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

Name : Ahmed Khaled

ID : 9

-Requirements :

- implementing binary heap and heap sort .
- implementing at least on of the sorting algorithms from each class (nlog(n) , n^2).
- Comparing the running time performance of the heap sort against the other implemented sorting techniques .

-The implemented sorting techniques :

O(nlog(n)) :

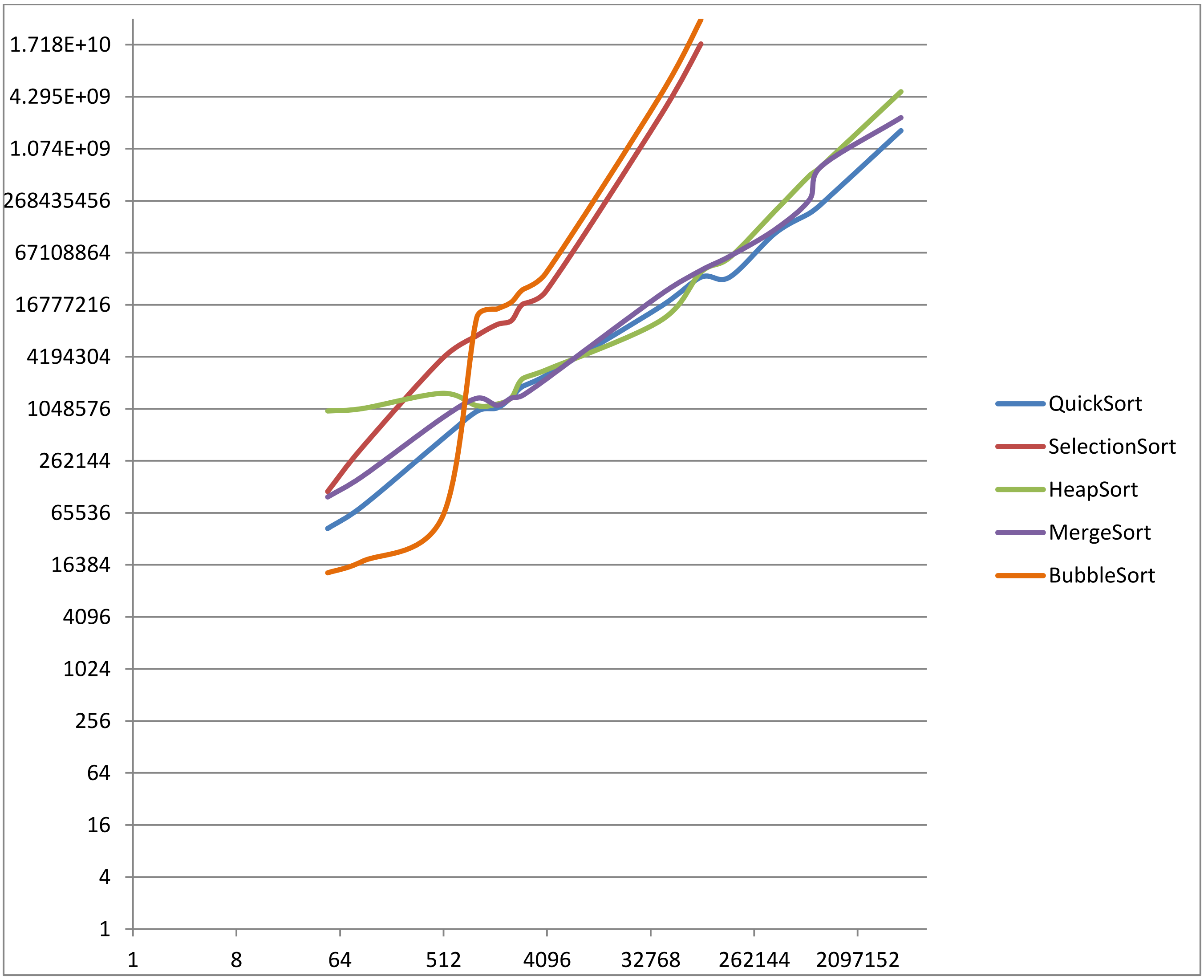
-Heap Sort .
-Quick Sort .
-Merge Sort .

