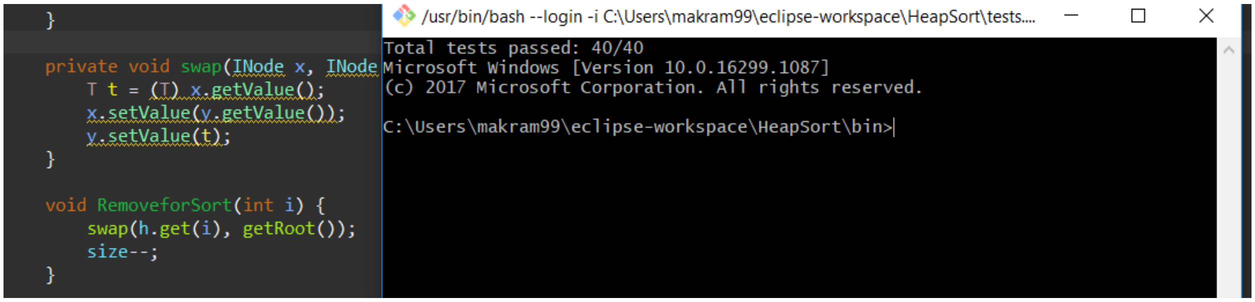
NAME/ Makrm William. ID/64.

Lab 1

For tests in jar in the lab file:

By using script sh.file

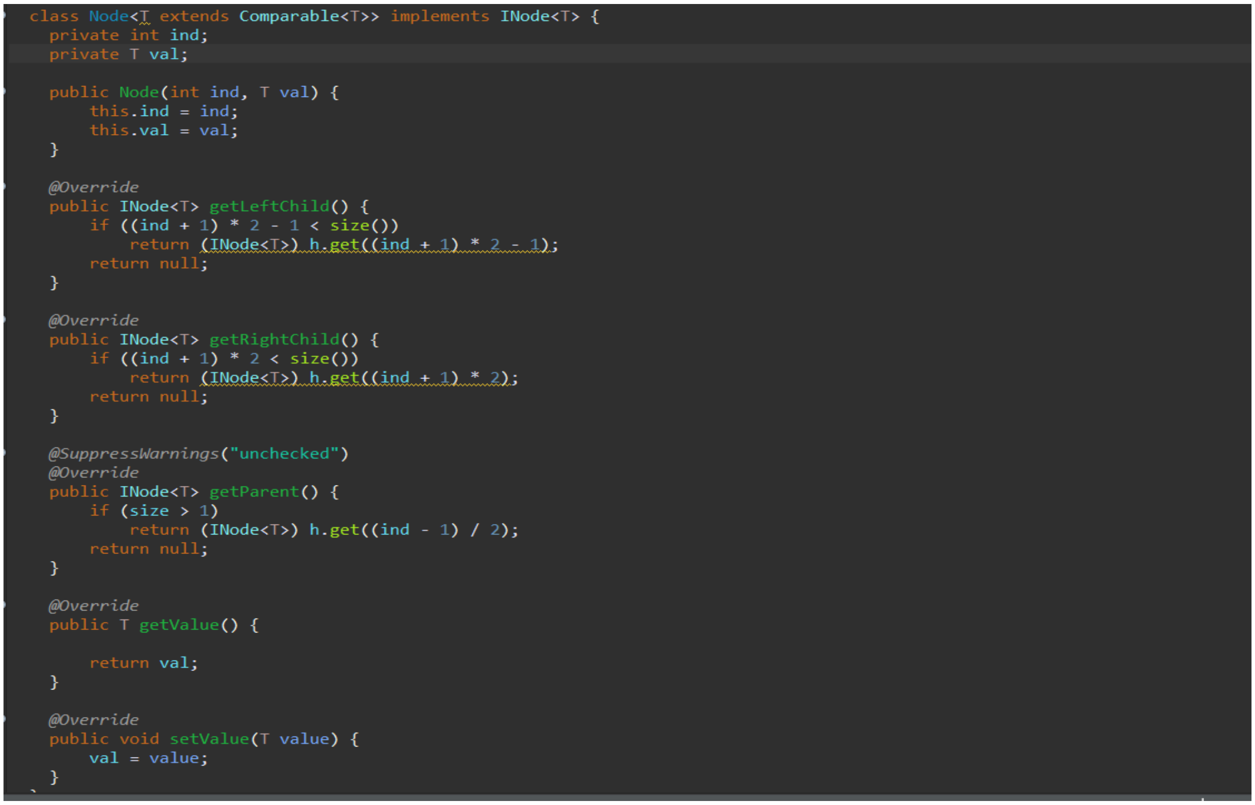


```
    }

    private void swap(INode x, INode
        T t = (T) x.getValue();
        x.setValue(y.getValue());
        y.setValue(t);
    }

    void RemoveforSort(int i) {
        swap(h.get(i), getRoot());
        size--;
    }
```

/usr/bin/bash --login -i C:\Users\makram99\eclipse-workspace\HeapSort\tests....

```
Total tests passed: 40/40
Microsoft Windows [Version 10.0.16299.1087]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\makram99\eclipse-workspace\HeapSort\bin>
```

