```
public class Stacks_Queue {
    // Implement Stack using Arrays
    class MyStack {
        int top;
        int arr[] = new int[1000];

        MyStack() {
            top = -1;
        }

        // Function to push an integer into the stack.
        void push(int a) {
            // Your code here
            top++;
            arr[top] = a;
        }

        // Function to remove an item from top of the stack.
        int pop() {
            // Your code here
            if (top == -1) {
                return -1;
            }
            int ele = arr[top];
            top--;
            return ele;
        }
    }

    // Implement Queue using Arrays
    class MyQueue {

        int front, rear;
        int arr[] = new int[100005];

        MyQueue() {
            front = -1;
            rear = -1;
        }

        // Function to push an element x in a queue.
        void push(int x) {
            // Your code here
            if (front == -1 && rear == -1) {
                front++;
                rear++;
                arr[front] = x;
            } else {
                rear++;
```

```
            arr[rear] = x;
        }
    }

    // Function to pop an element from queue and return that element.
    int pop() {
        // Your code here
        if (front == -1) {
            return -1;
        }
        if (front == rear) {
            int ele = arr[front];
            front++;
            return ele;
        }
        if (front > rear) {
            return -1;
        }
        int ele = arr[front];
        front++;
        return ele;
    }
}

// Implement Stack using Queue
class Queues {
    Queue<Integer> q1 = new LinkedList<Integer>();
    Queue<Integer> q2 = new LinkedList<Integer>();

    // Function to push an element into stack using two queues.
    void push(int a) {
        // Your code here
        q1.add(a);
    }

    // Function to pop an element from stack using two queues.
    int pop() {
        // Your code here
        int n1 = q1.size();
        if (n1 == 0) {
            return -1;
        }
        for (int i = 0; i < n1 - 1; i++) {
            q2.add(q1.remove());
        }
        int ele = q1.remove();
        for (int i = 0; i < n1 - 1; i++) {
            q1.add(q2.remove());
        }
```

```
        return ele;
    }

}

// Implement Queue using Stack
class Queue {
    Stack<Integer> input = new Stack<Integer>();
    Stack<Integer> output = new Stack<Integer>();

    /* The method pop which return the element poped out of the stack */
    int dequeue() {
        // Your code here
        int n1 = input.size();
        if (n1 == 0) {
            return -1;
        }
        for (int i = 0; i < n1 - 1; i++) {
            output.add(input.pop());
        }
        int ele = input.pop();
        for (int i = 0; i < n1 - 1; i++) {
            input.add(output.pop());
        }
        return ele;
    }

    /* The method push to push element into the stack */
    void enqueue(int x) {
        // Your code here
        input.push(x);
    }
}
// Implement stack using Linkedlist
// Implement queue using Linkedlist
// Check for balanced paranthesis
// Implement Min Stack
// Next Greater Element
class Solution {
    public int[] nextGreaterElement(int[] nums1, int[] arr) {
        int n = arr.length;
        int ans[] = new int[n];
        Stack<Integer> st = new Stack<>(); // will have indices
        for (int i = 0; i < n; i++) {
            while (st.size() > 0 && arr[i] > arr[st.peek()]) {
                ans[st.peek()] = arr[i]; // fix / store NGE
                st.pop();
            }
            st.push(i);
```

```
        }
        while (st.size() > 0) {
            ans[st.peek()] = -1;
            st.pop();
        }
        Map<Integer, Integer> mp = new HashMap<>();
        for (int i = 0; i < n; i++) {
            mp.put(arr[i], ans[i]);
        }
        int n1 = nums1.length;
        int ans1[] = new int[n1];
        for (int i = 0; i < n1; i++) {
            ans1[i] = mp.get(nums1[i]);
        }
        return ans1;
    }
}

// Next Greater Element 2
class Solution {
    public int[] nextGreaterElements(int[] nums) {
        int n = nums.length;
        int nge[] = new int[n];
        Stack<Integer> st = new Stack<>();
        for (int i = 2 * n - 1; i >= 0; i--) {
            while (st.isEmpty() == false && st.peek() <= nums[i % n]) {
                st.pop();
            }
            if (i < n) {
                if (st.isEmpty() == false) {
                    nge[i] = st.peek();
                } else {
                    nge[i] = -1;
                }
            }
            st.push(nums[i % n]);
        }
        return nge;
    }
}
// Next Smaller Element
class Solution {
    void immediateSmaller(int arr[], int n) {
        // code here
        for (int i = 0; i < n - 1; i++) {
            if (arr[i] > arr[i + 1]) {
                arr[i] = arr[i + 1];