O(n^2) :
-Selection Sort .
-Bubble Sort .

Experiment :

-Generating 5 random arrays of different sizes and evaluate the runtime for each sorting techniques .

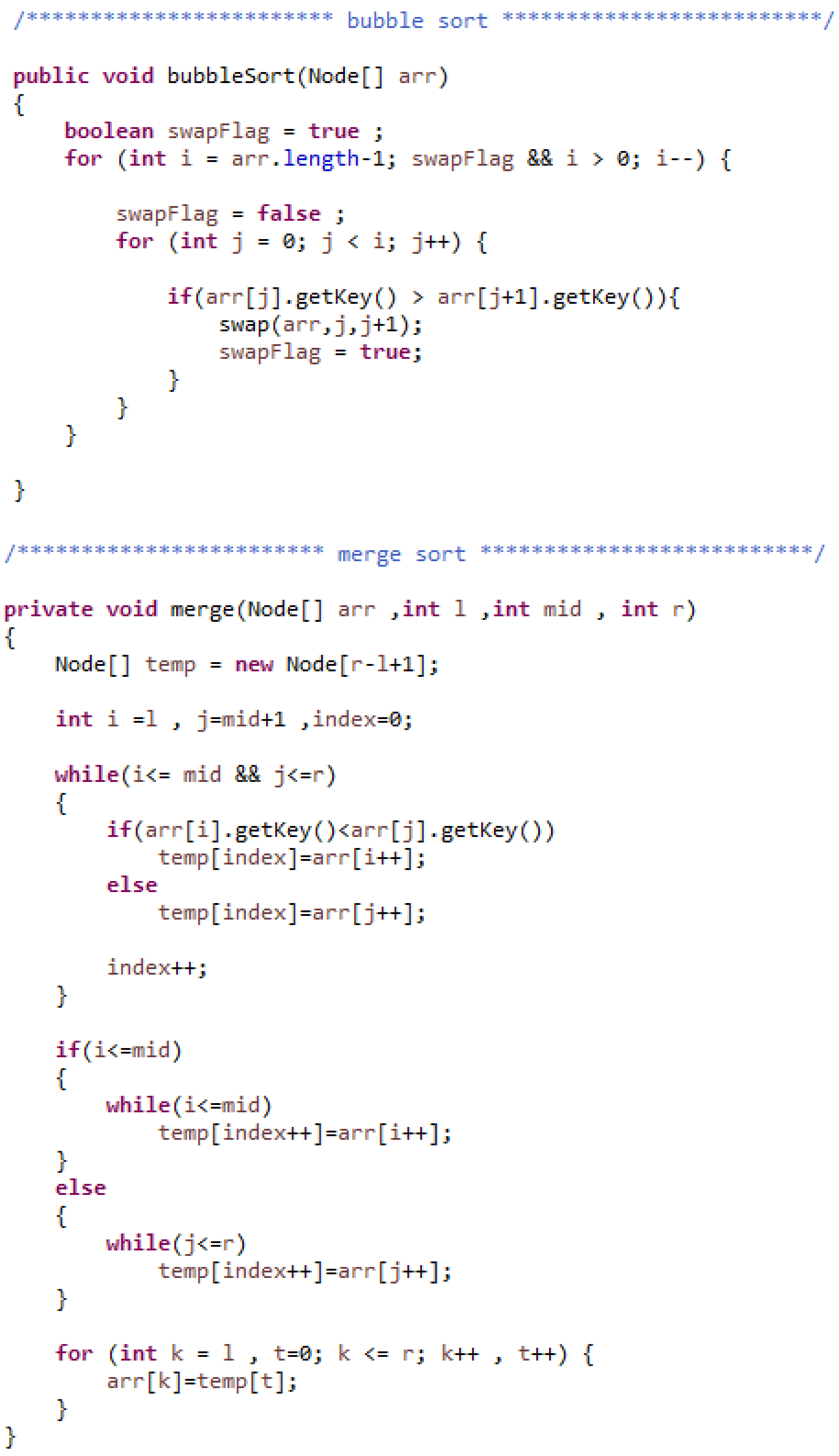
- The execution time of the sorting algorithms versus the input size :



