Assignment No 11

Que.1 Answer

```
class Solution {
   public String makeGood(String s) {
        StringBuilder res = new StringBuilder();
       for (char ch : s.toCharArray()) {
          int resLength = res.length();
       if (resLength!=0 && Math.abs(res.charAt(resLength-1) - ch) == 32) {
             res.deleteCharAt(resLength - 1);
          } else {
             res.append(ch);
          }
       }
       return res.toString();
   }
}
```

Que.2 Answer

```
class Solution {
    public String removeDuplicates(String s) {
        Stack<Character> stack = new Stack<>();

        for (char c : s.toCharArray()) {
            if (!stack.isEmpty() && c == stack.peek()) {
                stack.pop();
            } else {
                stack.push(c);
            }
        }

        StringBuilder ans = new StringBuilder();
        for (char c : stack) ans.append(c);
        return ans.toString();
    }
}
```

Que 3 Answer :

```
class StockSpanner {

    private Deque<int[]> stack = new LinkedList<>();

    public StockSpanner() {

    }

    public int next(int price) {

        int tot=1;

        while(!stack.isEmpty() && stack.peek()[1]<=price){

            tot+=stack.pop()[0];

        }

        stack.push(new int[]{tot,price});

        return tot;

    }

}
```

Que 4 Answer :

```
class Solution {
    public int timeRequiredToBuy(int[] tickets, int k) {
         Queue<Integer> queue = new LinkedList<>();
        int ans = 0;
        for(int i = 0; i < tickets.length; i++){
            queue.add(i);
        }
        while(!queue.isEmpty()){
            int index = queue.poll();
            tickets[index]--;
            ans++;
            if(tickets[index] == 0 && index == k){
                return ans;
            }
            if(tickets[index] > 0){
                queue.add(index);
            }
        }
        return ans;
    }
}
```

Que 5 *Answer* :

```
class ProductOfNumbers {
    List<Integer> list;
    public ProductOfNumbers() {
        list = new ArrayList<>();
    }
    public void add(int num) {
        list.add(num);
    }
    public int getProduct(int k) {
        int n = list.size();
        int prod = 1;
        for (int i = n - k; i < n; i++) {
            prod *= list.get(i);
        }
        return prod;
    }
}
```

Que 6 Answer :

```
 int n = heights.length;

       int smallleft[] = new int[n];

       int smallright[] = new int[n];

       Stack<Integer> s = new Stack<>();

       for(int i=0;i<heights.length;i++){

           while(!s.isEmpty() && heights[s.peek()] >= heights[i] ){

               s.pop();

           }

           if(s.isEmpty()){

               smallleft[i] = -1;

           }

           else{

               smallleft[i] = s.peek();

           }

           s.push(i);

       }

       s = new Stack<>();

       for(int i=heights.length-1; i>=0; i--){

           while(!s.isEmpty() && heights[s.peek()] >= heights[i] ){

               s.pop();

           }

           if(s.isEmpty()){

               smallright[i] = heights.length;
```

```
            }
            else{
                smallright[i] = s.peek();
            }

            s.push(i);
        }
        int largeRectangle = 0;
        for(int i=0; i<heights.length; i++){
            int height = heights[i];
            int width = smallright[i] -smallleft[i]-1;
            int currRectangle = height * width;
            largeRectangle = Math.max(currRectangle, largeRectangle);
        }
        return largeRectangle;
    }
}
```

Que 7 Answer :

```
class Solution {
    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 int[] maxSlidingWindow(int[] nums, int k) {
        int[] ans=new int[nums.length-k+1];
        PriorityQueue<Pair> pq=new PriorityQueue<>();
        for(int i=0;i<k;i++){
            pq.add(new Pair(nums[i],i));
        }
        ans[0]=pq.peek().val;
        for(int i=k;i<nums.length;i++){
            while(pq.size()>0 && pq.peek().idx<=(i-k)){
                pq.remove();
            }
            pq.add(new Pair(nums[i],i));
            ans[i-k+1]=pq.peek().val;
        }
        return ans;
    }
}
```

Que 8 Answer :

```
public class CircularQueue {
    private int[] queue;
    private int front;
    private int rear;
    private int size;
    private int capacity;

    public CircularQueue(int capacity) {
        this.capacity = capacity;
        this.queue = new int[capacity];
        this.front = -1;
        this.rear = -1;
        this.size = 0;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public boolean isFull() {
        return size == capacity;
    }
```

```
public void enQueue(int item) {
    if (isFull()) {
        System.out.println("Queue is full. Cannot enqueue.");
        return;
    }

    if (isEmpty()) {
        front = 0;
        rear = 0;
    } else {
        rear = (rear + 1) % capacity;
    }

    queue[rear] = item;
    size++;
}

public void deQueue() {
    if (isEmpty()) {
        System.out.println("Queue is empty. Cannot dequeue.");
        return;
    }

    if (front == rear) {
        front = -1;
        rear = -1;
    } else {
        front = (front + 1) % capacity;
    }

    size--;
}

public int Front() {
    if (isEmpty()) {
```