Heap Solutions

Solution 1:

Time Complexity : o(logk)
Space Complexity: o(1)

```
import java.io.*;
import java.util.*;

class Solution {
static PriorityQueue<Integer> min;
static int k;

static List<Integer> getAllKthNumber(int arr[]){

        List<Integer> list = new ArrayList<>();
        for (int val : arr) {
        if (min.size() < k)
                min.add(val);
        else {
                if (val > min.peek()) {
                min.poll();
                min.add(val);
                }
        }

        if (min.size() >= k)
                list.add(min.peek());
        else
                list.add(-1);
        }
        return list;
}

public static void main(String[] args){
        min = new PriorityQueue<>();
        k = 4;
        int arr[] = { 1, 2, 3, 4, 5, 6 };
        List<Integer> res = getAllKthNumber(arr);
        for (int x : res)
        System.out.print(x + " ");
}
```

}

Solution 2 :

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.io.*;
import java.util.*;
class Solution{
        public static void minTime(int arr[],
                                                        int N, int K){

                Queue<Integer> q = new LinkedList<>();

                boolean vis[] = new boolean[N + 1];
                int time = 0;

                for (int i = 0; i < K; i++) {
                        q.add(arr[i]);
                        vis[arr[i]] = true;
                }

                while (q.size() > 0) {
                        for (int i = 0; i < q.size(); i++) {
                                int curr = q.poll();
                                if (curr - 1 >= 1 &&
                                        !vis[curr - 1]) {
                                        vis[curr - 1] = true;
                                        q.add(curr - 1);
                                }

                                if (curr + 1 <= N &&
                                        !vis[curr + 1]) {
                                        vis[curr + 1] = true;
                                        q.add(curr + 1);
                                }
                        }
                        time++;
```

```
            }

            System.out.println(time - 1);
        }

        public static void main(String[] args){
            int N = 6;
            int arr[] = { 2, 6 };
            int K = arr.length;
            minTime(arr, N, K);
        }
}
```

Solution 3 :

Time Complexity : o(n2logn)
Space Complexity: o(n2)


```
import java.util.Stack;

class Solution{
    static String decode(String str){
        Stack<Integer> integerstack = new Stack<>();
        Stack<Character> stringstack = new Stack<>();
        String temp = "", result = "";
        for (int i = 0; i < str.length(); i++){
            int count = 0;
            if (Character.isDigit(str.charAt(i))){
                while (Character.isDigit(str.charAt(i))){
                    count = count * 10 + str.charAt(i) - '0';
                    i++;
                }

                i--;
                integerstack.push(count);
            }

            else if (str.charAt(i) == ']'){
                temp = "";
                count = 0;
```

```
                if (!integerstack.isEmpty()){
                        count = integerstack.peek();
                        integerstack.pop();
                }

                while (!stringstack.isEmpty() && stringstack.peek()!='[' ){
                        temp = stringstack.peek() + temp;
                        stringstack.pop();
                }

                if (!stringstack.empty() && stringstack.peek() == '[')
                        stringstack.pop();

                for (int j = 0; j < count; j++)
                        result = result + temp;

                for (int j = 0; j < result.length(); j++)
                        stringstack.push(result.charAt(j));

                result = "";
        }
        else if (str.charAt(i) == '['){
                if (Character.isDigit(str.charAt(i-1)))
                        stringstack.push(str.charAt(i));

                else{
                        stringstack.push(str.charAt(i));
                        integerstack.push(1);
                }
        }

        else
                stringstack.push(str.charAt(i));
}

while (!stringstack.isEmpty()){
        result = stringstack.peek() + result;
        stringstack.pop();
```

```
            }
            return result;
        }

        public static void main(String args[]){
            String str = "3[b2[ca]]";
            System.out.println(decode(str));
        }
}
```

Solution 4 :

Time Complexity : o(nlogn)
Space Complexity: o(n)

```
import java.util.*;
import java.io.*;

class Solution{

static int minops(ArrayList<Integer> nums){
        int sum = 0;
        for(int i = 0 ; i < nums.size() ; i++){
        sum += nums.get(i);
        }

        PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
        for(int i = 0 ; i < nums.size() ; i++){
        pq.add(-nums.get(i));
        }

        double temp = sum;
        int cnt = 0;
        while (temp > sum / 2) {
        int x = -pq.peek();
        pq.remove();
        temp -= Math.ceil(x * 1.0 / 2);
        pq.add(x / 2);
        cnt++;
        }
```

```
        return cnt;
}

public static void main(String args[]){
        ArrayList<Integer> nums = new ArrayList<Integer>(
        List.of(
                4, 6, 3, 9, 10, 2
        )
        );
        int count = minops(nums);
        System.out.println(count);
}
}
```

Solution 5 :

Time Complexity : o(n*k*logk)
Space Complexity: o(k)


```
import java.io.*;
import java.util.*;

class Node {
        int data;
        Node next;

        Node(int key){
                data = key;
                next = null;
        }
}

class NodeComparator implements Comparator<Node> {
        public int compare(Node k1, Node k2){
                if (k1.data > k2.data)
                        return 1;
                else if (k1.data < k2.data)
                        return -1;
                return 0;
        }
```

```
}
class Solution {
	static Node mergeKList(Node[] arr, int K){
		PriorityQueue<Node> queue
			= new PriorityQueue<>(new NodeComparator());
		Node at[] = new Node[K];
		Node head = new Node(0);
		Node last = head;
		for (int i = 0; i < K; i++) {
			if (arr[i] != null) {
				queue.add(arr[i]);
			}
		}
		if (queue.isEmpty())
			return null;
		while (!queue.isEmpty()) {
			Node curr = queue.poll();
			last.next = curr;
			last = last.next;
			if (curr.next != null) {
				queue.add(curr.next);
			}
		}
		return head.next;
	}
	public static void printList(Node node){
		while (node != null) {
			System.out.print(node.data + " ");
			node = node.next;
		}
	}

	public static void main(String[] args){
		int N = 3;
		Node[] a = new Node[N];
		Node head1 = new Node(1);
		a[0] = head1;
		head1.next = new Node(3);
		head1.next.next = new Node(5);
		head1.next.next.next = new Node(7);
```

```
        Node head2 = new Node(2);
        a[1] = head2;
        head2.next = new Node(4);
        head2.next.next = new Node(6);
        head2.next.next.next = new Node(8);

        Node head3 = new Node(0);
        a[2] = head3;
        head3.next = new Node(9);
        head3.next.next = new Node(10);
        head3.next.next.next = new Node(11);

        Node res = mergeKList(a, N);

        if (res != null)
            printList(res);
        System.out.println();
    }
}
```