Lab 1
Implementing Binary Heap & Sorting Techniques

The goal of this lab is to become familiar with the binary heap data structure as well as different sorting techniques.

1. Binary Heap

1.1 Introduction

The (binary) heap data structure is an array object that we can view as a nearly complete binary tree as shown in Figure 1. Each node of the tree corresponds to an element of the array.

The tree is completely filled on all levels except possibly the lowest, which is filled from the left up to a point.

An array A that represents a heap is an object with two attributes: **A.length**, which (as usual) gives the number of elements in the array, and **A.heap-size**, which represents how many elements in the heap are stored within array A.
There are two kinds of binary heaps: max-heaps and min-heaps. In both kinds, the values in the nodes satisfy a heap property.

In a max-heap, the max-heap property is that for every node i other than the root,

$$\mathbf{A[parent[i]] \geq A[i]}$$

1.2 Requirements

In this assignment, you're required to implement some basic procedures and show how they could be used in a sorting algorithm:

- The **MAX-HEAPIFY** procedure, which runs in O (lg n) time, is the key to maintaining the max-heap property.
- The **BUILD-MAX-HEAP** procedure, which runs in linear time, produces a max-heap from an unordered input array.
- The **HEAPSORT** procedure, which runs in O (n lg n) time, sorts an array in place.

You're required to implement the above procedures, pseudo code for the above procedures are explained in details in tutorials

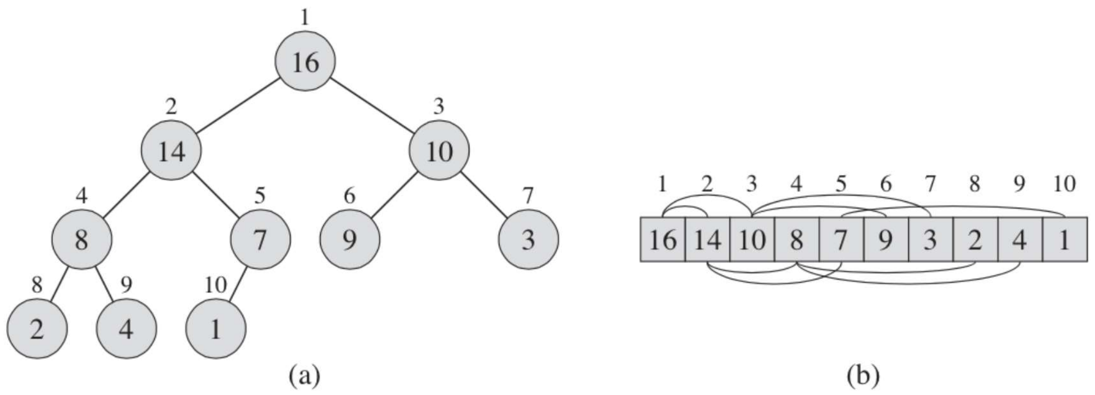

Figure 1

2. Sorting Techniques

You are required to implement the “heapsort" algorithm as an application for binary heaps.

You're required to compare the running time performance of your algorithms against:

- An O (n^2) sorting algorithm such as Selection Sort, Bubble Sort, or Insertion sort.
- An O (n lg n) sorting algorithm such as Merge Sort or Quick sort algorithm in the average case.

In addition to heapsort, select one of the sorting algorithms from each class mentioned above.

To test your implementation and analyze the running time performance, generate a dataset of random numbers and plot the relationship between the execution time of the sorting algorithm versus the input size.

3. Bonus

Implementing more sorting techniques (example: implementing both the quick sort and merge sort)
Graphical Illustration for the operation of different sorting algorithms (check http://www.sorting-algorithms.com/heap-sort)

4. Notes

- Implement your algorithms using (Java or C/C++)
- You should work in groups **of 2 members**
- Also you can extend the group to **3 members** but in this case you are required to implement all O(n lg n) algorithms and only two from O(n^2) algorithms
- Discussion will have higher weight than implementation, so you should understand your implementation well to get discussion marks.

Good Luck