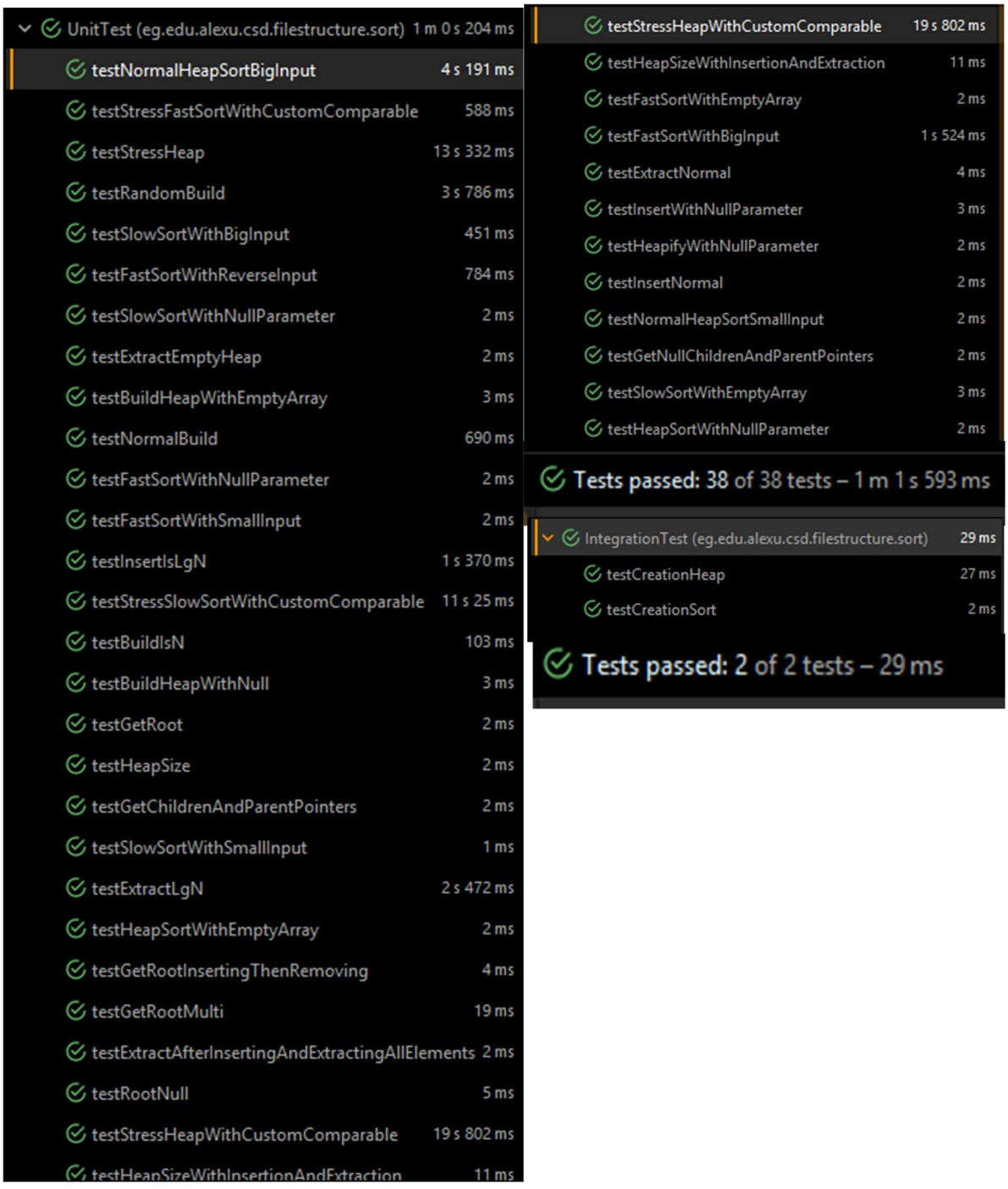
Ahmed Yasser Ahmed 10
Hossam Eldin AbdELGhany 24

Data Structures 2 - Lab 1
Implimenting Binary Heap & Sorting Techniques

Methodology used

We used the array list approach; an array list containg all the elements of the binary heap where each node's children are at *node index *2 +1* or *node index *2 +2* and we used an index to the last element in the array list that is in the heap which is helpful in the sort operation that uses the extract operation each time discarding the last element of the array list, if you repeat this operation n times where n is the size of the array list you end up with a sorted array

