Lab 1

Name: Toka Alaa Elgindy **ID: 14**

Name: Nada Salama Mohammed **ID: 57**

Introduction:

The (binary) heap data structure is an array object that we can view as a nearly complete binary tree as shown in 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, the specifics of which depend on the kind of heap. In a max-heap, the max-heap property is that for every node i other than the root,

$$A[parent[i]] \geq A[i] \quad (1)$$

that is, the value of a node is at most the value of its parent.

Data structure:

1) Every element in the heap is represented by a node which implements "INode" interface.
2) Heap is represented by an ArrayList of INode.

Algorithm used:

1) **Max-Heapify:**

```
Max-Heapify(INode node)
    If ( node != NIL)
        left = node.getLeftChild();
        right = node.getRightChild();
        if(left != null && left.getValue() >  node.getValue())
            largest = left;
        else largest = node;
        if(right != null && right.getValue() > (largest.getValue())
            largest = right;
        if(largest != node)
          swap(node,largest);
          if(node != getRoot())
              heapify(node.getParent());
         heapify(largest);
```

2) **BUILD-MAX-HEAP:**

```
BUILD-MAX-HEAP(Collection unordered)
  heap = new ArrayList
  if (unordered != null} (
        Iterator iterator = unordered.iterator
        int j = 0
        while(iterator.hasNext)
              n = new Node ( j++)
              n.setValue(iterator.next)
              heap.add(n)
              for (i = size/2 – 1  down to  i = 0)
                heapify(heap.get(i))
        end while
```

3) **MAX-HEAP-INSERT:**

```
MAX-HEAP-INSERT(element)
   if(element != null)
        n = new Node(size)
        n.setValue(element)
        heap.add(n)
        size++
        heapify(heap.get(size - 1).getParent())
  end if
```

4) **HEAP-REMOVE-MAX:**

```
HEAP-REMOVE-MAX
if(size == 0) return NIL;
        root = getRoot()
        if(size == 1)
              heap.remove(0);
              size--;
        else if(root != null)
              heap.get(0) = heap.get(size - 1)
              heap.remove(size - 1);
              size--;
              heapify(heap.get(0));
        end if
        return root
```

5)

```
public IHeap heapSort(ArrayList unordered) {
    MyHeap h = new MyHeap();
    if (unordered != null) {
        h.build(unordered);
        int n = h.size();
        // One by one extract an element from heap
        for (int i = n - 1; i >= 1; i--)
        {
            h.swap((INode)h.getArrHeap().get(0), (INode)h.getArrHeap().get(i));
            h.setSize(i);
            h.heapify((INode)h.getArrHeap().get(0));
        }
        h.setSize(n);
    }
    return h;
}
```

- **First build Heap with our unordered array list**
- **Then save the size of Heap in integer**
- **Then pass on all node in heap and swap with the maximum element in heap which is in index 0**
- **Then reduce the size by one**
- **Then heapify the swapped node**
- **Finally return our size to the actual size**

```
public void sortSlow(ArrayList unordered) {
    //Bubble Sort
    if(unordered != null) {
        int n = unordered.size();
        for (int i = 0; i < n-1; i++) {
            for (int j = 0; j < n-i-1; j++) {
                if (unordered.get(j) instanceof Integer) {
                int comp = Integer.compare((Integer) unordered.get(j), (Integer) unordered.get(j + 1));
                if (comp > 0) {
                    Object temp = unordered.get(j);
                    unordered.set(j, unordered.get(j + 1));
                    unordered.set(j + 1, temp);
                }
                }
            }
        }
    }
}
```

- **First check if input is not null**
- **then loop on all element in array list and check first if it is comparable or not**
- **then if index in our element greater than next one, swap them**

```
public void sortFast(ArrayList unordered) {
    if (unordered != null) {
        if (unordered.size() != 0)
            if (unordered.get(0) instanceof Integer)
                sort(unordered, 0, unordered.size() - 1);
    }
}

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

- **first called function sort with parameter of our unordered list and two integers are the index of left & the index of right**
- **second sort do recursive call to divide our list to sub list**
- **finally merge two sorted parts in function merge**