Data Structure 2 - Lab 1

Problem statement:

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

different sorting techniques.

1. Binary Heap:

Requirements:

- **MAX-HEAPIFY:** it's a procedure, which runs in $O(\log n)$ and it maintains the max-heap property Its input is a root node. When it is called, it assumes that the binary trees rooted to the left and right of the given node are max-heaps, but that the element at the root node might be smaller than its children, thus violating the max-heap property.
- **BUILD-MAX-HEAP:** produces a max-heap in linear time from an unordered input array.
- **HEAPSORT:** runs in $O(n \log n)$ time and sorts an array in-place
- **MAX-HEAP-INSERT & HEAP-EXTRACT-MAX:** runs in $O(\log n)$ time, allow the heap data structure to implement a priority queue.

2. Sorting Techniques:

Implementing 3 kinds of sorting:

1. Heapsort
2. An $O(n^2)$ sort like selection sort, bubble sort or insertion sort
3. An $O(n \log n)$ sort such as merge sort or quick sort

3. Integration

- ### INode:

1. Constructor

- Uses the arraylist of the heap and the index of the node as a reference for the node in order to access the other nodes.

2. getLeftChild/RightChild/Parent

- Calculates the index of the required node using the index of the current node using the suitable formula and return the node if the index is less than the size of the heap.

3. get/setValue

- getter and setter for the value of the node

• IHeap:

1. Constructor:

- Creates a new empty heap or premade heap using an arraylist.

2. getRoot

- Returns the first element of the arraylist which is always the root of the binary heap.

3. Size

- Returns the size of the heap.

4. Heapify

- It starts with the input node and heapifies up by checking the value of this node and the parent node until it reaches to the root of the heap.
- After it reaches the parent it starts to heapify back down by checking the value of this node and the children nodes until it reaches the last children of the heap.

5. Extract

- It returns the value of the root of the heap, which is the max value in the heap, after it replaces the value of the root with the value of the last node in the heap then removing the last node then it heapifies the heap starting with the root for maintaining the maximum heapify property.

6. Insert

- It creates a new node with the input value and adds it as the last element of the heap then it heapifies the heap starting with the new node.

7. Build

- It inserts every element in the collection one by one using the insert function.

- ISort:

1. HeapSort:

- HeapSort method using the max-heapify property and then reverse it to make it returning the heap.

2. sortSlow:

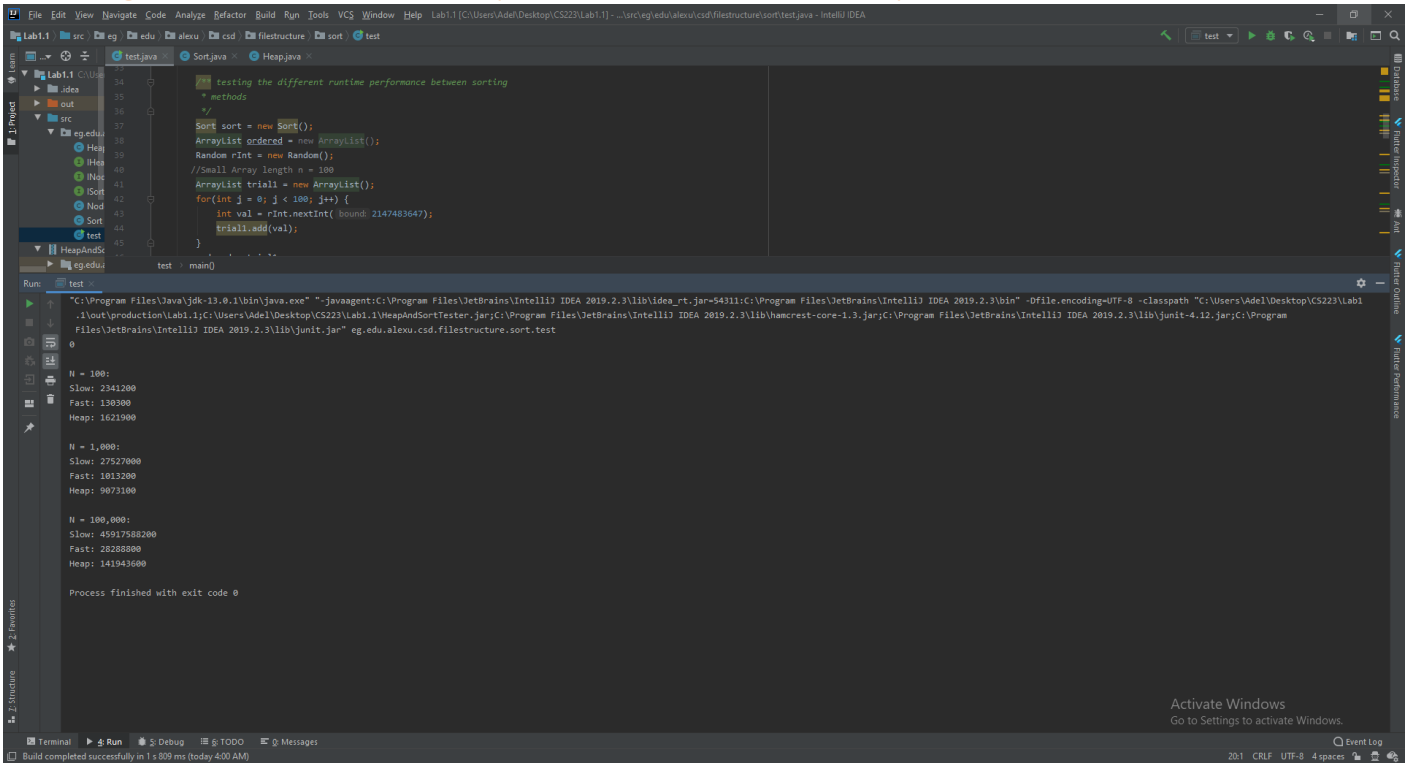
- Sorting Method used is Bubble Sort, it takes $O(n^2)$ time but it's done in place, but it also takes $O(n)$ if the array is already sorted.

3. sortFast:

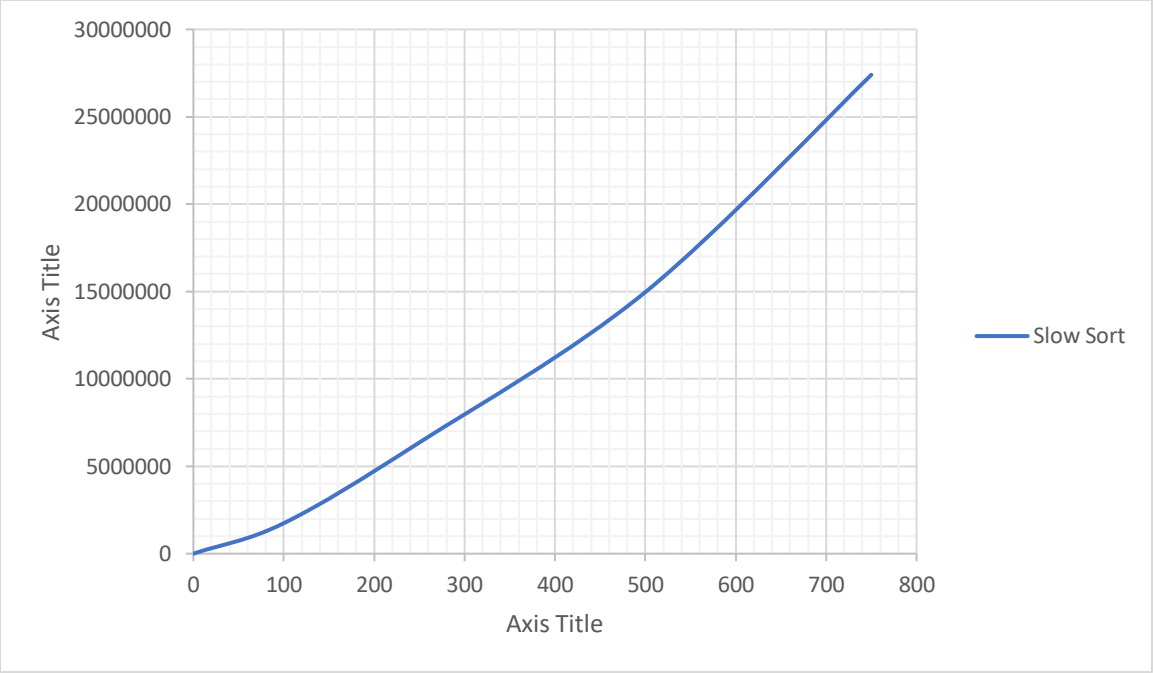
- Sorting Method used is Quicksort (recursive), it takes in average $O(n \log n)$ time but it can be $O(n^2)$ in the worst-case scenario depends on the pivot selected:
- Pivot: Middle element in the array

