HACETTEPE UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING
BBM204
PROGRAMMING ASSIGNMENT #1

Subject **:** Analysis of Sorting Algorithms
Submission Date **:** 12.3.2020
Due : 26.3.2020
Programming Language: Java

1) INTRODUCTION

In this assignment, you will be able to observe and analyze various sorting algorithms and compare their running time considering different data sizes in order to explore their characteristics for evaluating their feasibility in various scenarios. Besides, you will be expected to compare them with other algorithms for the same application.

2) BACKGROUND

Analysis of algorithms is one of the important fields of computer science that provides tools to analyze the efficiency of different methods. In other words, analysis of algorithms is the process of analyzing the problem-solving capability of the algorithm in terms of the time and space required (the size of memory for storage during computation). However, the main concern of the concept is the required time or performance. In general, we perform the following kinds 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; a) how much time, b) memory space, c) disk space it requires. Analysis of algorithms is mainly used to predict performance and to compare algorithms that are developed for the same task. Also, it provides guarantees for performance and helps to understand the theoretical basis. A complete analysis of the running time of an algorithm involves the following steps:

- Implementation of the algorithm
- Determining the time required for each basic operation
- Identifying unknown quantities that can be used to describe the frequency of execution of the basic operations
- Developing a realistic model to be the input for the program
- Analyzing the unknown quantities, assuming the modeled input.
- Calculating the total running time by multiplying the time by the frequency for each operation, then adding all the products.

In this experiment, you will analyze different sorting algorithms and compare their running times on a number of the dataset with changing sizes.

3) PROBLEM DEFINITION

The main objective of this assignment is to analyze the runtimes of different sorting algorithms. To achieve this goal, you are expected to:

- **i.** Generate ***n*** random integer numbers,
- **ii.** Run all the algorithms given below on the randomly generated integer numbers,
- **iii.** Conduct the experiments for different values of ***n***,
- **iv.** Plot a chart that will show the relation between ***n*** and related execution time,
- **v.** The execution duration of listed algorithms will depend on the size of the unsorted array. Compare this algorithm for **best case**, **average case** and **worst case**. Meanwhile, you should explain how you get these results.

Cocktail Shaker Sort Algorithm

```
procedure cocktailShakerSort( A : list of sortable items )
  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
        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
end procedure
```

Pigeonhole Sort Algorithm

```
procedure pigeonhole_sort( a : list of sortable items )
    # size of range of values in the list
    # (ie, number of pigeonholes we need)
    my_min = min(a)
    my_max = max(a)
    size = my_max - my_min + 1

    # our list of pigeonholes
    holes = [0] * size

    # Populate the pigeonholes.
    for x in a:
        assert type(x) is int, "integers only please"
        holes[x - my_min] += 1

    # Put the elements back into the array in order.
    i = 0
    for count in range(size):
        while holes[count] > 0:
            holes[count] -= 1
            a[i] = count + my_min
            i += 1
end procedure
```

Bitonic Sort Algorithm

```
# The sequence to be sorted starts at index position low,
# the parameter cnt is the number of elements to be sorted.
procedure compAndSwap(a, i, j, dire):
    if (dire==1 and a[i] > a[j]) or (dire==0 and a[i] < a[j]):
        a[i],a[j] = a[j],a[i]
    end if
end procedure
procedure bitonicMerge(a, low, cnt, dire):
    if cnt > 1:
        k = cnt/2
        for i in range(low , low+k):
            compAndSwap(a, i, i+k, dire)
        bitonicMerge(a, low, k, dire)
        bitonicMerge(a, low+k, k, dire)
    end if
end procedure
procedure bitonicSort(a, low, cnt, dire):
    if cnt > 1:
          k = cnt/2
          bitonicSort(a, low, k, 1)
          bitonicSort(a, low+k, k, 0)
          bitonicMerge(a, low, cnt, dire)
    end if
end procedure
# Caller of bitonicSort for sorting an array of length N in ASCENDING order
procedure sort(a,N, up):
    bitonicSort(a,0, N, up)
end procedure
```