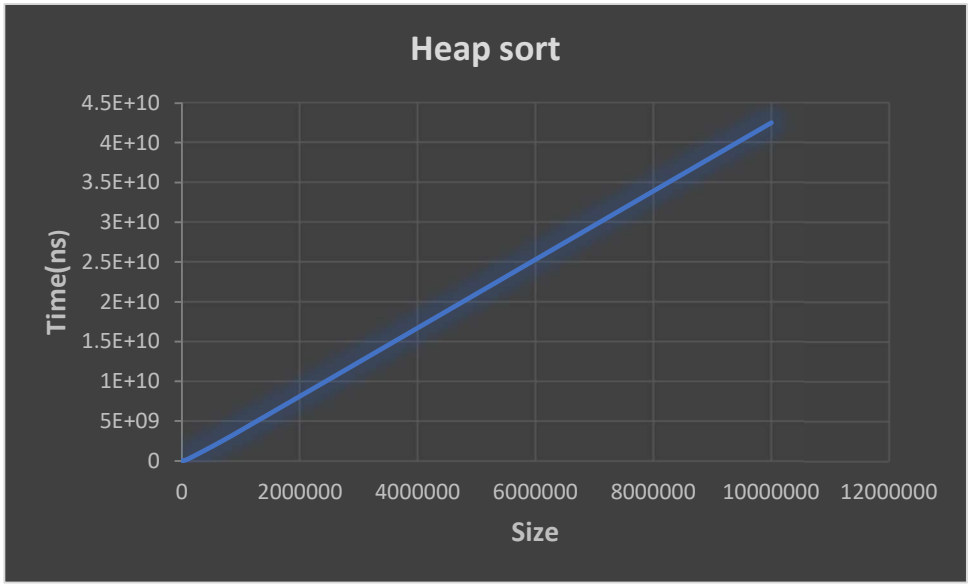
Passing the tests

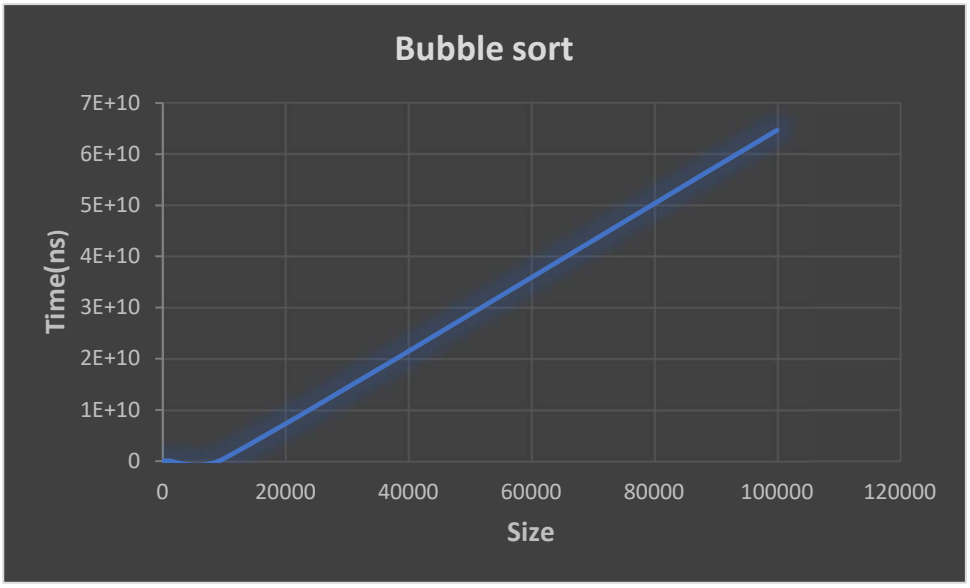


Sorting algorithms analysis:

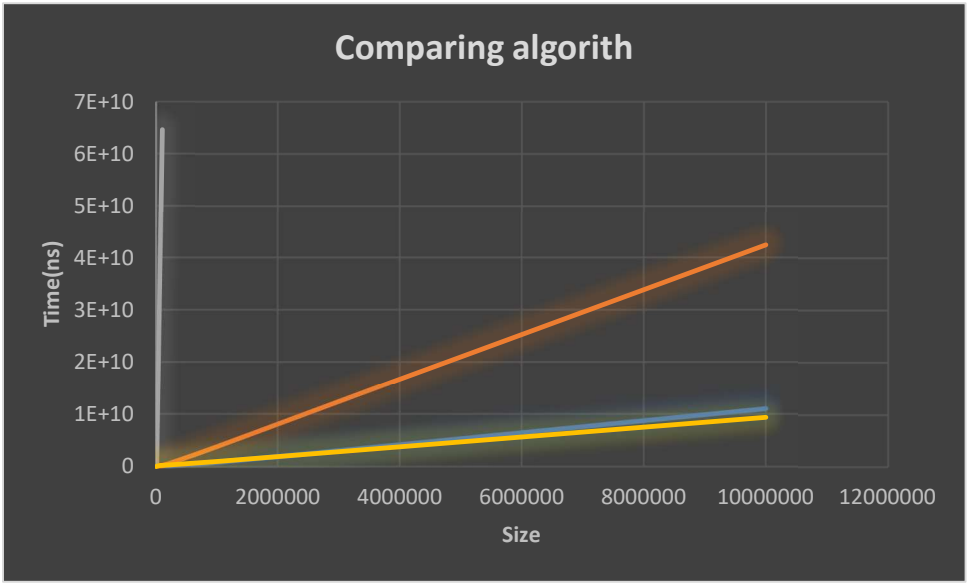
Algorithm\Size	10^1	10^2	10^3	10^4	10^5	10^6	10^7
Collection sort pre-implemented	647116	813599	4344517	26764960	125551973	849487011	11026297440
Heap sort	3977913	6710055	15795340	45496555	232882632	3852129640	42546707831
Bubble sort	155650	4549769	63535421	487102999	64729650756	N/A	N/A
Merge sort	152229	1891176	9910288	38148506	254483205	983590612	9446960121

The following table compares between the time in nanoseconds taken by different sorting algorithms to sort random numbers in different array sizes.


Heap sort
4.5E+10
4E+10
3.5E+10
3E+10
2.5E+10
2E+10
1.5E+10
1E+10
5E+09
0
Time(ns)
0
2000000
4000000
6000000
8000000
10000000
12000000
Size


Bubble sort
7E+10
6E+10
5E+10
4E+10
3E+10
2E+10
1E+10
0
Time(ns)
0
20000
40000
60000
80000
100000
120000
Size


Merge sort
1E+10
9E+09
8E+09
7E+09
6E+09
5E+09
4E+09
3E+09
2E+09
1E+09
0
Time(ns)
0
2000000
4000000
6000000
8000000
10000000
12000000
Size



Lines:

Grey: bubble sort.

Blue: pre-implemented.

Orange: heap sort.

Yellow: merge sort.

Test Code Snippets:

Pre-implemented collection sort:

```
public static void main(String[] args) {
    int i = 7;
    ArrayList<Integer> array = new ArrayList<Integer>();
    Random rand = new Random();
    long iniTime, finTime;

    int size = (int) Math.floor(Math.pow(10, i));
    for (int j = 0; j < size; j++) {
        array.add(rand.nextInt(Integer.MAX_VALUE));
    }
    iniTime = System.nanoTime();
    Collections.sort(array);
    finTime = System.nanoTime();
    System.out.println(finTime - iniTime);
}
```

Heap sort:

```
public static void main(String[] args) {
    int i = 7;
    ArrayList<Integer> array = new ArrayList<Integer>();
    Random rand = new Random();
    ISort<Integer> sorting = new SortingTechniques<Integer>();
    long iniTime, finTime;

    int size = (int) Math.floor(Math.pow(10, i));
    for (int j = 0; j < size; j++) {
        array.add(rand.nextInt(Integer.MAX_VALUE));
    }
    iniTime = System.nanoTime();
    sorting.heapSort(array);
    finTime = System.nanoTime();
    System.out.println(finTime - iniTime);
}
```

To test the slow (bubble) sort and the fast (merge) sort, we change the referred line to (sorting.sortSlow(array) OR sorting.sortFast(array)).To test different ArrayList sizes, we change the variable i.