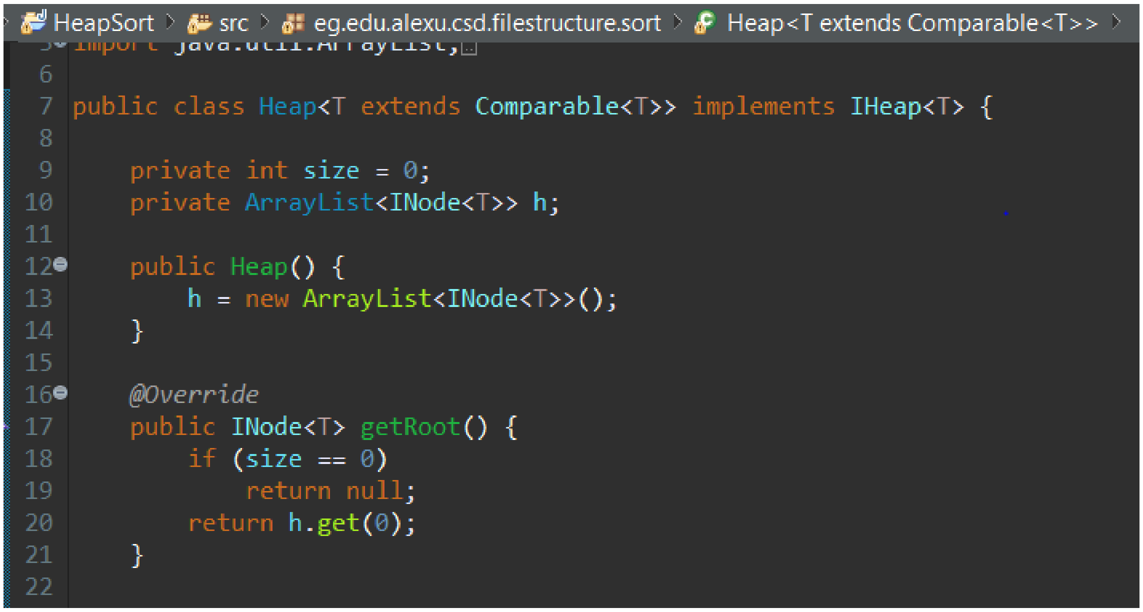
Screens from the code & for explain:



```
class Node<T extends Comparable<T>> implements INode<T> {
    private int ind;
    private T val;

    public Node(int ind, T val) {
        this.ind = ind;
        this.val = val;
    }

    @Override
    public INode<T> getLeftChild() {
        if ((ind + 1) * 2 - 1 < size())
            return (INode<T>) h.get((ind + 1) * 2 - 1);
        return null;
    }

    @Override
    public INode<T> getRightChild() {
        if ((ind + 1) * 2 < size())
            return (INode<T>) h.get((ind + 1) * 2);
        return null;
    }

    @SuppressWarnings("unchecked")
    @Override
    public INode<T> getParent() {
        if (size > 1)
            return (INode<T>) h.get((ind - 1) / 2);
        return null;
    }

    @Override
    public T getValue() {

        return val;
    }

    @Override
    public void setValue(T value) {
        val = value;
    }
```

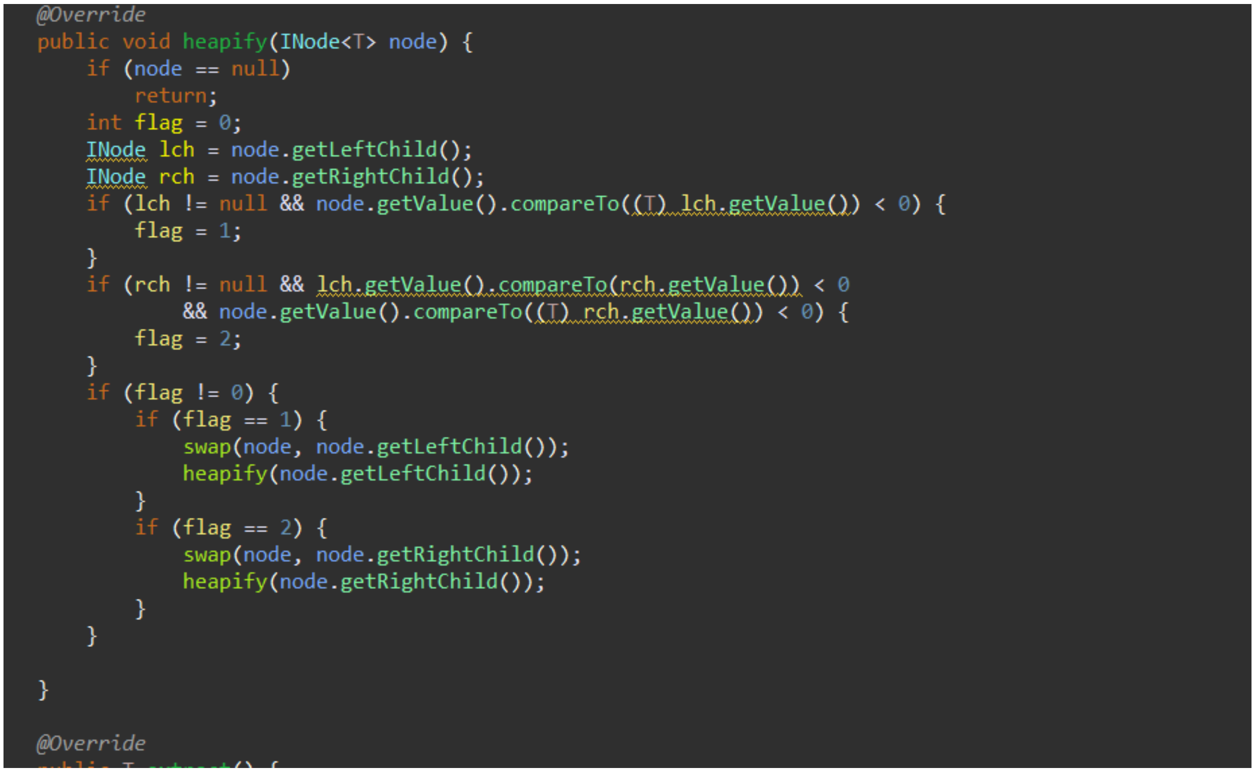
Here the class Node is inner class in the heap class to use the arraylist directly in it.

