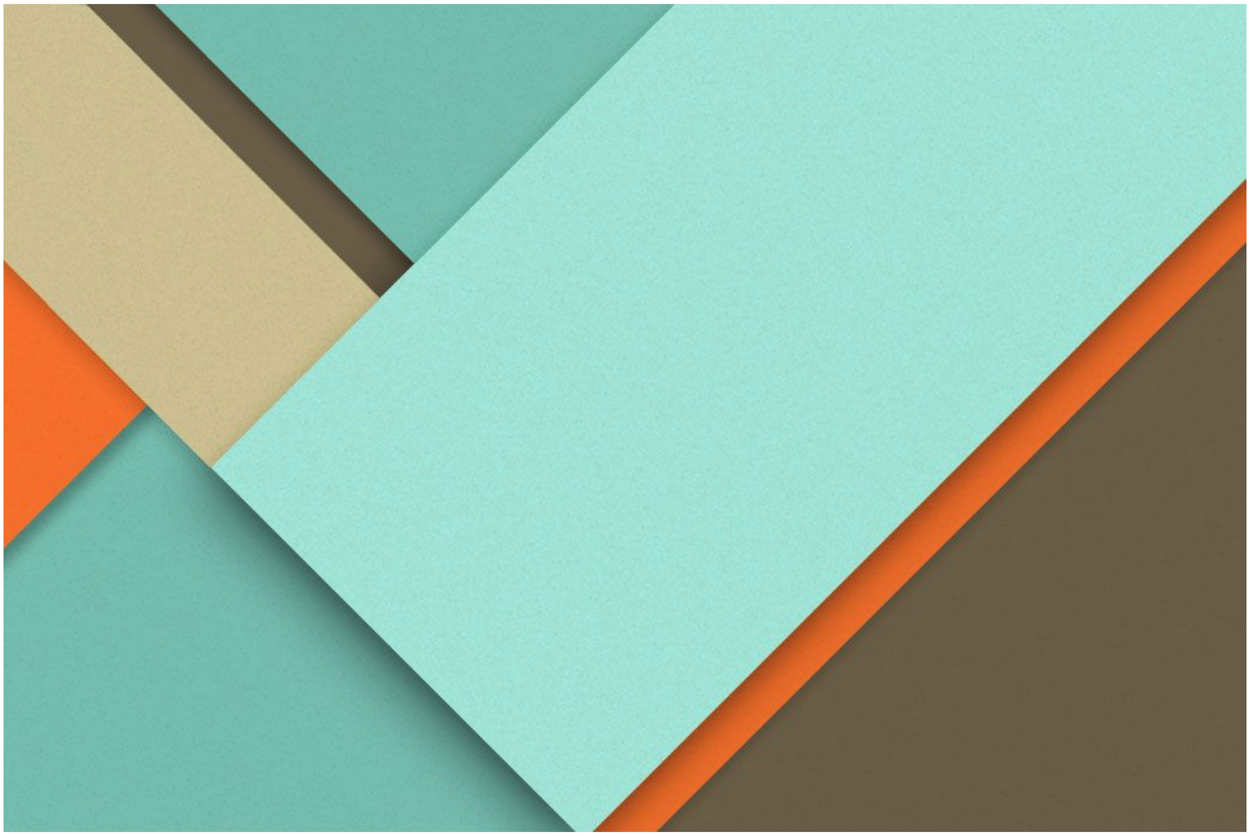


Heap Sort Implementation

09.04.20XX

Ahmad Abdallah Waheeb (04)

