HEAPS

Code for adding an element in Priority queue:

```
import java.util.PriorityQueue;
class printApriorityQueue{

    public static void main(String[] args) {
        PriorityQueue<Integer>pq= new PriorityQueue<>();
        pq.add(3);
        pq.add(5);
        pq.add(1);
        pq.add(7);

        // to print a priority queue

        while(!pq.isEmpty()){
            System.out.print(pq.peek());
            pq.remove();
        }
    }

}

Output:
1 3 5 7
```

```
import java.util.Comparator;
import java.util.PriorityQueue;
class printApriorityQueue{

    public static void main(String[] args) {
        PriorityQueue<Integer>pq= new
PriorityQueue<>(Comparator.reverseOrder());// to print reverse order
        // pass the comparator.reverseorder() fxn in constructor
        pq.add(3);
        pq.add(5);
        pq.add(1);
        pq.add(7);

        // to print a priority queue

        while(!pq.isEmpty()){
```

```
            System.out.print(pq.peek());
            pq.remove();
        }
    }

}
Output: 7 5 3 1
```

Code for printing a Student information based on ranks:

```
import java.util.*;
public class comparatorinStudentclass {
  static class student implements Comparable<student>{
    String name;
    int rank;

    student(String name, int rank){
        this.name= name;
        this.rank= rank;
          }
    @Override
    public int compareTo(student s2){
        return this.rank - s2.rank;
    }
 }

 public static void main(String[] args) {
    PriorityQueue<student> pq= new PriorityQueue<>();
    pq.add(new student("q", 25));
    pq.add(new student("d", 2));
    pq.add(new student("C", 2));
    pq.add(new student("A", 25));

    while(!pq.isEmpty()){
        System.out.println(pq.peek().name+"-->"+pq.peek().rank);
        pq.remove();
    }

 }

}
Output:
```

```
d-->2
C-->2
A-->25
q-->25
```

code to insert into Heap:

```
import java.util.*;
public class addinHeap {

    static class addition{
        ArrayList<Integer> arr=new ArrayList<>();

        public void add(int data){
            arr.add(data); // adds data at last index

            int x= arr.size()-1;
            int par= (x-2)/2;

            while(arr.get(x)<arr.get(par)){// swap data
                int temp= arr.get(x);// inserts child in parent
                arr.set(x,par);         // insert par in child
                arr.set(par,temp);          // insert child in parent
            }

        }
        public int peek(){
            return arr.get(0);
        }


    }

    public static void main(String[] args) {


    }
}
```

Code for heap sort in ascending order:

```
public class Heapsort {
    public static void heapsort(int arr[]){
        int n= arr.length;
        for(int i=n/2;i>0;i--){
            heapify(arr,i,n);
        }
        for(int i=n-1;i>0;i--){
            int temp=arr[0];
            arr[0]=arr[i];
            arr[i]= temp;
            heapify(arr,0,i);
        }
    }

    public static void heapify(int arr[], int i,int size){
        int left=2*i+1;
        int right= 2*i+2;
        int max=i;
        if(left<size && arr[left]>arr[max]){
            max=left;
        }
        if(right<size && arr[right]>arr[max]){
            max=right;
        }
        if(max !=i){
            int temp= arr[i];
            arr[i]=arr[max];
            arr[max]= temp;
            heapify(arr, max, size);

        }
    }

    public static void main(String[] args) {
        int arr[]= {5,3,4,2,1};
        heapsort(arr);
        for(int i=0;i<arr.length;i++){
            System.out.println(arr[i] + " ");
        }
    }
}
Output:
1 2 3 4 5
```

Code for Heap sort in Descending order:

```
public class reverseHeapsort {

        public static void heapsort(int arr[]){
            int n= arr.length;
            for(int i=n/2;i>0;i--){
                heapify(arr,i,n);
            }
            for(int i=n-1;i>0;i--){
                int temp=arr[0];
                arr[0]=arr[i];
                arr[i]= temp;
                heapify(arr,0,i);
            }
        }
            public static void heapify(int arr[], int i,int size){
            int left=2*i+1;
            int right= 2*i+2;
            int min=i;
            if(left<size && arr[left] < arr[min]){
                min=left;
            }
            if(right<size && arr[right] < arr[min]){
                min=right;
            }
            if(min !=i){
                int temp= arr[i];
                arr[i]=arr[min];
                arr[min]= temp;
                heapify(arr, min, size);

            }       }

        public static void main(String[] args) {
            int arr[]= {1,5,3,4,2};
            heapsort(arr);
            for(int i=0;i<arr.length;i++){
                System.out.println(arr[i] + " ");
            }
        }    }
Output:
5,4,3,2,1
```

