Assignment 1
Sorting techniques

Sorting is one of the most fundamental algorithmic problems within computer science. It has been claimed that as many as 25% of all CPU cycles are spent sorting, which provides a lot of incentive for further study and optimization of sorting. In addition, there are many other tasks (searching, calculating the median, finding the mode, removing duplicates, etc.) that can be implemented much more efficiently once the data is sorted. The wide variety of algorithms gives us a lot of richness to explore, especially when considering the tradeoffs offered in terms of efficiency, operation mix, code complexity, best/worst case inputs, and so on.

1. Lab Goal

- You need to understand:
  - Different between sorting in place (no extra memory required) vs. An extra space
  - Different running times for each algorithms, best and worst case running times

TA: Eng. Sami Mamdouh
Eng. Mohamed Ibrahim
Eng. Reem Hesham
Eng. Omar Aly

Prof Dr. Amr Elmasry

2. Binary Head and Heap sort

2.1 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,

A[parent[i]] ≥ A[i]

2.2 Requirements

You're required to implement some basic procedures:

- 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

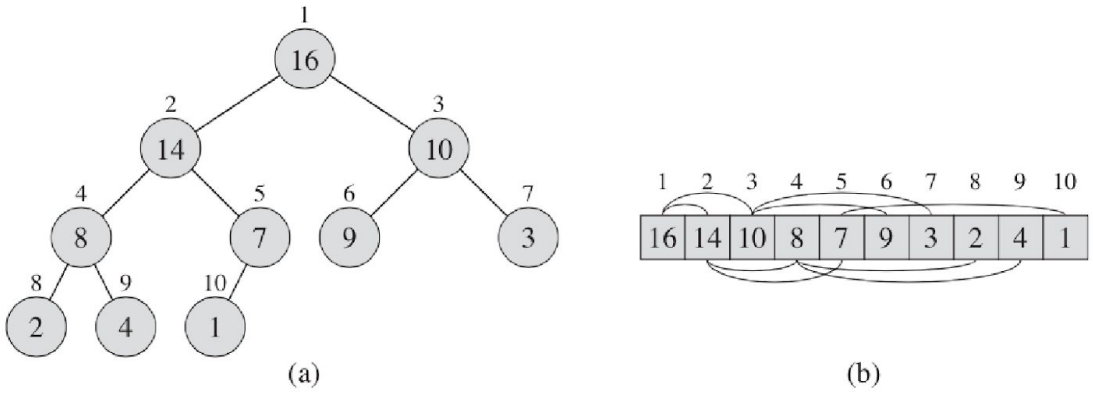


Figure 1

You are required to implement the **"heapsort"** algorithm as an application for binary heaps, you main should expect loading a file with numbers to sort and print them with the time taken to do the sorting.

3. Sorting Techniques

You're required to implement:

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

Now you have implemented 3 sort techniques: Heap Sort, O(n^2) and O(n log n)

To test your implementation and analyze the running time performance, you will be given 2 files with numbers each in a line you should sort each one of them with different algorithms and print if they have the same numbers or not:

Example:

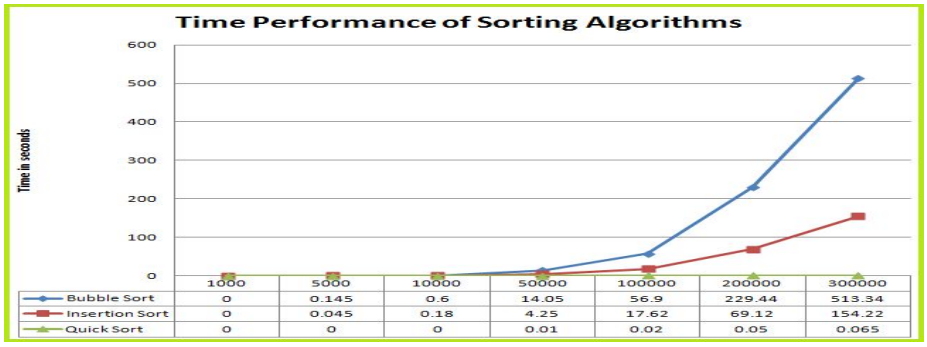
file1.txt	file2.txt
1	2
4	4
2	1
1	1
5	5

Steps:

- ➔ Load file1.txt and sort it with your implementation of O(n^2) algorithms
- ➔ Load file2.txt and sort it with your implementation of O(n log n) algorithms
- ➔ Compare the 2 arrays
- ➔ If equal print:
  - ◆ Files are identical OR Files are not identical
  - ◆ Running time for O(n log n) is 90ms
  - ◆ Running time for O(n^2) is 130ms
- ➔ For above Example they are identical as the numbers will be as followings for both files:
  1
  1
  2
  4
  5

4. Report

A graph is required between **Time (milliseconds)** vs **Array Size** for different sorting algorithms, generate random arrays of different sizes and calculate the time required to sort it using the algorithms you implemented above and plot the results to a graph as what is shown in the image, you can use Excel sheet to achieve the job.



Summary of Deliverables :

- Part1:
  - Heap Sort
  - Main function that take a file sort it and print the sorted array and time taken
- Part 2:
  - O(n^2) Algorithm
  - O(n log n) Algorithm
  - Main function that take 2 files and print if they are identical and running times
- Part 3:
  - Report

3. References

Some references that can help you understand sorting and its application

- Link: Algorithms and Running times
- Link: Visualization for different sorting algorithms
- Link: Measure time taken by function in C

4. Notes

- Implement your algorithms using (Java, C/C++ or Python)
- We will submit a URL to upload the source code, your compressed files name should be students ids for example 1234_3456.zip
- You should work in groups **of 2 members**
- Discussion will have higher weight than implementation, so you should understand your implementation well to get discussion marks.

Good Luck

TA: Eng. Sami Mamdouh
Eng. Mohamed Ibrahim
Eng. Reem Hesham
Eng. Omar Aly

Prof Dr. Amr Elmasry