I use 0 index for the array so equation looks some different.


HeapSort > src > eg.edu.alexu.csd.filestructure.sort > Heap<T extends Comparable<T>> >

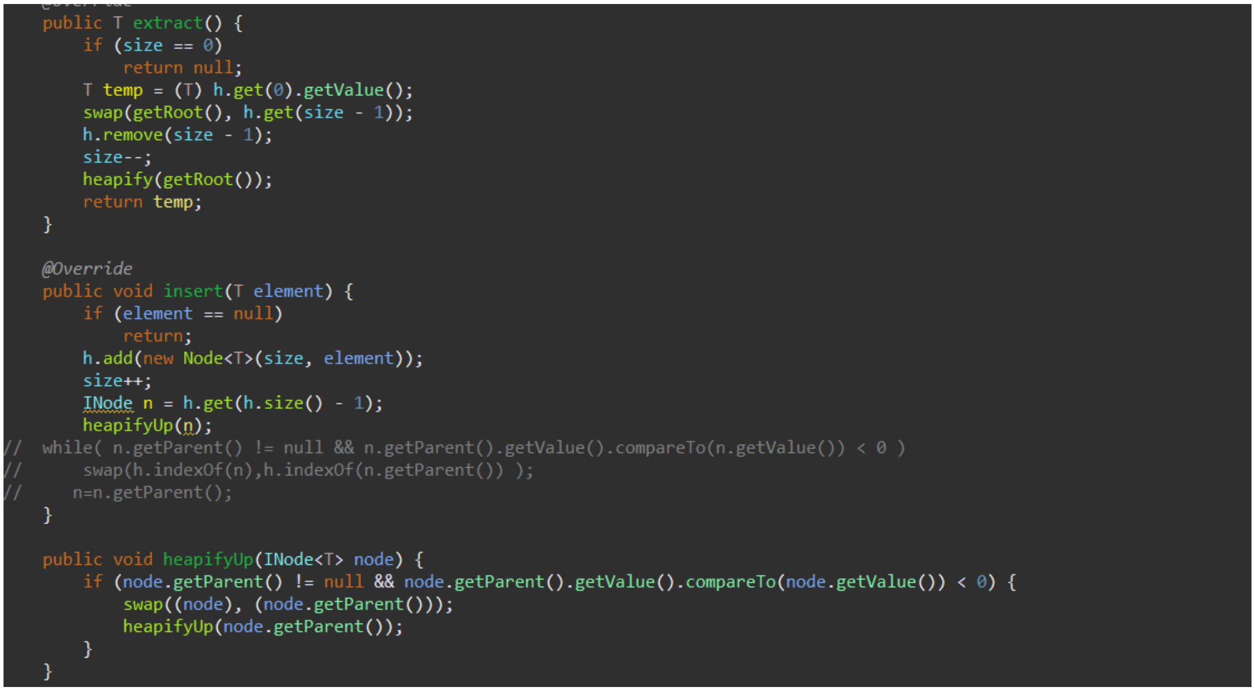
```java
import java.util.ArrayList;

public class Heap<T extends Comparable<T>> implements IHeap<T> {

    private int size = 0;
    private ArrayList<INode<T>> h;

    public Heap() {
        h = new ArrayList<INode<T>>();
    }

    @Override
    public INode<T> getRoot() {
        if (size == 0)
            return null;
        return h.get(0);
    }

```

In heap class it has only two attributes for all methods which size & h.



