graph. The ZIP file will be submitted via the department's submission system.

## Grading

The assignment will be graded out of 100 points:

- 0 *(no submission)*, 20 *(an attempt at a solution)*, 40 *(a partially correct solution)*, 60 *(a mostly correct solution)*, 80 *(a correct solution)*, 100 *(a particularly creative or insightful solution)*

Note: Preparing good report is important as well as your solutions!

## Late Policy

You may use up to three *extension* days (in total) over the course of the semester for the three problem sets you will take. Any additional unapproved late submission will be weighted by 0.5.

## Academic Integrity

All work on 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 pseudocode) 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.