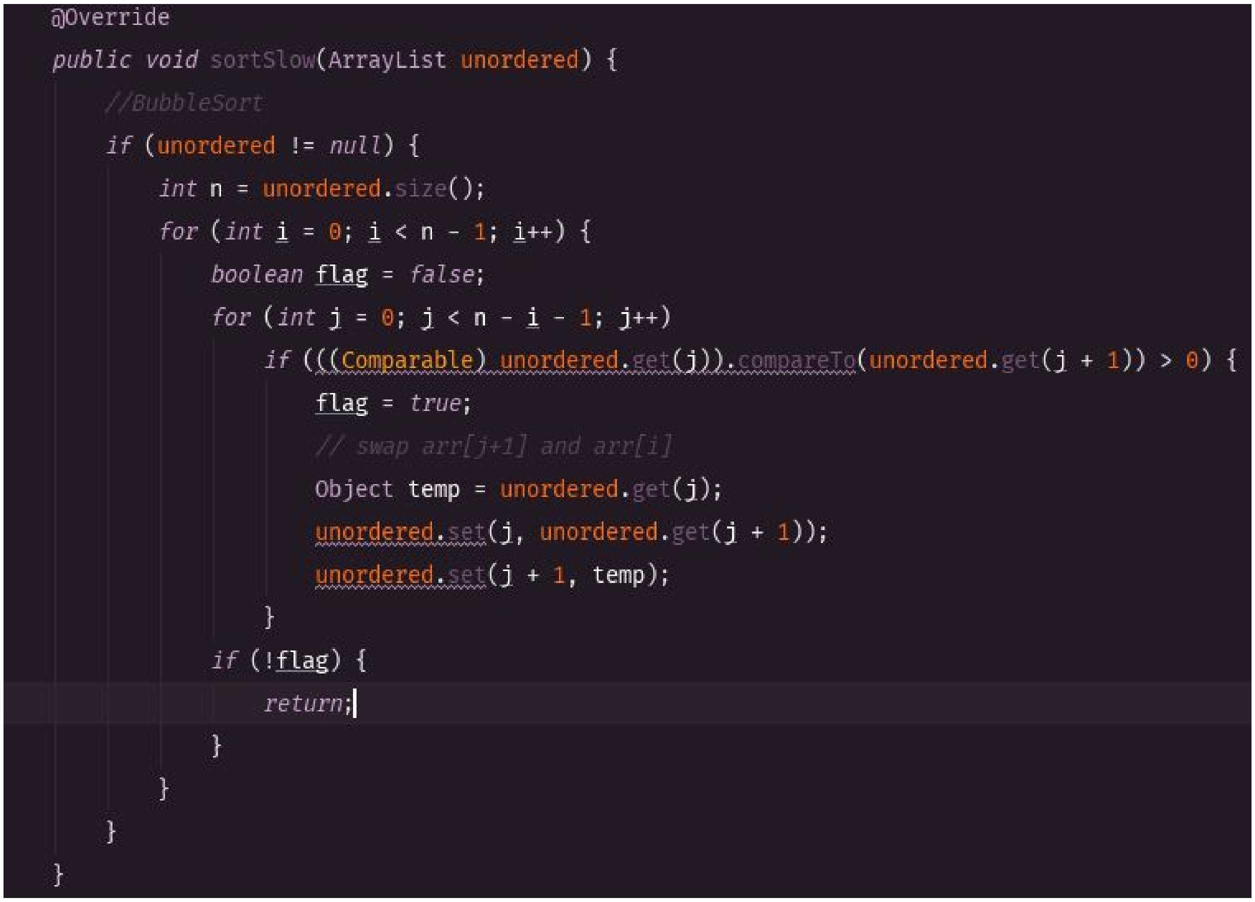
Ahmed Mohamed El Zeny (06)

Problem Statement

Implementing the Heap. Then using it to implement the Heap Sort. Also implementing a Fast Sort method O(nlgn) and a Slow Sort method O(n*n).