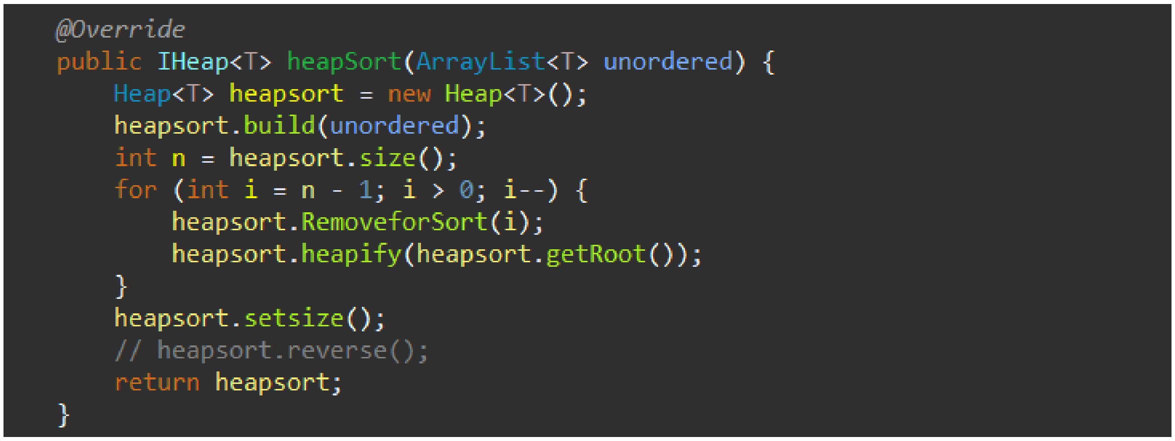
```java
@Override
public void heapify(INode<T> node) {
    if (node == null)
        return;
    int flag = 0;
    INode lch = node.getLeftChild();
    INode rch = node.getRightChild();
    if (lch != null && node.getValue().compareTo((T) lch.getValue()) < 0) {
        flag = 1;
    }
    if (rch != null && lch.getValue().compareTo(rch.getValue()) < 0
            && node.getValue().compareTo((T) rch.getValue()) < 0) {
        flag = 2;
    }
    if (flag != 0) {
        if (flag == 1) {
            swap(node, node.getLeftChild());
            heapify(node.getLeftChild());
        }
        if (flag == 2) {
            swap(node, node.getRightChild());
            heapify(node.getRightChild());
        }
    }

}

@Override
public T extract() {
```

in heapify method, flag is to determine the position of largest node.



```
    public T extract() {
        if (size == 0)
            return null;
        T temp = (T) h.get(0).getValue();
        swap(getRoot(), h.get(size - 1));
        h.remove(size - 1);
        size--;
        heapify(getRoot());
        return temp;
    }

    @Override
    public void insert(T element) {
        if (element == null)
            return;
        h.add(new Node<T>(size, element));
        size++;
        INode n = h.get(h.size() - 1);
        heapifyUp(n);
//  while( n.getParent() != null && n.getParent().getValue().compareTo(n.getValue()) < 0 )
//      swap(h.indexOf(n),h.indexOf(n.getParent()) );
//     n=n.getParent();
    }

    public void heapifyUp(INode<T> node) {
        if (node.getParent() != null && node.getParent().getValue().compareTo(node.getValue()) < 0) {
            swap((node), (node.getParent()));
            heapifyUp(node.getParent());
        }
    }
```

In insert method, it uses a recursion method heapifyUp to check parent values and swap if they are smaller.