            } else {
                arr[i] = -1;
```

```
            }
        }
        arr[n - 1] = -1;
    }
}
// Number of NGEs to the right
class Solution {
    public static int[] count_NGEs(int N, int arr[], int queries, int indices[]) {
        // code here
        int n = indices.length;
        int ans[] = new int[queries];
        for (int i = 0; i < queries; i++) {
            int start = indices[i];
            int count = 0;
            for (int j = start; j < N; j++) {
                if (arr[j] > arr[start]) {
                    count++;
                }
            }
            ans[i] = count;
        }
        return ans;
    }
}
// Trapping Rainwater
class Solution {
    public int trap(int[] arr) {
        int n = arr.length;
        int[] leftmax = new int[n];
        int[] rightmax = new int[n];
        leftmax[0] = Integer.MIN_VALUE;
        for (int i = 1; i < n; i++) {
            leftmax[i] = Math.max(leftmax[i - 1], arr[i - 1]);
        }
        rightmax[n - 1] = Integer.MIN_VALUE;
        for (int i = n - 2; i >= 0; i--) {
            rightmax[i] = Math.max(rightmax[i + 1], arr[i + 1]);
        }
        int water = 0;
        for (int i = 1; i <= n - 2; i++) {
            int units = Math.min(leftmax[i], rightmax[i]) - arr[i];
            if (units > 0) {
                water += units;
            }
        }
        return water;
    }
}
// Sum of subarray minimum
```

```
class Solution {
    public int sumSubarrayMins(int[] arr) {
        Stack<Integer> st1 = new Stack<>();
        Stack<Integer> st2 = new Stack<>();
        int n = arr.length;
        int nser[] = new int[n];
        int nsel[] = new int[n];
        for (int i = 0; i < n; i++) {
            while (!st1.isEmpty() && arr[i] <= arr[st1.peek()]) {
                nser[st1.peek()] = i;
                st1.pop();
            }
            st1.push(i);
        }
        while (!st1.isEmpty()) {
            nser[st1.pop()] = n;
        }
        for (int i = n - 1; i >= 0; i--) {
            while (!st2.isEmpty() && arr[i] < arr[st2.peek()]) {
                nsel[st2.peek()] = i;
                st2.pop();
            }
            st2.push(i);
        }
        while (!st2.isEmpty()) {
            nsel[st2.pop()] = -1;
        }
        int ans = 0;
        int M = 1000000007;
        for (int i = 0; i < n; i++) {
            int num = (int) (i - nsel[i]) * (int) (nser[i] - i);
            int temp = ((num % M) * (arr[i] % M)) % M;
            ans = (ans % M + temp % M) % M;
        }
        return ans;
    }
}

// Asteroid Collision
class Solution {
    public int[] asteroidCollision(int[] asteroids) {
        Stack<Integer> st = new Stack<>();
        for (int i = 0; i < asteroids.length; i++) {
            if (st.isEmpty() || asteroids[i] > 0) {
                st.push(asteroids[i]);
            } else {
                while (true) {
                    int peek = st.peek();
                    if (peek < 0) {
```

```
                    st.push(asteroids[i]);
                    break;
                } else if (peek == -asteroids[i]) {
                    st.pop();
                    break;
                } else if (peek > -asteroids[i]) {
                    break;
                } else {
                    st.pop();
                    if (st.isEmpty()) {
                        st.push(asteroids[i]);
                        break;
                    }
                }
            }
        }
        int[] res = new int[st.size()];
        for (int i = st.size() - 1; i >= 0; i--) {
            res[i] = st.pop();
        }
        return res;
    }
}
// Sum of subarray ranges
// Remove k Digits
class Solution {
    public String removeKdigits(String num, int k) {
        if (num.length() == k) {
            return "0";
        }
        if (k == 0) {
            return num;
        }
        String result = "";
        Stack<Character> st = new Stack<>();
        st.push(num.charAt(0));
        for (int i = 1; i < num.length(); i++) {
            while (k > 0 && !st.isEmpty() && num.charAt(i) < st.peek()) {
                --k;
                st.pop();
            }
            st.push(num.charAt(i));
            if (st.size() == 1 && num.charAt(i) == '0') {
                st.pop();
            }
        }
        while (k > 0 && !st.isEmpty()) {
            --k;
```

```
            st.pop();
        }
        while (!st.isEmpty()) {
            result += st.pop();
        }
        StringBuilder s1 = new StringBuilder(result);
        String s2 = s1.reverse().toString();
        if (s2.length() == 0) {
            return "0";
        }
        return s2;

    }
}

// Largest rectangle in a histogram
class Solution {
    public int largestRectangleArea(int[] arr) {
        int n = arr.length;
        Stack<Integer> st1 = new Stack<>();
        Stack<Integer> st2 = new Stack<>();