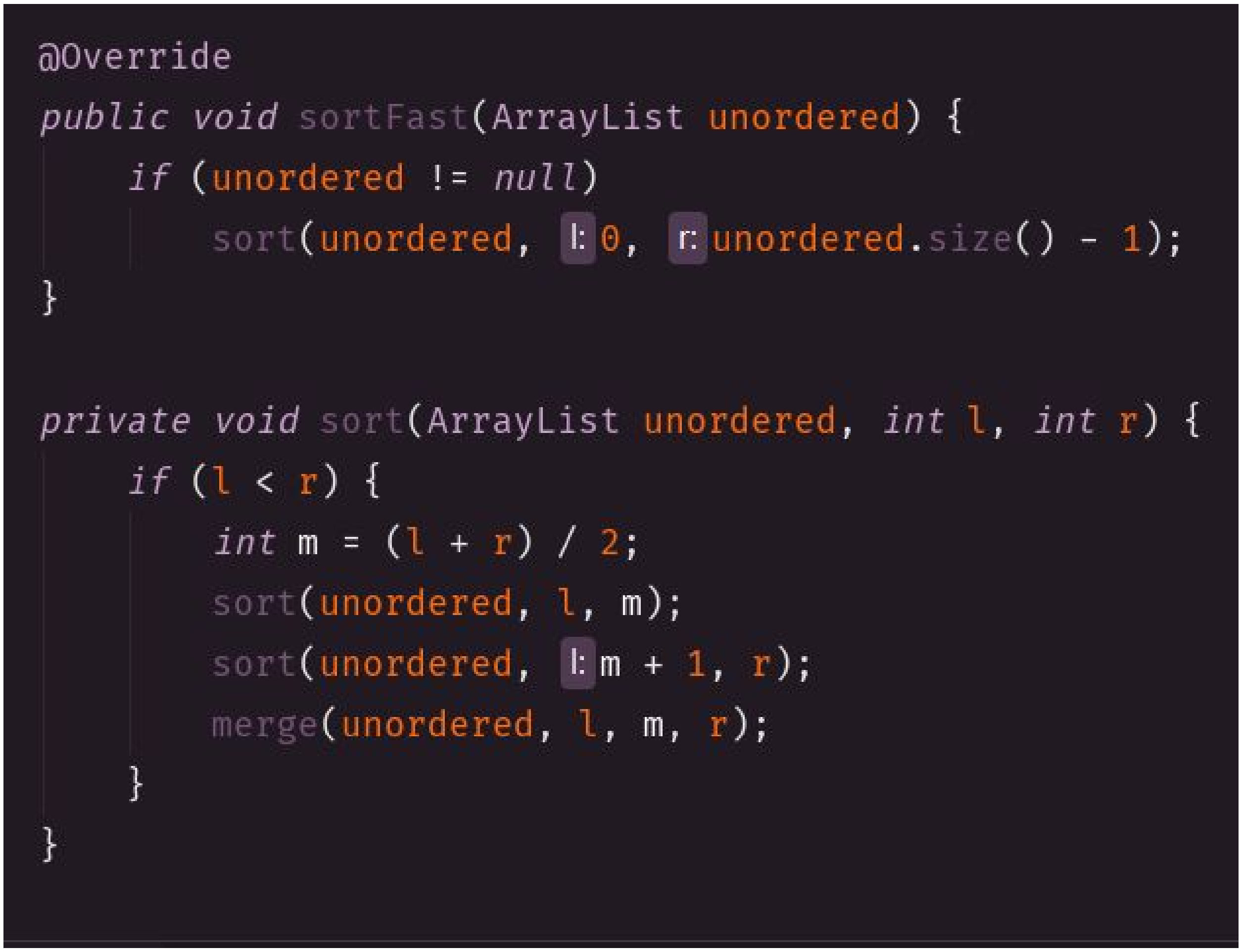
Code Snippets

sortSlow method (BubbleSort)

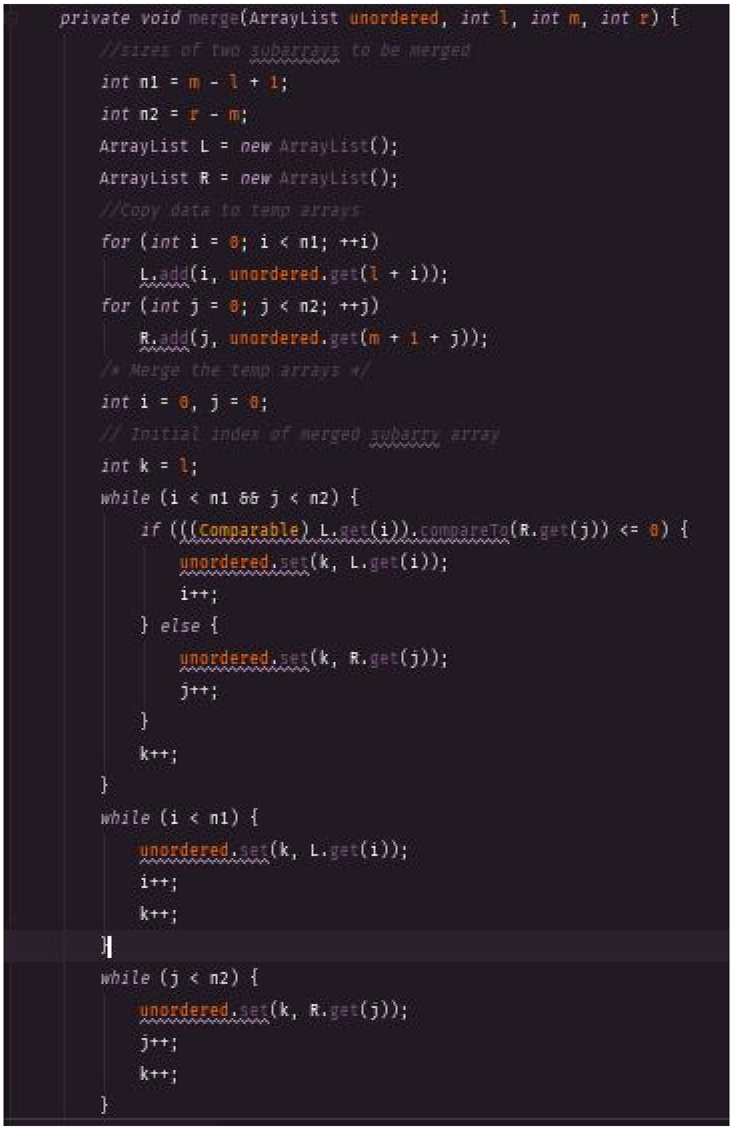


```
@Override
public void sortSlow(ArrayList unordered) {
    //BubbleSort
    if (unordered != null) {
        int n = unordered.size();
        for (int i = 0; i < n - 1; i++) {
            boolean flag = false;
            for (int j = 0; j < n - i - 1; j++)
                if (((Comparable) unordered.get(j)).compareTo(unordered.get(j + 1)) > 0) {
                    flag = true;
                    // swap arr[j+1] and arr[i]
                    Object temp = unordered.get(j);
                    unordered.set(j, unordered.get(j + 1));
                    unordered.set(j + 1, temp);
                }
            if (!flag) {
                return;
            }
        }
    }
}
```