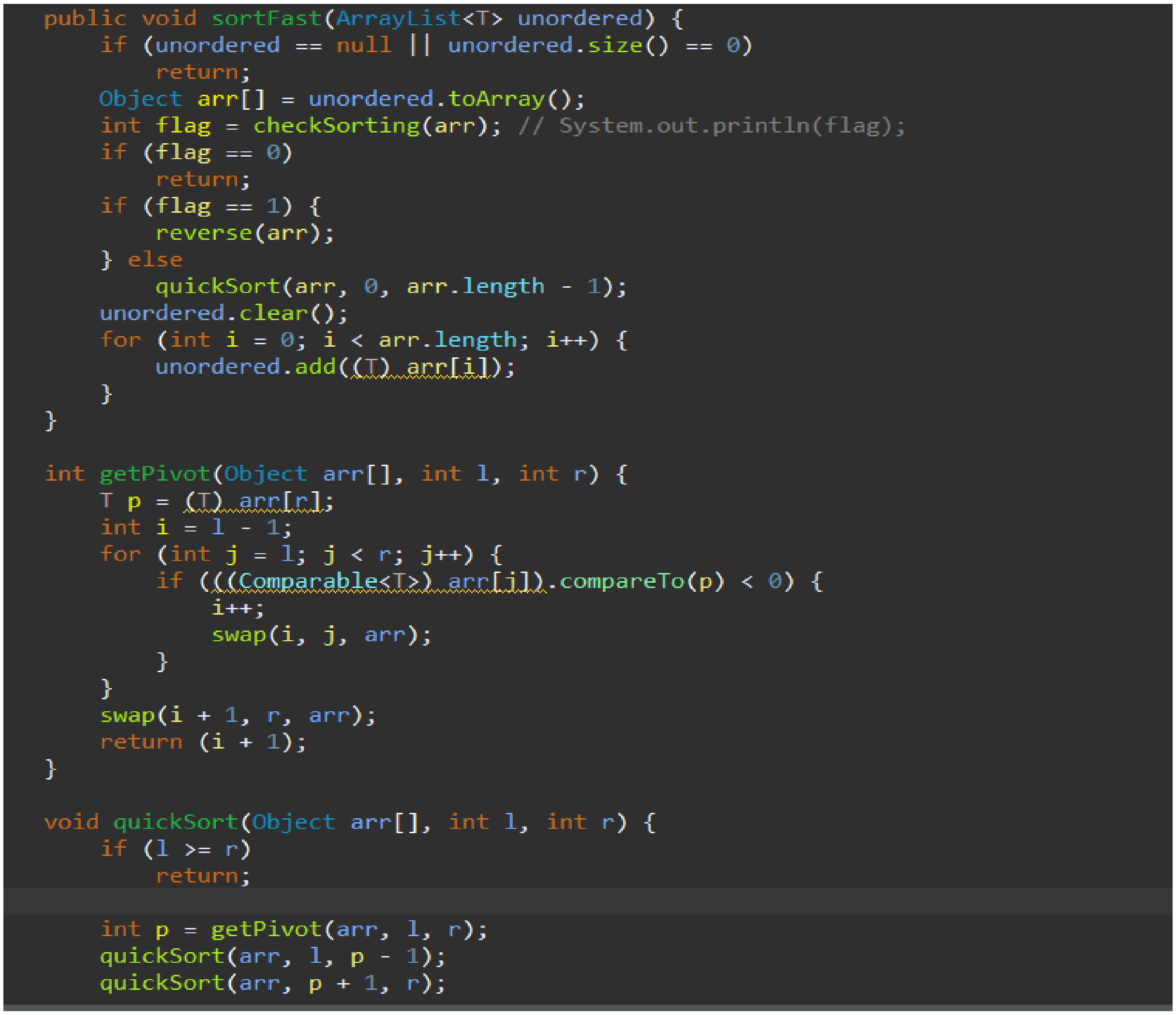


```
 @Override
 public IHeap<T> heapSort(ArrayList<T> unordered) {
     Heap<T> heapsort = new Heap<T>();
     heapsort.build(unordered);
     int n = heapsort.size();
     for (int i = n - 1; i > 0; i--) {
         heapsort.RemoveforSort(i);
         heapsort.heapify(heapsort.getRoot());
     }
     heapsort.setsize();
     // heapsort.reverse();
     return heapsort;
 }
```

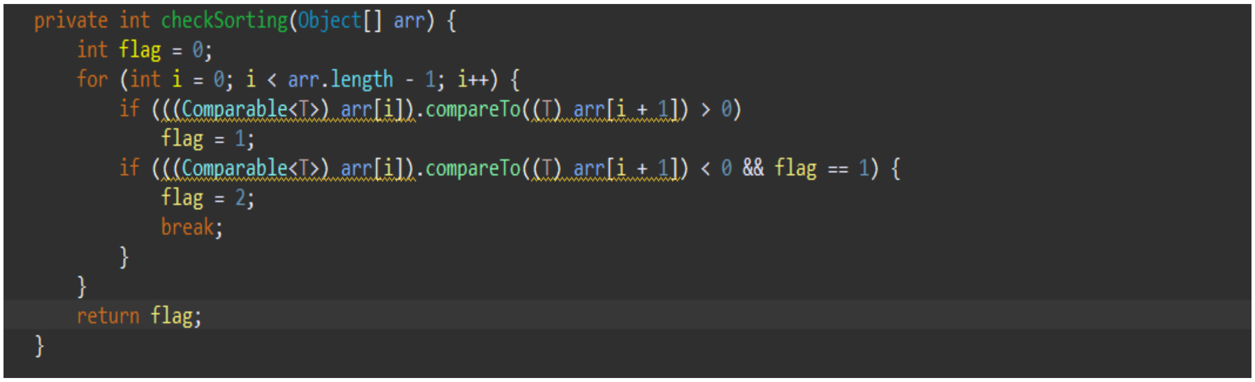
In heap sort to sort it inplace ,it’s swap root to end of array and minus its size. at end its back size to its original.

In slow sort using Bubble sort and check for sorting in each loop to get the best case O(n).

In fast sort using quicksort and use a check method for sorting array to avoid worst case O(n^2).