K-Nearest cars problem:

```
import java.util.*;

public class knearest {
    static class pts implements Comparable<pts>{
        int x;
        int y;
        int sqr;
        int idx;
        public pts(int x, int y,int sqr,int idx){
            this.x= x;
            this.y=y;
            this.sqr= sqr;
            this.idx=idx;
        }

    @Override
    public int compareTo(pts p2){
        return this.sqr- p2.sqr;
    }
    }

    public static void main(String[] args) {
        int pts[][]= {{3,3},{5,-1},{-2,4}};
        int k=2;
        PriorityQueue<pts> pq= new PriorityQueue<>();
        for(int i=0;i<pts.length;i++){
            int sqr= pts[i][0]*pts[i][0]+ pts[i][1]*pts[i][1]  ;
            pq.add(new pts(pts[i][0], pts[i][1], sqr, i));
        }
        for(int i=0;i<k;i++){
            System.out.println("C" + pq.remove().idx);
        }
    }

}
Output:
C0
C2
```

Code for N ropes problem:

```
import java.util.PriorityQueue;
```

```
public class Nropesproblem {
    public static void main(String[] args) {
        int ropes[]= {2,3,3,4,6};
        PriorityQueue<Integer> pq= new PriorityQueue<>();

        for(int i=0;i<ropes.length;i++){
            pq.add(ropes[i]);
        }

        int cost=0;

        while(pq.size()>1){
            int min= pq.remove();
            int min2= pq.remove();
            cost+=min+min2;
            pq.add(cost);

        }
        System.out.println(cost);

    }

}
Output:
58
```

Code for weakest soldier:

```
import java.util.PriorityQueue;

public class weakestSoldier {
    static class row implements Comparable<row>{
        int soldiers;
        int idx;

        public row(int soldiers,int idx){
            this.soldiers= soldiers;
            this.idx= idx;
        }

        @Override
        public int compareTo(row r2){
            if(this.soldiers==r2.soldiers){
```

```
                return  this.idx-r2.idx;
            }else{
                return this.soldiers-r2.soldiers;
            }
        }

    }

    public static void main(String[] args) {
        int army[][]={{1,0,0,0} , {1,1,1,1},{1,0,0,0},{1,0,0,0}};
        int k=2;
        PriorityQueue <row> pq= new PriorityQueue<>();
        for(int i=0;i<army.length;i++){
            int count=0;
            for(int j=0;j<army[0].length;j++){
                count+=army[i][j]==1?1:0;

            }
            pq.add(new row(count, i));
        }
         for(int i=0;i<k;i++){
            System.out.println( "R"+pq.remove().idx);
         }
    }

}
Output:
R0
R2
```

Code for Window Sliding problem :

```
import java.util.PriorityQueue;

public class slidingwindowproblem {
    static class Pair implements Comparable<Pair>{
        int val;
        int idx;
        public Pair(int val, int idx){
            this.val=val;
            this.idx=idx;
        }
        @Override

        public int compareTo(Pair p2){
```

```
            return p2.val-this.val;
        }
    }
    public static void main(String[] args) {
        int arr[]={1,3,-1,-3,5,3,6,7};
        int k=3;
        int res[]= new int[arr.length-k+1];
        PriorityQueue <Pair> pq= new PriorityQueue<>();
        for(int i=0;i<k;i++){
            pq.add(new Pair(arr[i],i));
        }
        res[0]= pq.peek().val;
        for(int i=k;i<arr.length;i++){
            while(pq.size()>0 && pq.peek().idx<=(i-k)){
                pq.remove();
            }
            pq.add(new Pair(arr[i], i));
            res[i-k+1]=pq.peek().val;
        }
        for(int i=0;i<res.length;i++){
            System.out.print(res[i]+" ");
        }
    }
}
Output:
3 3 5 5 6 7
```