sortFast method (MergeSort)

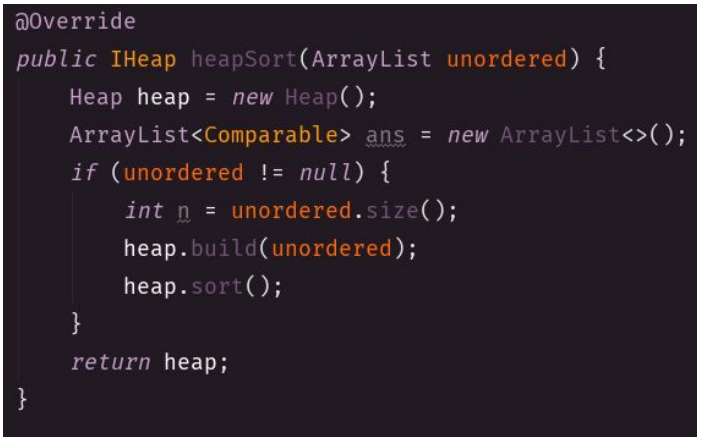


```
@Override
public void sortFast(ArrayList unordered) {
    if (unordered != null)
        sort(unordered, l: 0, r: unordered.size() - 1);
}

private void sort(ArrayList unordered, int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        sort(unordered, l, m);
        sort(unordered, l: m + 1, r);
        merge(unordered, l, m, r);
    }
}
```

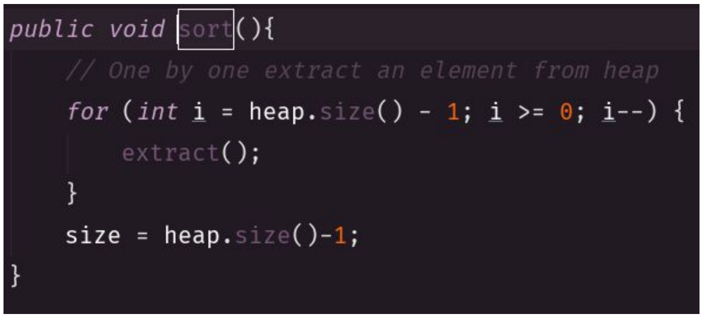


```
private void merge(ArrayList unordered, int l, int m, int r) {
    //sizes of two subarrays to be merged
    int n1 = m - l + 1;
    int n2 = r - m;
    ArrayList L = new ArrayList();
    ArrayList R = new ArrayList();
    //Copy data to temp arrays
    for (int i = 0; i < n1; ++i)
        L.add(i, unordered.get(l + i));
    for (int j = 0; j < n2; ++j)
        R.add(j, unordered.get(m + 1 + j));
    /* Merge the temp arrays */
    int i = 0, j = 0;
    // Initial index of merged subarry array
    int k = l;
    while (i < n1 && j < n2) {
        if (((Comparable) L.get(i)).compareTo(R.get(j)) <= 0) {
            unordered.set(k, L.get(i));
            i++;
        } else {
            unordered.set(k, R.get(j));
            j++;
        }
        k++;
    }
    while (i < n1) {
        unordered.set(k, L.get(i));
        i++;
        k++;
    }
    while (j < n2) {
        unordered.set(k, R.get(j));
        j++;
        k++;
    }
```