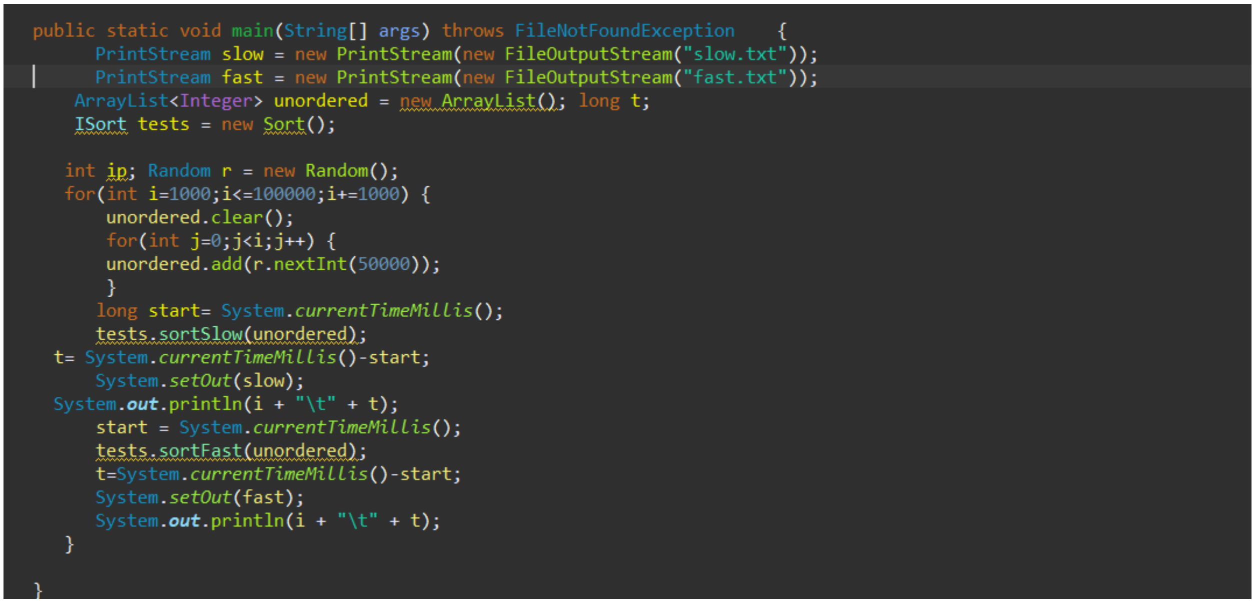
```
public void sortFast(ArrayList<T> unordered) {
    if (unordered == null || unordered.size() == 0)
        return;
    Object arr[] = unordered.toArray();
    int flag = checkSorting(arr); // System.out.println(flag);
    if (flag == 0)
        return;
    if (flag == 1) {
        reverse(arr);
    } else
        quickSort(arr, 0, arr.length - 1);
    unordered.clear();
    for (int i = 0; i < arr.length; i++) {
        unordered.add((T) arr[i]);
    }
}

int getPivot(Object arr[], int l, int r) {
    T p = (T) arr[r];
    int i = l - 1;
    for (int j = l; j < r; j++) {
        if (((Comparable<T>) arr[j]).compareTo(p) < 0) {
            i++;
            swap(i, j, arr);
        }
    }
    swap(i + 1, r, arr);
    return (i + 1);
}

void quickSort(Object arr[], int l, int r) {
    if (l >= r)
        return;

    int p = getPivot(arr, l, r);
    quickSort(arr, l, p - 1);
    quickSort(arr, p + 1, r);
```



```
private int checkSorting(Object[] arr) {
    int flag = 0;
    for (int i = 0; i < arr.length - 1; i++) {
        if (((Comparable<T>) arr[i]).compareTo((T) arr[i + 1]) > 0)
            flag = 1;
        if (((Comparable<T>) arr[i]).compareTo((T) arr[i + 1]) < 0 && flag == 1) {
            flag = 2;
            break;
        }
    }
    return flag;
}
```

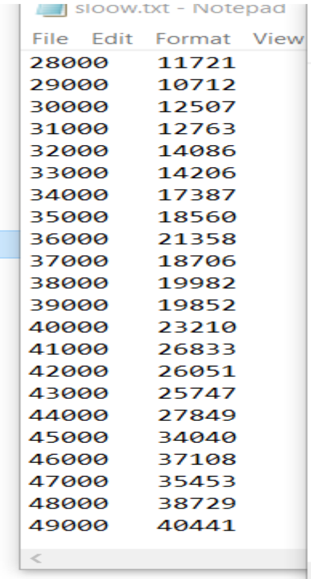
To test your implementation and analyze the running time performance, to generate a dataset of random numbers and plot the relationship between the execution time of the sorting algorithm versus the input size.

the class testing using random input for sequence size to record the time in each size in two txt for slow and fast