Tests:

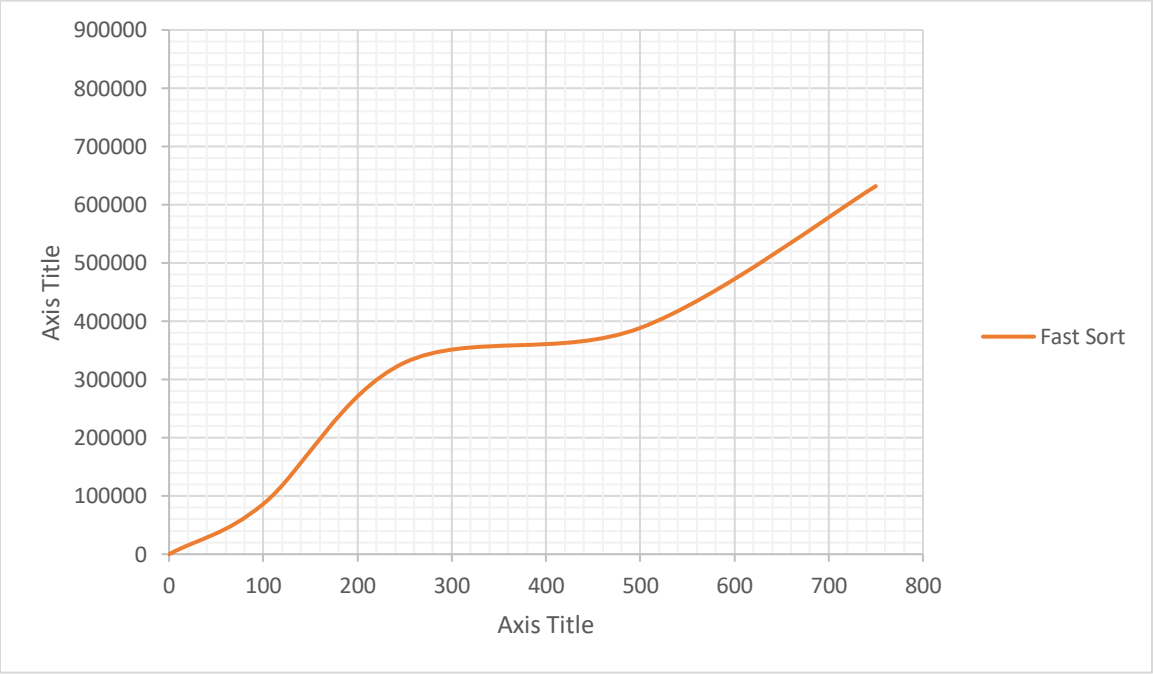
Sorting time differences: (Time in nanoSecond)