HeapSort



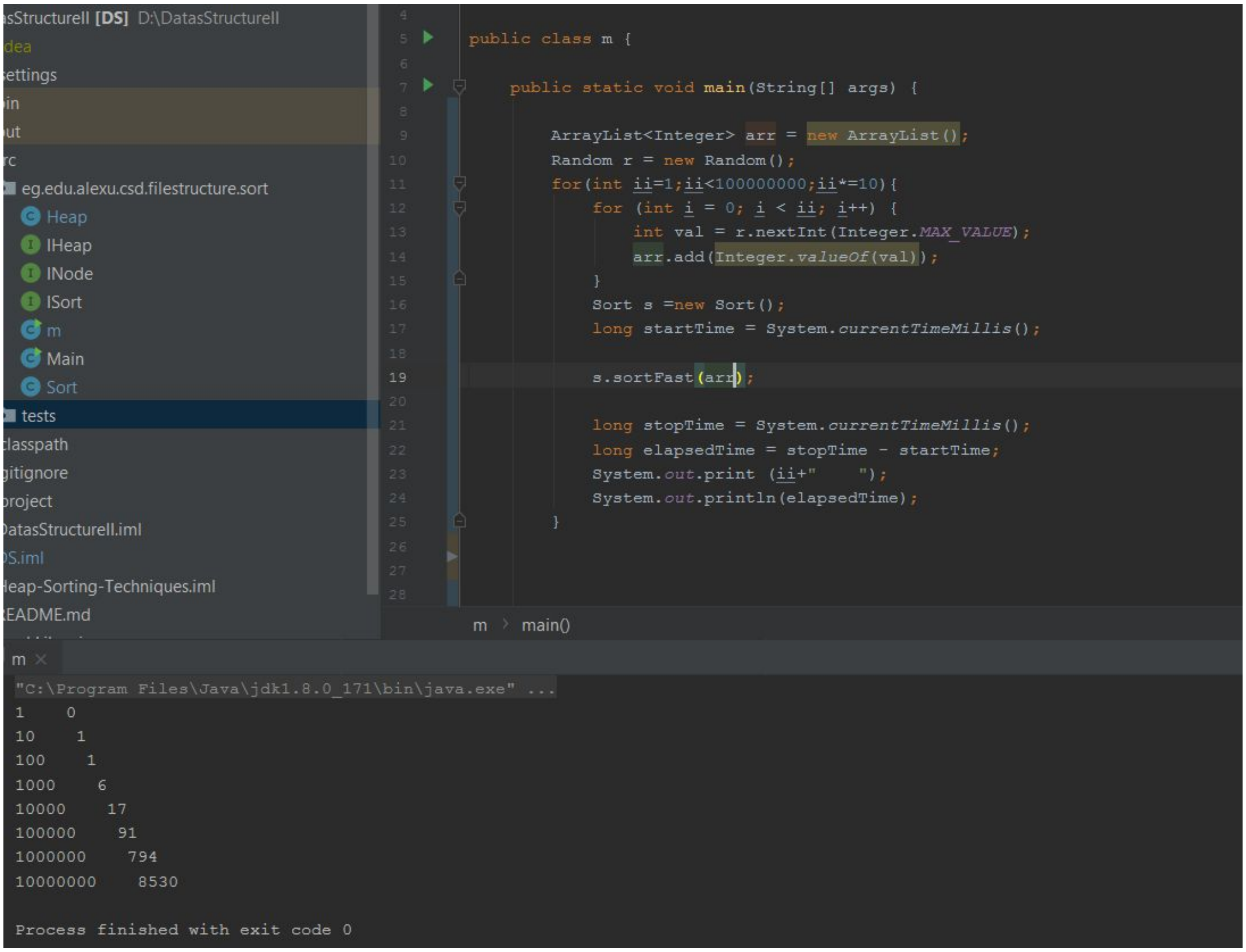
```
@Override
public IHeap heapSort(ArrayList unordered) {
    Heap heap = new Heap();
    ArrayList<Comparable> ans = new ArrayList<>();
    if (unordered != null) {
        int n = unordered.size();
        heap.build(unordered);
        heap.sort();
    }
    return heap;
}
```



```
public void sort(){
    // One by one extract an element from heap
    for (int i = heap.size() - 1; i >= 0; i--) {
        extract();
    }
    size = heap.size()-1;
}
```

