Stacks

Assignment Questions

**Q1. Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing
only 1's and return its area.**

Example1:
Input:
[["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

Output matrix:6

Example 2:
Input: matrix = [["0"]]
Output:0

Example 3:
Input: matrix = [["1"]]
output:1

Ans:
```
public class MaxRectangle {
    public int maximalRectangle(char[][] matrix) {
        if (matrix == null || matrix.length == 0) return 0;

        int maxArea = 0;
        int[] heights = new int[matrix[0].length];

        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                heights[j] = matrix[i][j] == '1' ? heights[j] + 1 : 0;
            }
            maxArea = Math.max(maxArea, maxAreaInHist(heights));
        }

        return maxArea;
    }

    private int maxAreaInHist(int[] heights) {
        int n = heights.length;
        int[] stack = new int[n + 1];
        int top = -1;
        int maxArea = 0;

        for (int i = 0; i <= n; i++) {
            int height = i == n ? 0 : heights[i];
            while (top >= 0 && heights[stack[top]] >= height) {
                int h = heights[stack[top--]];
                int w = top < 0 ? i : i - stack[top] - 1;
                maxArea = Math.max(maxArea, h * w);
            }
            stack[++top] = i;
        }

        return maxArea;
    }
}
```

**Q2. Given an encoded string, return its decoded string. The encoding rule is: k[encoded_string],
where the encoded_string inside the square brackets is being repeated exactly k times. Note
that k is guaranteed to be a positive integer.**

You may assume that the input string is always valid; there are no extra white spaces, square brackets are well-formed, etc. Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there will not be input like 3a or 2[4].

Example 1:
Input s = "3[a]2[bc]"
Output:"aaabcbc"

Example 2:
Input s = "3[a2[c]]"
Output:"accaccacc"

Example 3:
Input s ="2[abc]3[cd]ef"
output:"abcabccdcdcdef"

Ans:

```
public class DecodeString {
    public String decodeString(String s) {
        Stack<Integer> countStack = new Stack<>();
        Stack<String> resStack = new Stack<>();
        String res = "";
        int k = 0;

        for (int i = 0; i < s.length(); i++) {
            if (Character.isDigit(s.charAt(i))) {
                k = k * 10 + s.charAt(i) - '0';
            } else if (s.charAt(i) == '[') {
```

```
            countStack.push(k);
            resStack.push(res);
            res = "";
            k = 0;
        } else if (s.charAt(i) == ']') {
            int count = countStack.pop();
            String temp = resStack.pop();
            for (int j = 0; j < count; j++) {
                temp += res;
            }
            res = temp;
        } else {
            res += s.charAt(i);
        }
    }

    return res;
}
```

Q3. You are keeping the scores for a baseball game with strange rules. At the beginning of the
game, you start with an empty record.
You are given a list of strings operations, where operations[i] is the ith operation you must apply
to the record and is one of the following:
An integer x.
Record a new score of x.
'+'.
Record a new score that is the sum of the previous two scores.
'D'.
Record a new score that is the double of the previous score.
'C'.
Invalidate the previous score, removing it from the record.
Return the sum of all the scores on the record after applying all the operations.

Example 1:
input:ops = ["5","2","C","D","+"]
Output:30

Explanation:
"5" - Add 5 to the record, record is now [5].
"2" - Add 2 to the record, record is now [5, 2].
"C" - Invalidate and remove the previous score, record is now [5].
"D" - Add 2 * 5 = 10 to the record, record is now [5, 10].
"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].
The total sum is 5 + 10 + 15 = 30.

Example 2:
input:ops = ["5","-2","4","C","D","9","+","+"]
Output:27

Explanation:
"5" - Add 5 to the record, record is now [5].
"-2" - Add -2 to the record, record is now [5, -2].
"4" - Add 4 to the record, record is now [5, -2, 4].
"C" - Invalidate and remove the previous score, record is now [5, -2].
"D" - Add 2 * -2 = -4 to the record, record is now [5, -2, -4].
"9" - Add 9 to the record, record is now [5, -2, -4, 9].
"+" - Add -4 + 9 = 5 to the record, record is now [5, -2, -4, 9, 5].
"+" - Add 9 + 5 = 14 to the record, record is now [5, -2, -4, 9, 5, 14].
The total sum is 5 + -2 + -4 + 9 + 5 + 14 = 27.

Example 2:
input:ops = ["1","C"]
Output:0

Explanation:
"1" - Add 1 to the record, record is now [1].

"C" - Invalidate and remove the previous score, record is now [].
Since the record is empty, the total sum is 0.

Ans:

```
public class BaseballGame {
    public int calPoints(String[] ops) {
        Stack<Integer> stack = new Stack<>();

        for (String op : ops) {
            if (op.equals("+")) {
                int top = stack.pop();
                int secondTop = stack.pop();
                stack.push(secondTop);
                stack.push(top);
                stack.push(top + secondTop);
            } else if (op.equals("D")) {
                stack.push(2 * stack.peek());
            } else if (op.equals("C")) {
                stack.pop();
            } else {
                stack.push(Integer.parseInt(op));
            }
        }

        int sum = 0;
        while (!stack.isEmpty()) {
            sum += stack.pop();
        }

        return sum;
    }
}
```

Q4. We are given an array of asteroids of integers representing asteroids in a row.For each
asteroid, the absolute value represents its size, and the sign represents its direction (positive
meaning right, negative meaning left). Each asteroid moves at the same speed.
Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will
explode. If both are the same size, both will explode. Two asteroids moving in the same direction
will never meet.

Example1:
Input:asteroids = [5,10,-5]
Output:[5,10]

Example2:
Input:asteroids = [8,-8]
output:[]

Example3:
Input:asteroids = [10,2,-5]
output:[10]

Ans:

```
public class AsteroidCollision {
    public int[] asteroidCollision(int[] asteroids) {
        Stack<Integer> stack = new Stack<>();

        for (int asteroid : asteroids) {
            while (!stack.isEmpty() && asteroid < 0 && stack.peek() > 0) {
                if (stack.peek() < -asteroid) {
```

```
                stack.pop();
                continue;
            } else if (stack.peek() == -asteroid) {
                stack.pop();
            }
            break;
        }
        if (asteroid > 0 || stack.isEmpty() || stack.peek() < 0) {
            stack.push(asteroid);
        }
    }

    int[] result = new int[stack.size()];
    int i = stack.size() - 1;
    while (!stack.isEmpty()) {
        result[i--] = stack.pop();
    }

    return result;
  }
}
```

Q5. Given an array of integers temperatures represents the daily temperatures, return an array
answer such that answer[i] is the number of days you have to wait after the ith day to get a
warmer temperature. If there is no future day for which this is possible, keep answer[i] == 0
instead.

Example1:
Input:temperatures = [73,74,75,71,69,72,76,73]
output:[1,1,4,2,1,1,0,0]

Example2:
Input:temperatures = [30,40,50,60]
output:[1,1,1,0]

Example3:
Input:temperatures == [30,60,90]
output:[1,1,0]

Ans:

```
public class DailyTemperatures {
    public int[] dailyTemperatures(int[] temperatures) {
        int n = temperatures.length;
        int[] result = new int[n];
        Stack<Integer> stack = new Stack<>();

        for (int i = 0; i < n; i++) {
            while (!stack.isEmpty() && temperatures[i] >
temperatures[stack.peek()]) {
                int index = stack.pop();
                result[index] = i - index;
            }
            stack.push(i);
        }

        return result;
    }
}
```