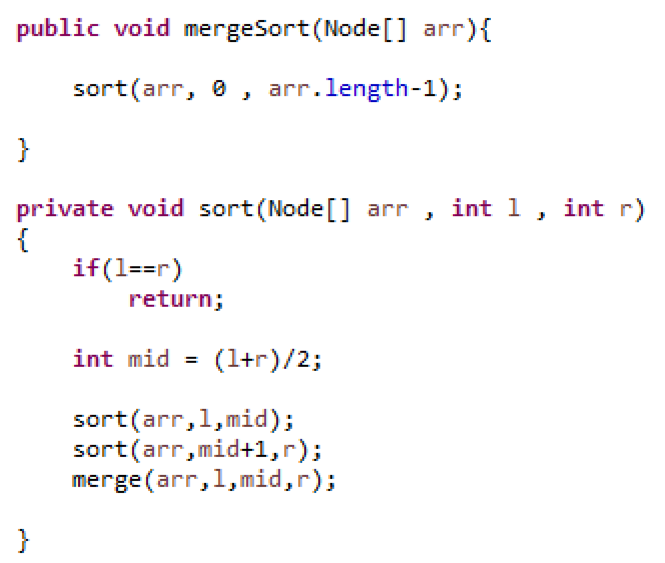
Y- axis : running time

X- axis : Input Size

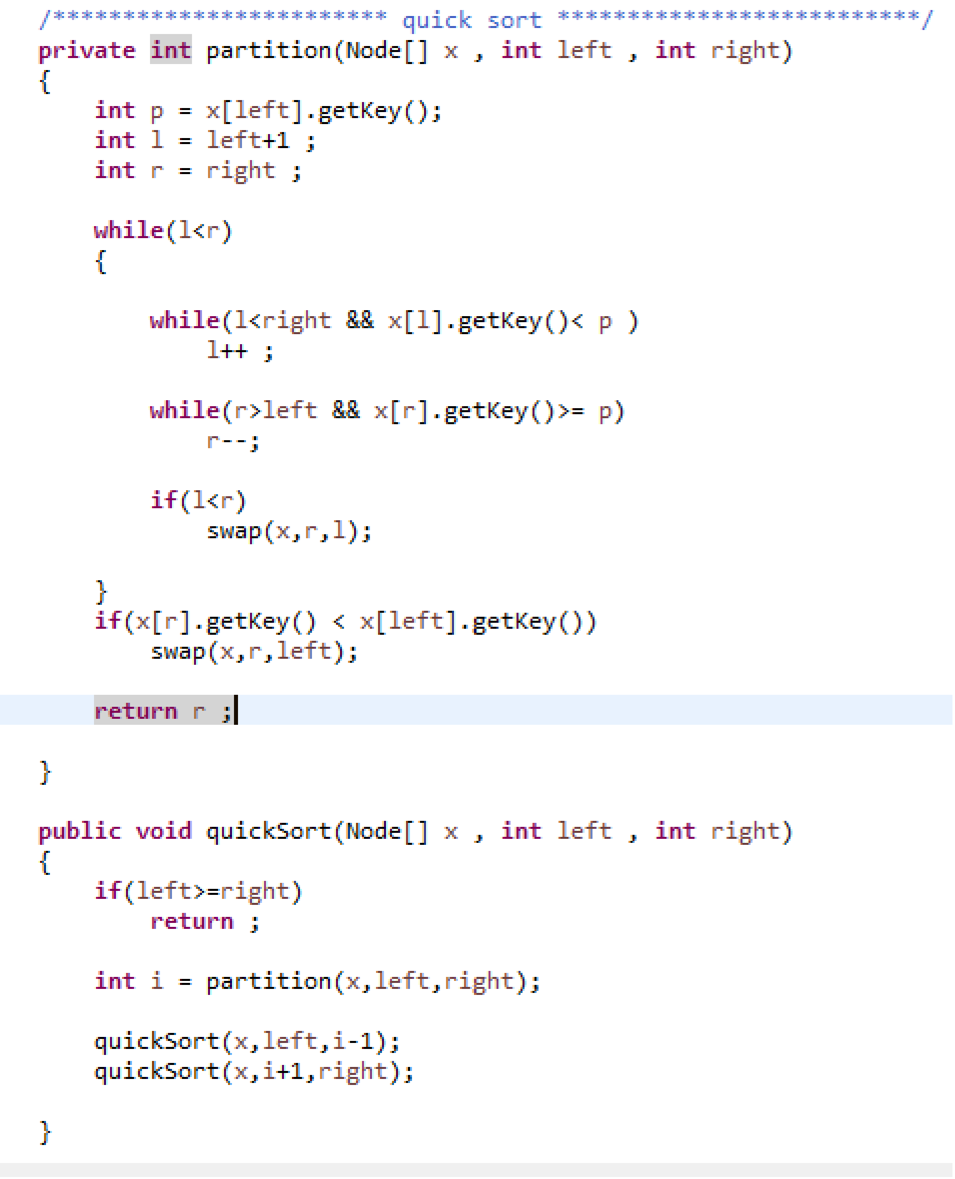
Source Code :