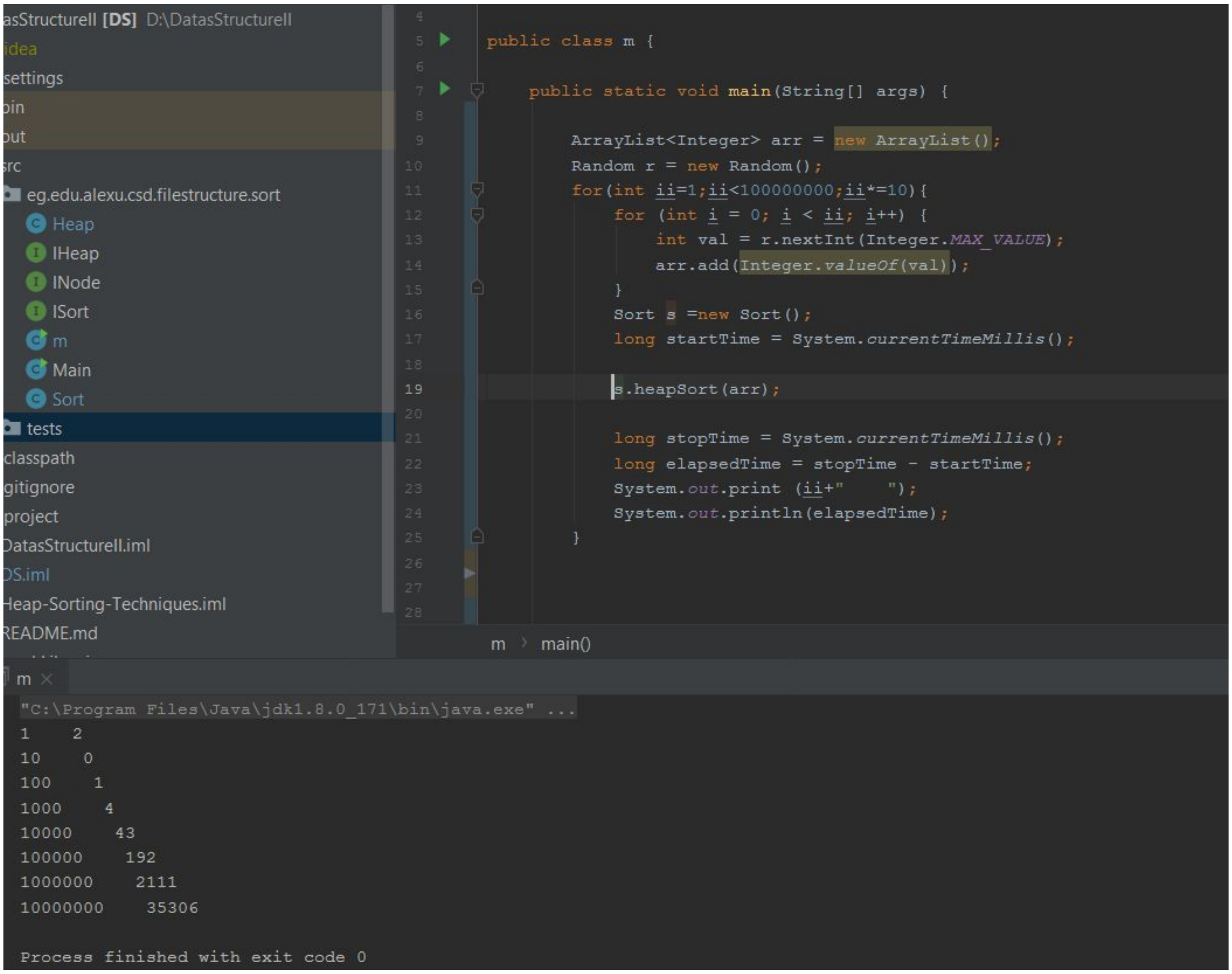
Sort Analysis

Functions:



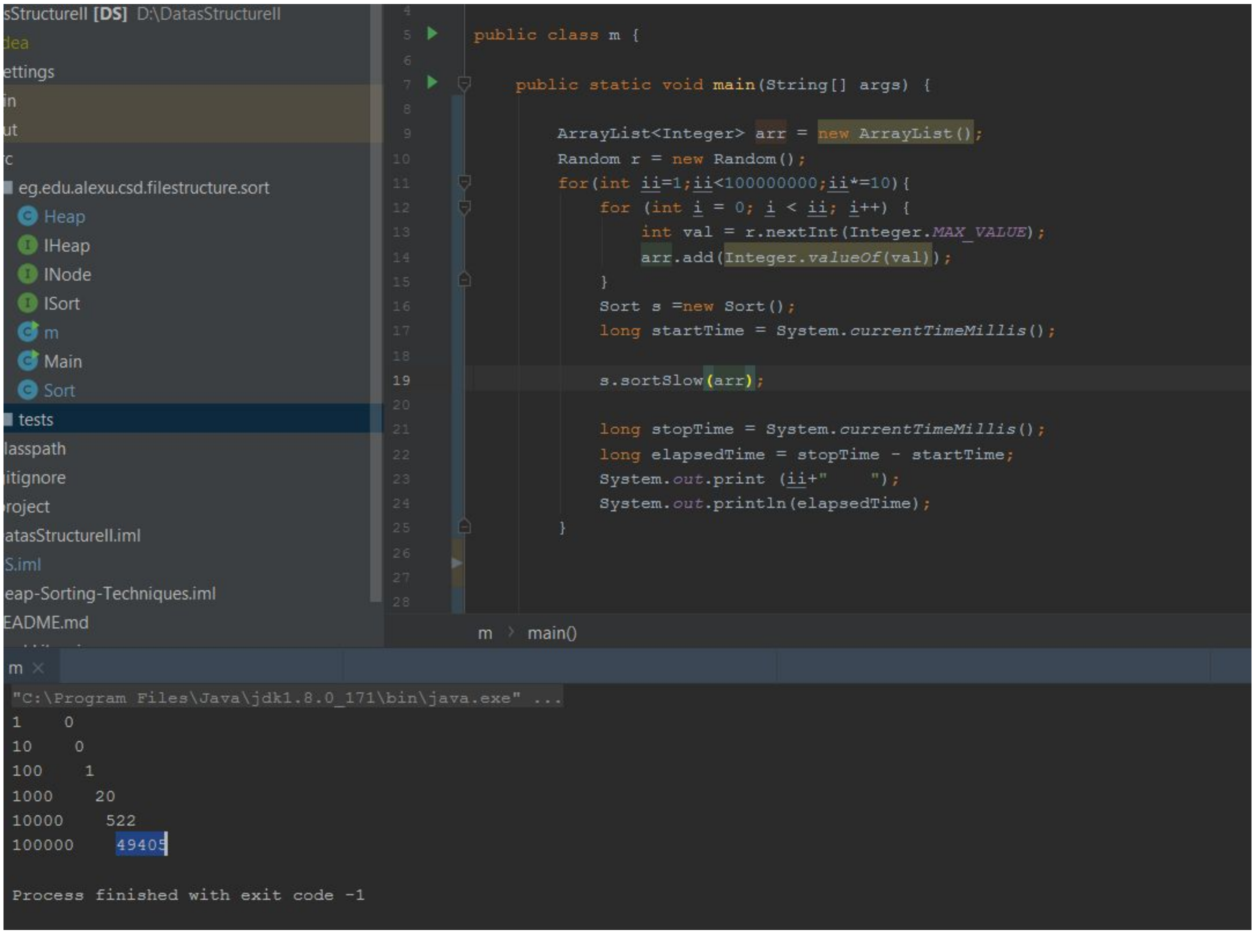
```
public class m {

    public static void main(String[] args) {

        ArrayList<Integer> arr = new ArrayList();
        Random r = new Random();
        for(int ii=1;ii<100000000;ii*=10){
            for (int i = 0; i < ii; i++) {
                int val = r.nextInt(Integer.MAX_VALUE);
                arr.add(Integer.valueOf(val));
            }
            Sort s =new Sort();
            long startTime = System.currentTimeMillis();

            s.sortFast(arr);

            long stopTime = System.currentTimeMillis();
            long elapsedTime = stopTime - startTime;
            System.out.print (ii+"     ");
            System.out.println(elapsedTime);
        }
```

```
"C:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...
1     0
10     1
100     1
1000     6
10000     17
100000     91
1000000     794
10000000     8530

Process finished with exit code 0
```



```
public class m {

    public static void main(String[] args) {

        ArrayList<Integer> arr = new ArrayList();
        Random r = new Random();
        for(int ii=1;ii<100000000;ii*=10){
            for (int i = 0; i < ii; i++) {
                int val = r.nextInt(Integer.MAX_VALUE);
                arr.add(Integer.valueOf(val));
            }
            Sort s =new Sort();
            long startTime = System.currentTimeMillis();

            s.heapSort(arr);

            long stopTime = System.currentTimeMillis();
            long elapsedTime = stopTime - startTime;
            System.out.print (ii+"     ");
            System.out.println(elapsedTime);
        }
```