Comb Sort Algorithm

```
procedure combsort( input : list of sortable items )

    gap = input.size // Initialize gap size
    shrink = 1.3 // Set the gap shrink factor
    sorted = false

    while sorted = false
        # Update the gap value for a next comb
        gap = floor(gap / shrink)
        if gap ≤ 1
            gap = 1
            sorted = true // If there are no swaps this pass, we are done
        end if
        # A single "comb" over the input list
        i = 0
        while i + gap < input.size // See Shell sort for a similar idea
            if input[i] > input[i+gap]
                swap(input[i], input[i+gap])
                sorted = false
                // If this assignment never happens within the loop,
                // then there have been no swaps and the list is sorted.
             end if
             i = i + 1
         end while
     end while
 end procedure
```

4) REPORT

In the report, you are supposed to create a chart which plots the relationship between the size of array and execution duration. Each of the functions commonly found as execution duration has a characteristic shape when plotted on a chart. Some of these are shown below. Algorithms given above might have different Big-O complexity. In your chart, you should show the asymptotic properties of each of these algorithms.

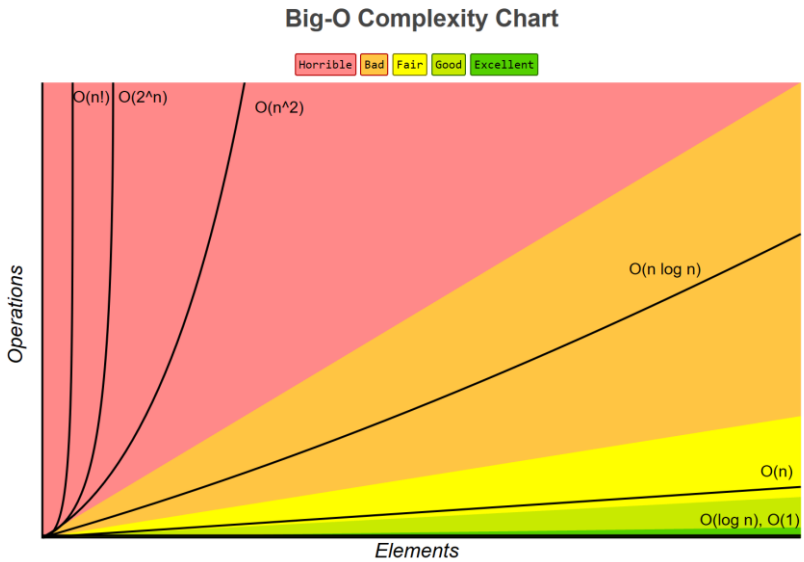
Hint: Run a few trials with different iterations. Using the number of iterations as your x variable and the resulting duration as your y variable, plot a chart. Your results should be compatible with this Big-O complexity chart.

Your report should include the following items for each algorithm:

- Theoretical information about the algorithm include its complexity regarding time and space
- Testing the algorithm with the values of ***n*** such as 5.000, 10.000, 50.000, 100.000, 250.000, 500.000 for best, average and worst case scenarios
- Plot of durations in terms of nanoseconds or milliseconds on a chart
- A table listing the results along with the chart

After completing this stage, you should conclude about the performance of the algorithms among the given ones by following the guide line listed below:

- In which range of ***n***, your best algorithm significantly outperforms the others and why?
- In which range of ***n***, the algorithm(s) you deal with, get worse and why?
- Which of the algorithm(s) you deal with behave(s) fine even on worst case scenario?
- Which of the algorithm(s) you deal with behave(s) badly on average and worst case scenarios?



NOTES AND RESTRICTIONS:

- Your experiment should be submitted (as a single .pdf file named “student_id-report.pdf”) before the due date via submit system. Late submissions will be penalized according to the previously announced rules.
- 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.
- You can ask your questions in the course’s piazza medium. However, your questions will be answered during the day time. Moreover, the questions asked after 5 PM may not be answered during night time. Consequently, we highly suggest you ask your questions before 5 PM.