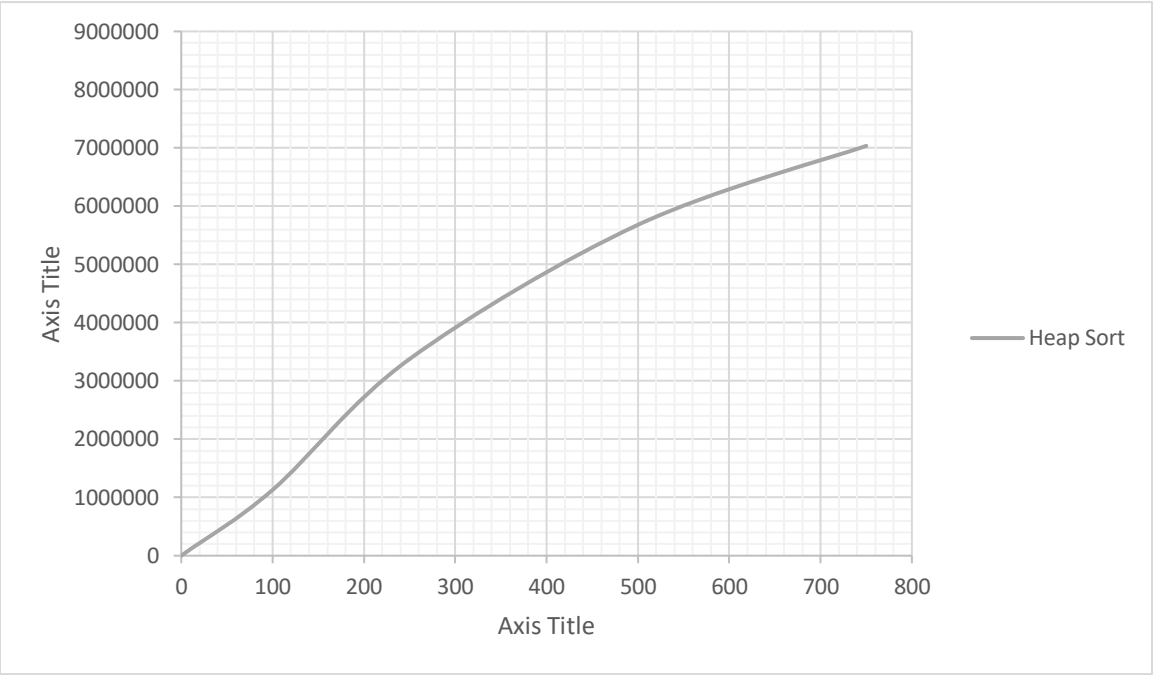
Slow Sort Graph:



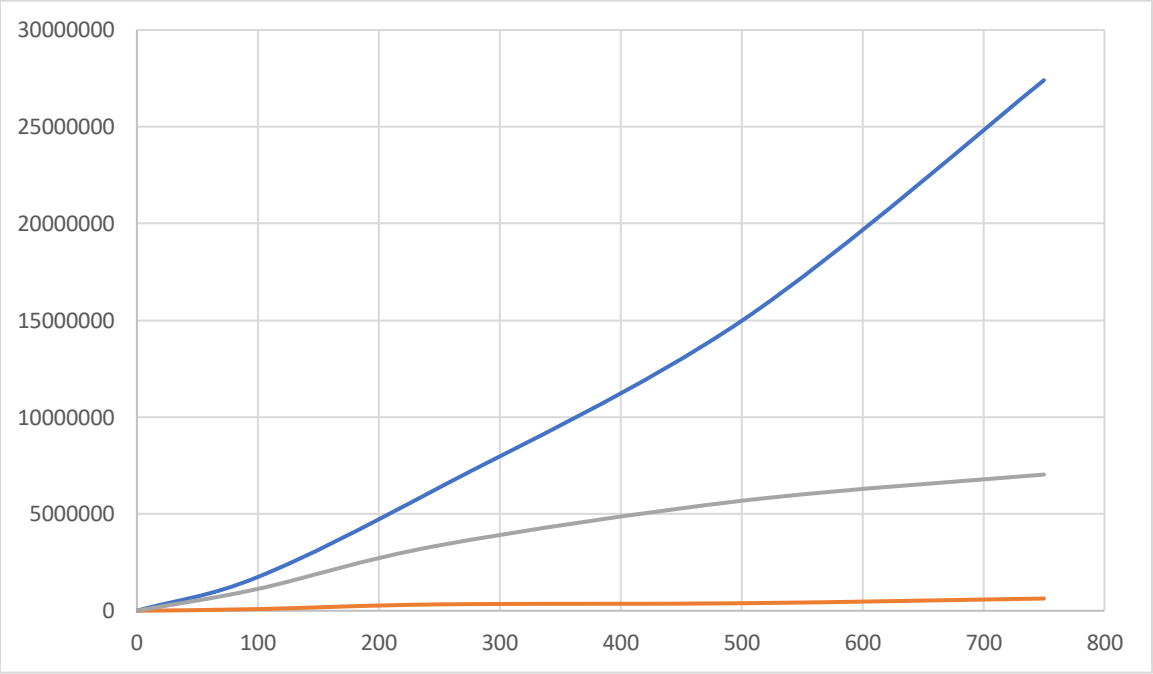
Fast Sort Graph:



Heap Sort Graph:

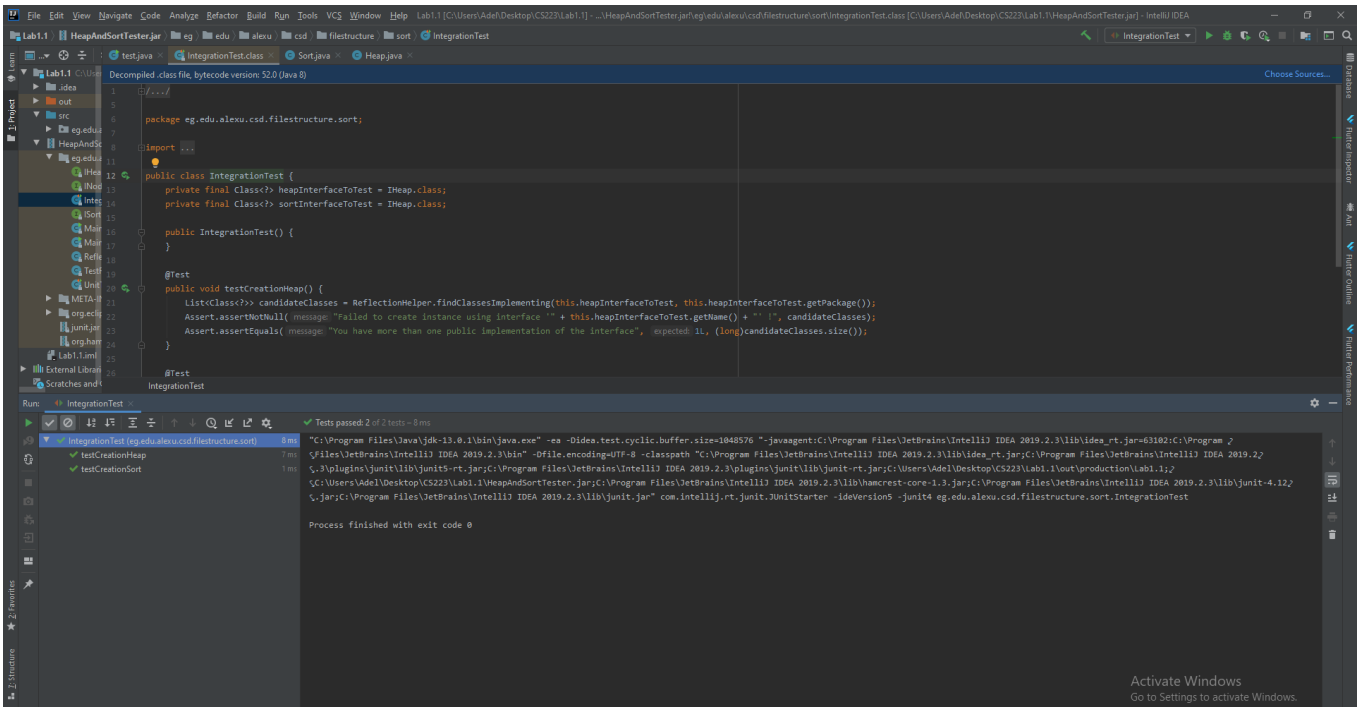


Overall Graph:

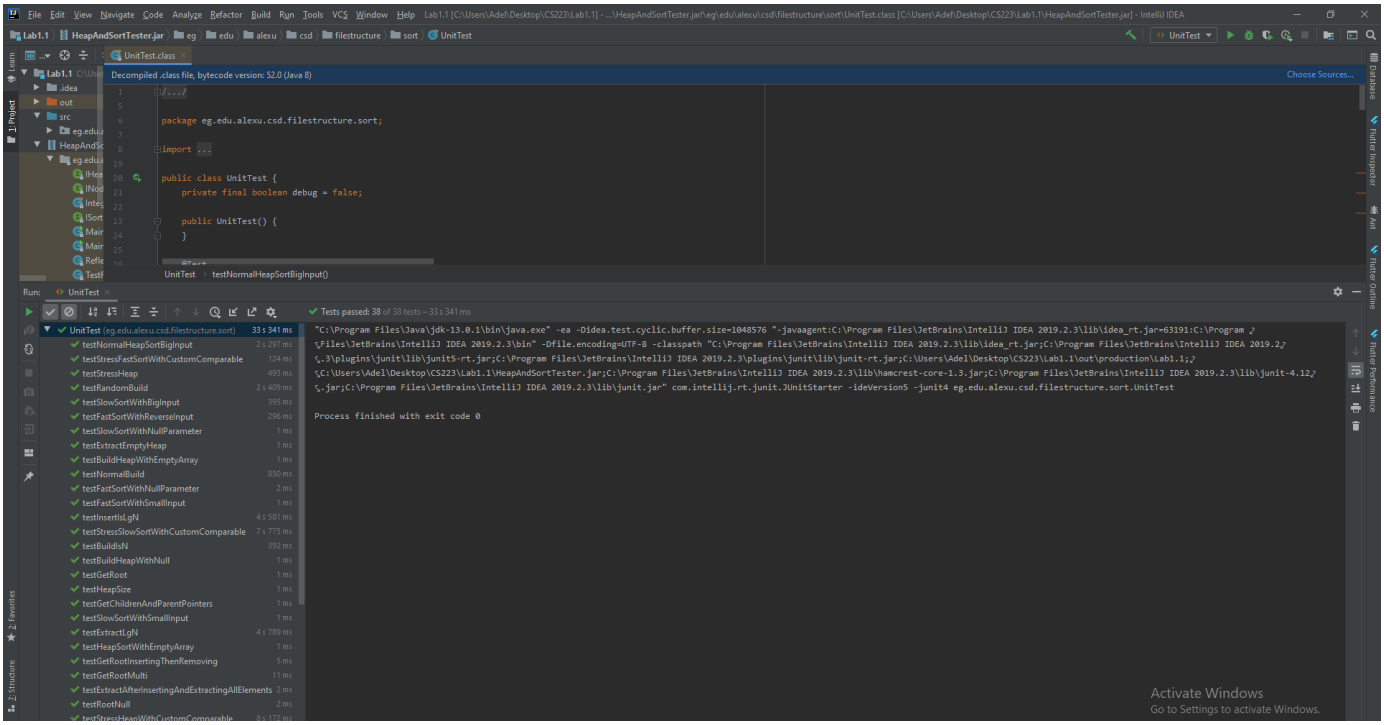


Junit Tests:

1. Integration test:



2. UnitTest:



Students:

- Seif Eldeen Ehab Mostafa Ibrahim - #33
- AbdelRahman Adel AbdelFattah AbdelRaouf Mahmoud - #37