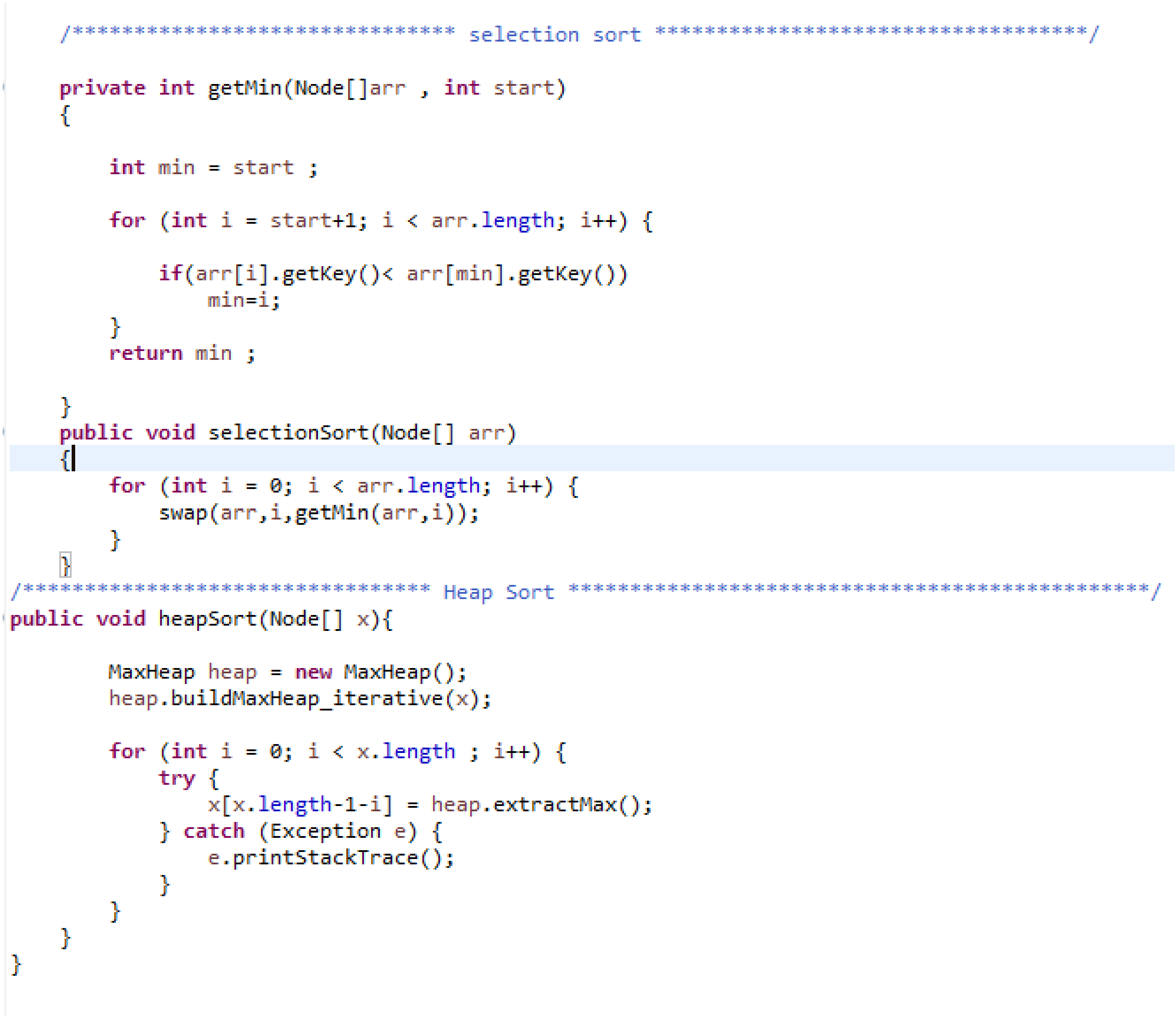
```
/************************ bubble sort *************************/

public void bubbleSort(Node[] arr)
{
    boolean swapFlag = true ;
    for (int i = arr.length-1; swapFlag && i > 0; i--) {

        swapFlag = false ;
        for (int j = 0; j < i; j++) {

            if(arr[j].getKey() > arr[j+1].getKey()){
                swap(arr,j,j+1);
                swapFlag = true;
            }
        }
    }

}

/************************ merge sort *************************/

private void merge(Node[] arr ,int l ,int mid , int r)
{
    Node[] temp = new Node[r-l+1];

    int i =l , j=mid+1 ,index=0;

    while(i<= mid && j<=r)
    {
        if(arr[i].getKey()<arr[j].getKey())
            temp[index]=arr[i++];
        else
            temp[index]=arr[j++];

        index++;
    }

    if(i<=mid)
    {
        while(i<=mid)
            temp[index++]=arr[i++];
    }
    else
    {
        while(j<=r)
            temp[index++]=arr[j++];
    }

    for (int k = l , t=0; k <= r; k++ , t++) {
        arr[k]=temp[t];
    }
}
```



```
public void mergeSort(Node[] arr){

    sort(arr, 0 , arr.length-1);

}

private void sort(Node[] arr , int l , int r)
{
    if(l==r)
        return;

    int mid = (l+r)/2;

    sort(arr,l,mid);
    sort(arr,mid+1,r);
    merge(arr,l,mid,r);

}
```



```
/************************* quick sort *************************/
private int partition(Node[] x , int left , int right)