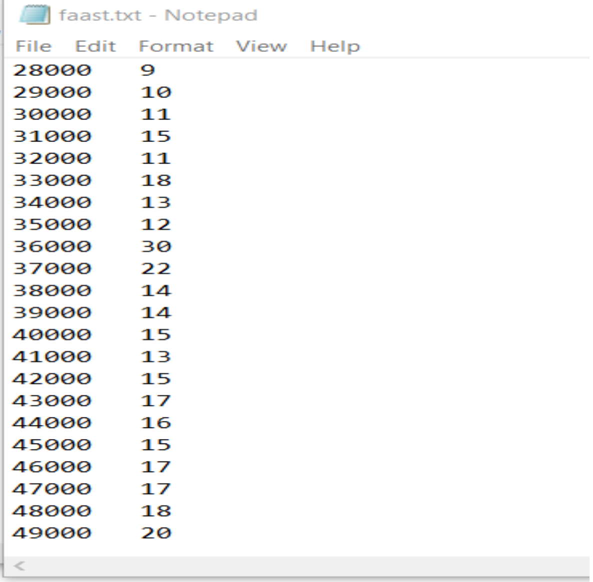


```
public static void main(String[] args) throws FileNotFoundException    {
        PrintStream slow = new PrintStream(new FileOutputStream("slow.txt"));
        PrintStream fast = new PrintStream(new FileOutputStream("fast.txt"));
      ArrayList<Integer> unordered = new ArrayList(); long t;
      ISort tests = new Sort();

     int ip; Random r = new Random();
     for(int i=1000;i<=100000;i+=1000) {
         unordered.clear();
         for(int j=0;j<i;j++) {
         unordered.add(r.nextInt(50000));
         }
        long start= System.currentTimeMillis();
        tests.sortSlow(unordered);
   t= System.currentTimeMillis()-start;
        System.setOut(slow);
   System.out.println(i + "\t" + t);
        start = System.currentTimeMillis();
        tests.sortFast(unordered);
        t=System.currentTimeMillis()-start;
        System.setOut(fast);
        System.out.println(i + "\t" + t);
     }

}
```

We can see the different in txt files for slow and fast in big input :

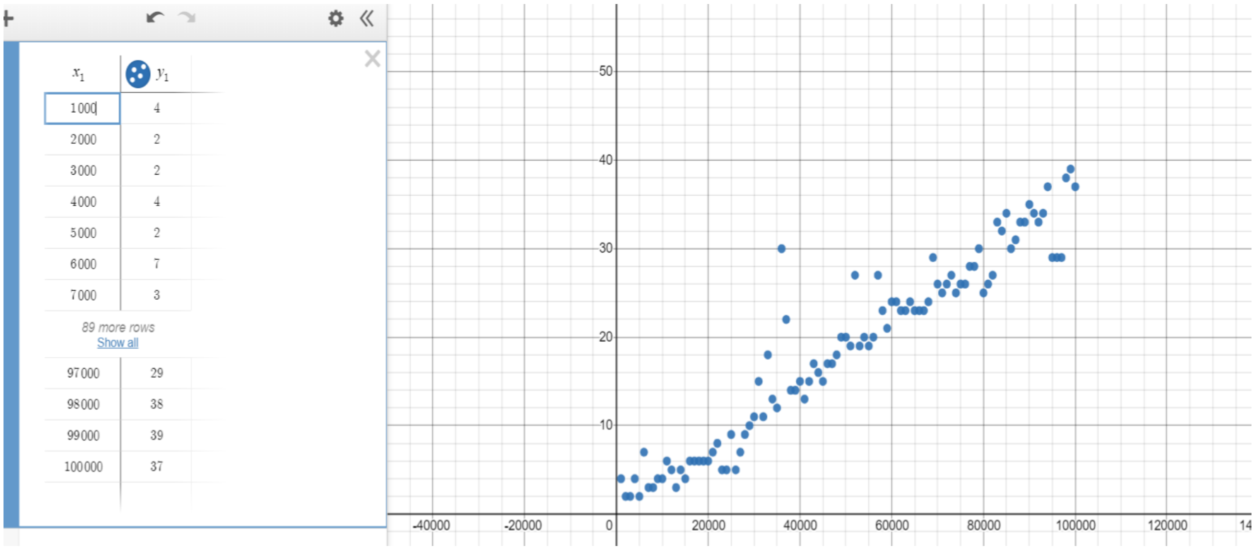


```
28000    11721
29000    10712
30000    12507
31000    12763
32000    14086
33000    14206
34000    17387
35000    18560
36000    21358
37000    18706
38000    19982
39000    19852
40000    23210
41000    26833
42000    26051
43000    25747
44000    27849
45000    34040
46000    37108
47000    35453
48000    38729
49000    40441
```