        int nsel[] = new int[n];
        int nser[] = new int[n];
        for (int i = 0; i < n; i++) {
            while (!st1.isEmpty() && arr[i] < arr[st1.peek()]) {
                nser[st1.peek()] = i;
                st1.pop();
            }
            st1.push(i);
        }
        while (!st1.isEmpty()) {
            nser[st1.pop()] = n;
        }
        for (int i = n - 1; i >= 0; i--) {
            while (!st2.isEmpty() && arr[i] < arr[st2.peek()]) {
                nsel[st2.peek()] = i;
                st2.pop();
            }
            st2.push(i);
        }
        while (!st2.isEmpty()) {
            nsel[st2.pop()] = -1;
        }
        int max = 0;
        for (int i = 0; i < n; i++) {
            int a = (nser[i] - nsel[i] - 1) * arr[i];
            max = Math.max(a, max);
        }
        return max;
```

```
    }
}
// Maximal Rectangles
class Solution {
    public int maximalRectangle(char[][] matrix) {
        int n = matrix.length;
        int m = matrix[0].length;
        int height[] = new int[m];
        int maxarea = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (matrix[i][j] == '1') {
                    height[j] += 1;
                } else {
                    height[j] = 0;
                }
            }
            int area = f(height, m);
            maxarea = Math.max(area, maxarea);
        }
        return maxarea;
    }

    public int f(int arr[], int n) {
        // printArr(arr);
        Stack<Integer> st1 = new Stack<>();
        Stack<Integer> st2 = new Stack<>();
        int nsel[] = new int[n];
        int nser[] = new int[n];
        for (int i = 0; i < n; i++) {
            while (!st1.isEmpty() && arr[i] < arr[st1.peek()]) {
                nsel[st1.peek()] = i;
                st1.pop();
            }
            st1.push(i);
        }
        while (!st1.isEmpty()) {
            nsel[st1.pop()] = n;
        }
        for (int i = n - 1; i >= 0; i--) {
            while (!st2.isEmpty() && arr[i] < arr[st2.peek()]) {
                nser[st2.peek()] = i;
                st2.pop();
            }
            st2.push(i);
        }
        while (!st2.isEmpty()) {
            nser[st2.pop()] = -1;
        }
```

```
        int max = 0;
        for (int i = 0; i < n; i++) {
            int a = (nsel[i] - nser[i] - 1) * arr[i];
            max = Math.max(a, max);
        }
        return max;
    }

}
// Sliding Window maximum
class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
        int n = nums.length;
        int nge[] = new int[n];
        int res[] = new int[n - k + 1];
        nge = nextGreaterEleOnRight(nums, n);
        int i, j = 0;
        for (i = 0; i < n - k + 1; i++) {
            if (i > j) {
                j = i;
            }
            while (nge[j] < i + k) {
                j = nge[j];
            }
            res[i] = nums[j];
        }
        return res;
    }

    public int[] nextGreaterEleOnRight(int arr[], int n) {
        Stack<Integer> st = new Stack<>();
        int nge[] = new int[n];
        for (int i = 0; i < n; i++) {
            while (st.size() > 0 && arr[i] > arr[st.peek()]) {
                nge[st.peek()] = i;
                st.pop();
            }
            st.push(i);
        }
        while (st.size() > 0) {
            nge[st.peek()] = n;
            st.pop();
        }
        return nge;
    }
}

// Stock span problem
class Solution {
```

```
    // Function to calculate the span of stockâ€™s price for all n days.
    public static int[] calculateSpan(int price[], int n) {
        // Your code here
        int S[] = new int[n];
        Stack<Integer> st = new Stack<>();
        st.push(0);
        S[0] = 1;
        for (int i = 1; i < n; i++) {
            while (!st.isEmpty() && price[st.peek()] <= price[i]) {
                st.pop();
            }
            S[i] = (st.isEmpty()) ? (i + 1) : (i - st.peek());
            st.push(i);
        }
        return S;
    }

}

// The Celebrity Problem
class Solution {
    // Function to find if there is a celebrity in the party or not.
    int celebrity(int M[][], int n) {
        // code here
        Stack<Integer> st = new Stack<>();
        for (int i = 0; i < n; i++) {
            st.push(i);
        }
        while (st.size() > 1) {
            int A = st.pop();
            int B = st.pop();
            if (M[A][B] == 0) {
                st.push(A);
            } else if (M[B][A] == 0) {
                st.push(B);
            }
        }
        int cel = st.peek();
        int c = 0, c1 = 0;
        for (int i = 0; i < n; i++) {
            if (M[i][cel] == 1) {
                c++;
            }
        }
        for (int j = 0; j < n; j++) {
            if (M[cel][j] == 0) {
                c1++;
            }
        }
```

```
            return c == n - 1 && c1 == n ? cel : -1;

        }
    }
    // LRU cache (IMPORTANT)
    // LFU cache
}
```