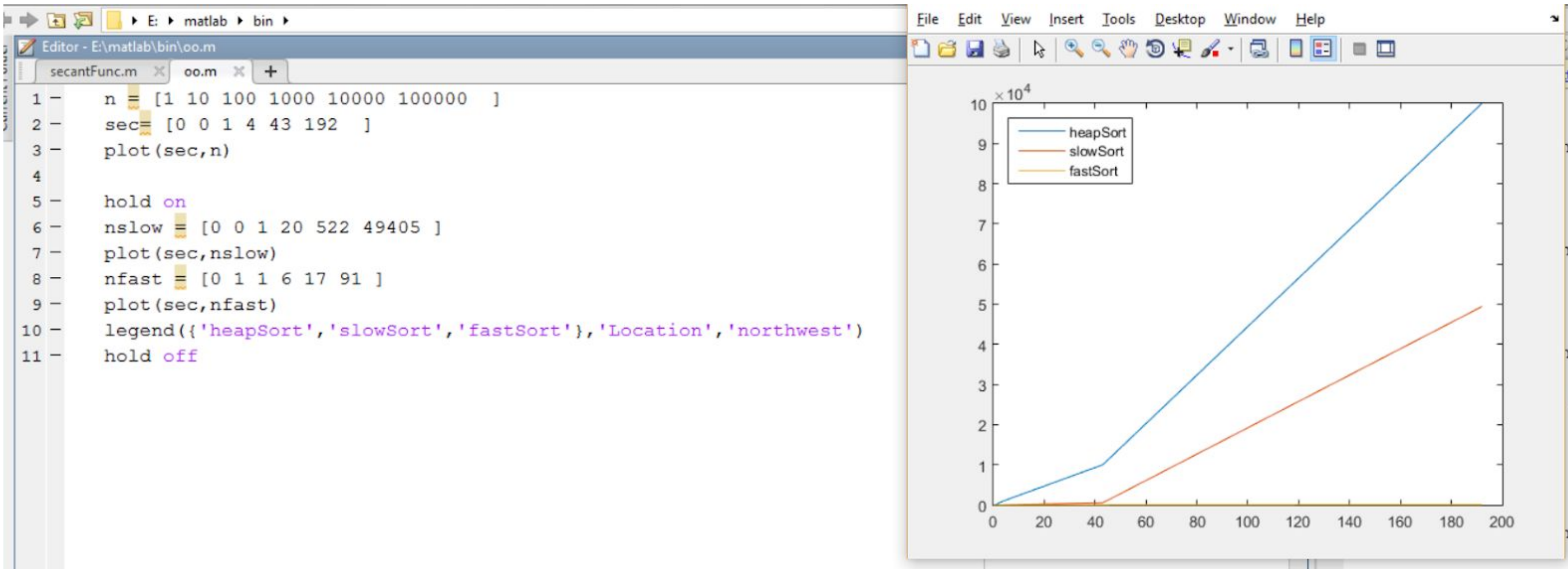
```
"C:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...
1     2
10     0
100     1
1000     4
10000     43
100000     192
1000000     2111
10000000     35306

Process finished with exit code 0
```



```
public class m {

    public static void main(String[] args) {

        ArrayList<Integer> arr = new ArrayList();
        Random r = new Random();
        for(int ii=1;ii<100000000;ii*=10){
            for (int i = 0; i < ii; i++) {
                int val = r.nextInt(Integer.MAX_VALUE);
                arr.add(Integer.valueOf(val));
            }
            Sort s =new Sort();
            long startTime = System.currentTimeMillis();

            s.sortSlow(arr);

            long stopTime = System.currentTimeMillis();
            long elapsedTime = stopTime - startTime;
            System.out.print (ii+"     ");
            System.out.println(elapsedTime);
        }
```

```
"C:\Program Files\Java\jdk1.8.0_171\bin\java.exe" ...
1     0
10     0
100     1
1000     20
10000     522
100000     49405

Process finished with exit code -1
```

Plotting results:



```
n = [1 10 100 1000 10000 100000  ]
sec= [0 0 1 4 43 192  ]
plot(sec,n)

hold on
nslow = [0 0 1 20 522 49405 ]
plot(sec,nslow)
nfast = [0 1 1 6 17 91 ]
plot(sec,nfast)
legend({'heapSort','slowSort','fastSort'},'Location','northwest')
hold off
```

Repo and code:

https://github.com/ahmedezeny/Heap-Sorting-Techniques