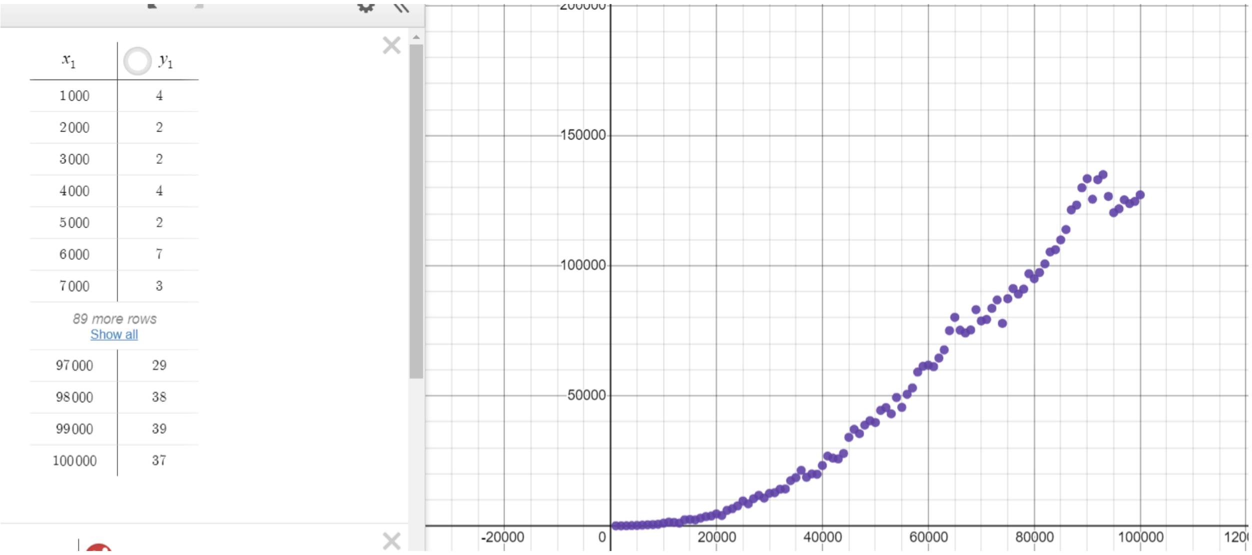


```
28000    9
29000    10
30000    11
31000    15
32000    11
33000    18
34000    13
35000    12
36000    30
37000    22
38000    14
39000    14
40000    15
41000    13
42000    15
43000    17
44000    16
45000    15
46000    17
47000    17
48000    18
49000    20
```

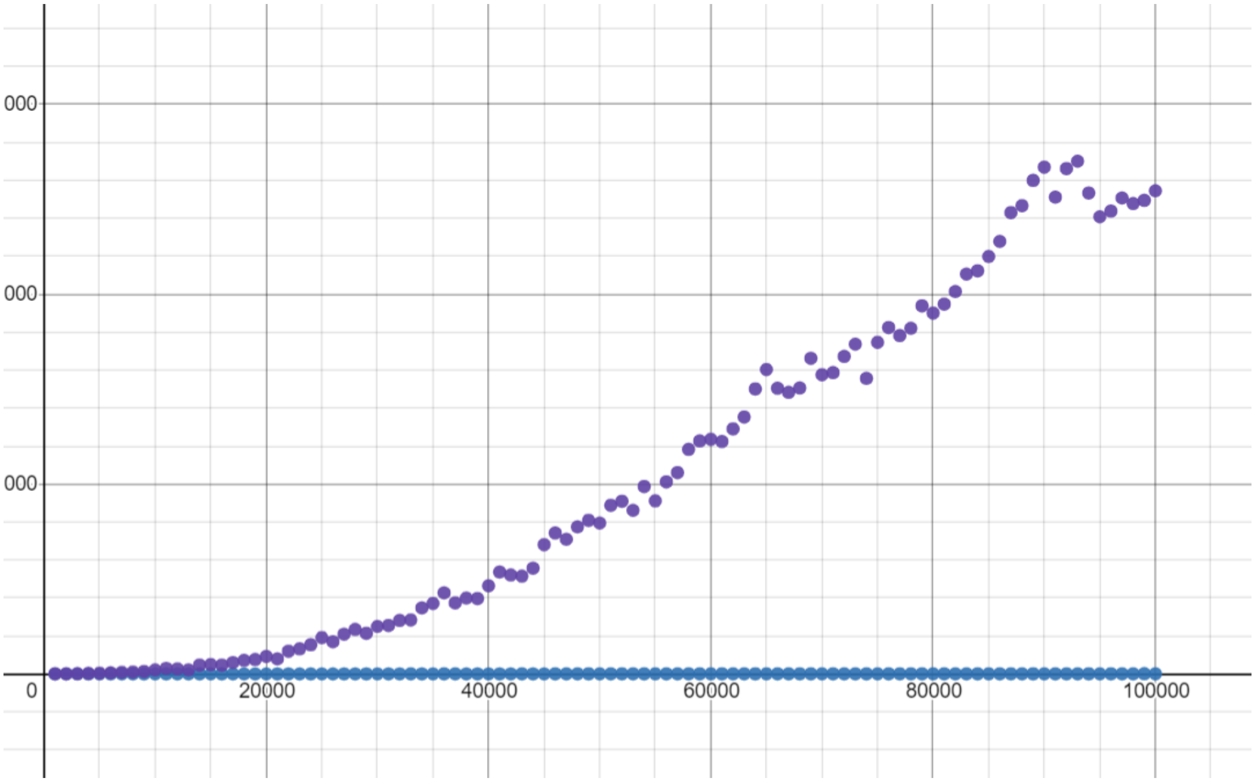
using https://www.desmos.com/calculator to draw the database in txt files .



x_1	y_1
1000	4
2000	2
3000	2
4000	4
5000	2
6000	7
7000	3
89 more rows Show all	
97000	29
98000	38
99000	39
100000	37



x_1	y_1
1000	4
2000	2
3000	2
4000	4
5000	2
6000	7
7000	3
89 more rows Show all	
97000	29
98000	38
99000	39
100000	37


000
000
000
0
20000
40000
60000
80000
100000