Queue Solutions

Solution 1 :

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

```
import java.util.LinkedList;
import java.util.Queue;

public class Solution {
	static void generatePrintBinary(int n){
		Queue<String> q = new LinkedList<String>();
		q.add("1");
		while (n-- > 0) {
			String s1 = q.peek();
			q.remove();
			System.out.println(s1);
			String s2 = s1;
			q.add(s1 + "0");
			q.add(s2 + "1");
		}
	}

	public static void main(String[] args){
		int n = 10;
		generatePrintBinary(n);
	}
}
```

Solution 2 :

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

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

```
class Solution{
	static int minCost(int arr[], int n){
		PriorityQueue<Integer> pq
			= new PriorityQueue<Integer>();

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

		int res = 0;
		while (pq.size() > 1) {
			int first = pq.poll();
			int second = pq.poll();
			res += first + second;
			pq.add(first + second);
		}
		return res;
	}

	public static void main(String args[]){
		int len[] = { 4, 3, 2, 6 };
		int size = len.length;
		System.out.println("Total cost for connecting"
						+ " ropes is "
						+ minCost(len, size));
	}
}
```

Solution 3 :

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

```
import java.util.*;

class Solution {
	static class Job {
		char job_id;
		int deadline;
		int profit;
```

```
        Job(char job_id, int deadline, int profit){
            this.deadline = deadline;
            this.job_id = job_id;
            this.profit = profit;
        }
}

static void printJobScheduling(ArrayList<Job> arr){
        int n = arr.size();
        Collections.sort(arr, (a, b) -> {
            return a.deadline - b.deadline;
        });
        ArrayList<Job> result = new ArrayList<>();
        PriorityQueue<Job> maxHeap = new PriorityQueue<>(
            (a, b) -> { return b.profit - a.profit; });
        for (int i = n - 1; i > -1; i--) {
            int slot_available;
            if (i == 0) {
                slot_available = arr.get(i).deadline;
            }
            else {
                slot_available = arr.get(i).deadline
                                 - arr.get(i - 1).deadline;
            }
            maxHeap.add(arr.get(i));
            while (slot_available > 0
                && maxHeap.size() > 0) {
                Job job = maxHeap.remove();
                slot_available--;
                result.add(job);
            }
        }

        Collections.sort(result, (a, b) -> {
            return a.deadline - b.deadline;
        });

        for (Job job : result) {
            System.out.print(job.job_id + " ");
        }
```

```
            System.out.println();
        }

        public static void main(String[] args){
            ArrayList<Job> arr = new ArrayList<Job>();

            arr.add(new Job('a', 2, 100));
            arr.add(new Job('b', 1, 19));
            arr.add(new Job('c', 2, 27));
            arr.add(new Job('d', 1, 25));
            arr.add(new Job('e', 3, 15));

            System.out.println("Following is maximum "
                                    + "profit sequence of jobs");
            printJobScheduling(arr);
        }
}
```

Solution 4 :

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

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

import java.util.*;

class Solution {

        static class cell {
            int x, y;
            int dis;
            public cell(int x, int y, int dis){
                this.x = x;
                this.y = y;
                this.dis = dis;
```

```
        }
}

static boolean isInside(int x, int y, int N){
        if (x >= 1 && x <= N && y >= 1 && y <= N)
                return true;
        return false;
}

static int minStepToReachTarget(
        int knightPos[], int targetPos[],
        int N){
        int dx[] = { -2, -1, 1, 2, -2, -1, 1, 2 };
        int dy[] = { -1, -2, -2, -1, 1, 2, 2, 1 };

        Vector<cell> q = new Vector<>();
        q.add(new cell(knightPos[0], knightPos[1], 0));
        cell t;
        int x, y;
        boolean visit[][] = new boolean[N + 1][N + 1];
        visit[knightPos[0]][knightPos[1]] = true;
        while (!q.isEmpty()) {
                t = q.firstElement();
                q.remove(0);
                if (t.x == targetPos[0] && t.y == targetPos[1])
                        return t.dis;
                for (int i = 0; i < 8; i++) {
                        x = t.x + dx[i];
                        y = t.y + dy[i];
                        if (isInside(x, y, N) && !visit[x][y]) {
                                visit[x][y] = true;
                                q.add(new cell(x, y, t.dis + 1));
                        }
                }
        }
        return Integer.MAX_VALUE;
}

public static void main(String[] args){
        int N = 30;
```

```
                int knightPos[] = { 1, 1 };
                int targetPos[] = { 30, 30 };
                System.out.println(
                        minStepToReachTarget(
                                knightPos, targetPos, N));
        }
}
```

Solution 5 :

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

```
import java.util.Deque;
import java.util.LinkedList;

public class Solution {

   static void printMax(int arr[], int n, int k){

      Deque<Integer> Qi = new LinkedList<Integer>();
      int i;
      for (i = 0; i < k; ++i) {
         while (!Qi.isEmpty() && arr[i] >=
                  arr[Qi.peekLast()])
            Qi.removeLast();
         Qi.addLast(i);
      }
      for (; i < n; ++i) {
         System.out.print(arr[Qi.peek()] + " ");
         while ((!Qi.isEmpty()) && Qi.peek() <=
                              i - k)
            Qi.removeFirst();
         while ((!Qi.isEmpty()) && arr[i] >=
                     arr[Qi.peekLast()])
            Qi.removeLast();
         Qi.addLast(i);
      }
      System.out.print(arr[Qi.peek()]);
   }
```

```
    public static void main(String[] args){
        int arr[] = { 12, 1, 78, 90, 57, 89, 56 };
        int k = 3;
        printMax(arr, arr.length, k);
    }
}
```