{
    int p = x[left].getKey();
    int l = left+1 ;
    int r = right ;

    while(l<r)
    {

        while(l<right && x[l].getKey()< p )
            l++ ;

        while(r>left && x[r].getKey()>= p)
            r--;

        if(l<r)
            swap(x,r,l);

    }
    if(x[r].getKey() < x[left].getKey())
        swap(x,r,left);

    return r ;

}

public void quickSort(Node[] x , int left , int right)
{
    if(left>=right)
        return ;

    int i = partition(x,left,right);

    quickSort(x,left,i-1);
    quickSort(x,i+1,right);

}
```



```
    /******************************** selection sort ************************************/

    private int getMin(Node[]arr , int start)
    {

        int min = start ;

        for (int i = start+1; i < arr.length; i++) {

            if(arr[i].getKey()< arr[min].getKey())
                min=i;
        }
        return min ;

    }
    public void selectionSort(Node[] arr)
    {
        for (int i = 0; i < arr.length; i++) {
            swap(arr,i,getMin(arr,i));
        }
    }
/********************************** Heap Sort ************************************************/
public void heapSort(Node[] x){

        MaxHeap heap = new MaxHeap();
        heap.buildMaxHeap_iterative(x);

        for (int i = 0; i < x.length ; i++) {
            try {
                x[x.length-1-i] = heap.extractMax();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```