INTERNSHALA

DATA STRUCTURE AND ALGORITHM

ASSIGNIMENT- 4

Problem 1

Valid parenthesis

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.An input string is valid if:Open brackets must be closed by the same type of brackets.Open brackets must be closed in the correct order.Every close bracket has a corresponding open bracket of the same type.

Example 1

Input: s = "()"

Output: true

Example 2

Input: s = "()[]{}"

Output: true

Question link

https://leetcode.com/problems/valid-parentheses/description/

Code

```
// valid parenthesis


class Stack{
    constructor(){
        this.items=[];
```

```
    }

    push(element){
        this.items.push(element);
    }
    pop(){
        if(this.isEmpty()){
            return "underflow"
        }
        return this.items.pop();
    }
    top(){
        if(this.isEmpty()){
            return "Stack is empty";
        }
        return this.items[this.items.length -1];
    }
    isEmpty(){
        return this.items.length==0;
    }
    size(){
        return this.items.length;
    }
}

function validParenthesis(s){
    let stack=new Stack();
```

```
for(let i=0;i<s.length;i++){
    if(s[i]=='(' || s[i]=='{' || s[i]=='['){
        stack.push(s[i]);
    }
    else if(s[i]==')'){
        if(stack.isEmpty()){
            return false;

        }else{
            if(stack.top()!='('){
                return false;
            }else{
                stack.pop();
            }
        }
    }
    else if(s[i]=='}'){
        if(stack.isEmpty()){
            return false;

        }else{
            if(stack.top()!='{'){
                return false;
            }else{
                stack.pop();
            }
        }
```

```
        }
        else if(s[i]==']'){
            if(stack.isEmpty()){
                return false;

            }else{
              if(stack.top()!='['){
                  return false;
               }else{
                  stack.pop();
               }
            }
        }
    }
    if(!stack.isEmpty()){
        return false;
    }else{
        return true;
    }
}

let result=validParenthesis("( ))");
console.log(result);
```

DSA assign-4 > JS problem1.js > validParenthesis

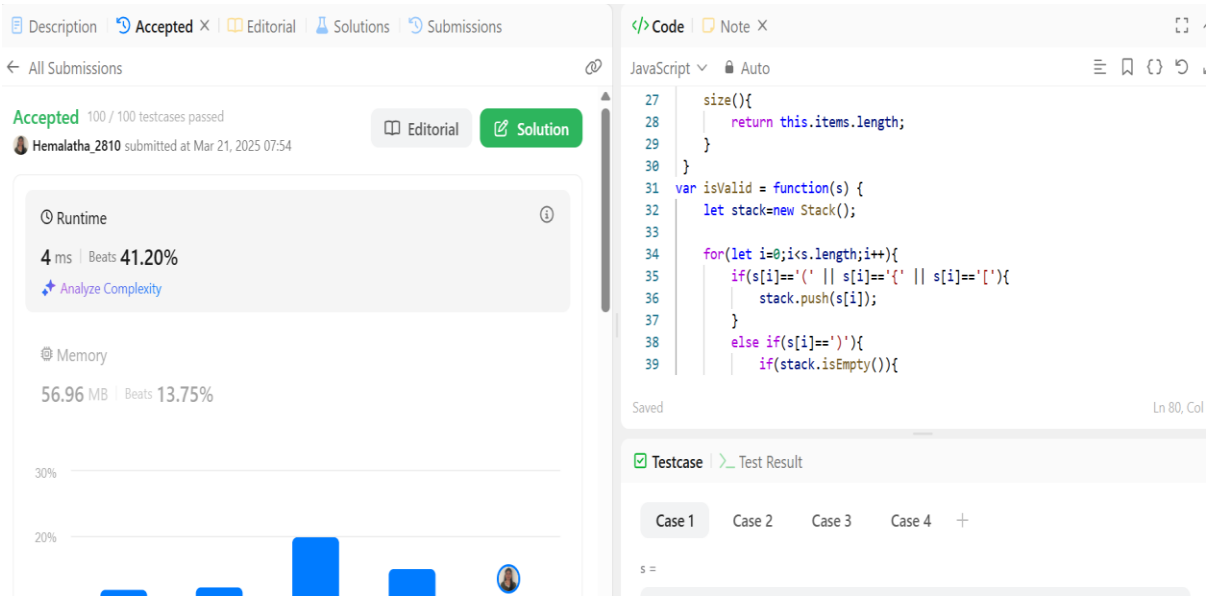
```
2
3   // valid parenthesis
4
5   class Stack{
6       constructor(){
7           this.items=[];
8       }
9
10      push(element){
11          this.items.push(element);
12      }
13      pop(){
14          if(this.isEmpty()){
15              return "underflow"
```

OUTPUT ··· Filter Code

```
[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-4\problem1.js"
true

[Done] exited with code=0 in 0.211 seconds

[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-4\problem1.js"
false
```

Submission link

https://leetcode.com/problems/valid-parentheses/submissions/1580852272/



Conclusion

Time complexity O(n)

Stack operation like pop( ), push( ), top( ),isEmpty( ) and Size( ) takes O(1) time to run the code . The loop runs n character and takes O(n)

$$O(n)+O(1)=O(n)$$

Space complexity O(n)

In worst-case scenario occurs when all characters in s are opening brackets (e.g., "(({{[["), causing all n elements to be stored in the stack.

Hence, the maximum stack size is **O(n)**.

Problem 2

The next greater element of some element x in an array is the first greater element that is to the right of x in the same array.You are given two distinct 0-indexed integer arrays nums1 and nums2, where nums1 is a subset of nums2.For each 0 <= i < nums1.length, find the index j such that nums1[i] == nums2[j] and determine the next greater element of nums2[j] in nums2. If there is no next greater element, then the answer for this query is -1.Return an array ans of length nums1.length such that ans[i] is the next greater element as described above.

Example

Input: nums1 = [4,1,2], nums2 = [1,3,4,2]

Output: [-1,3,-1]

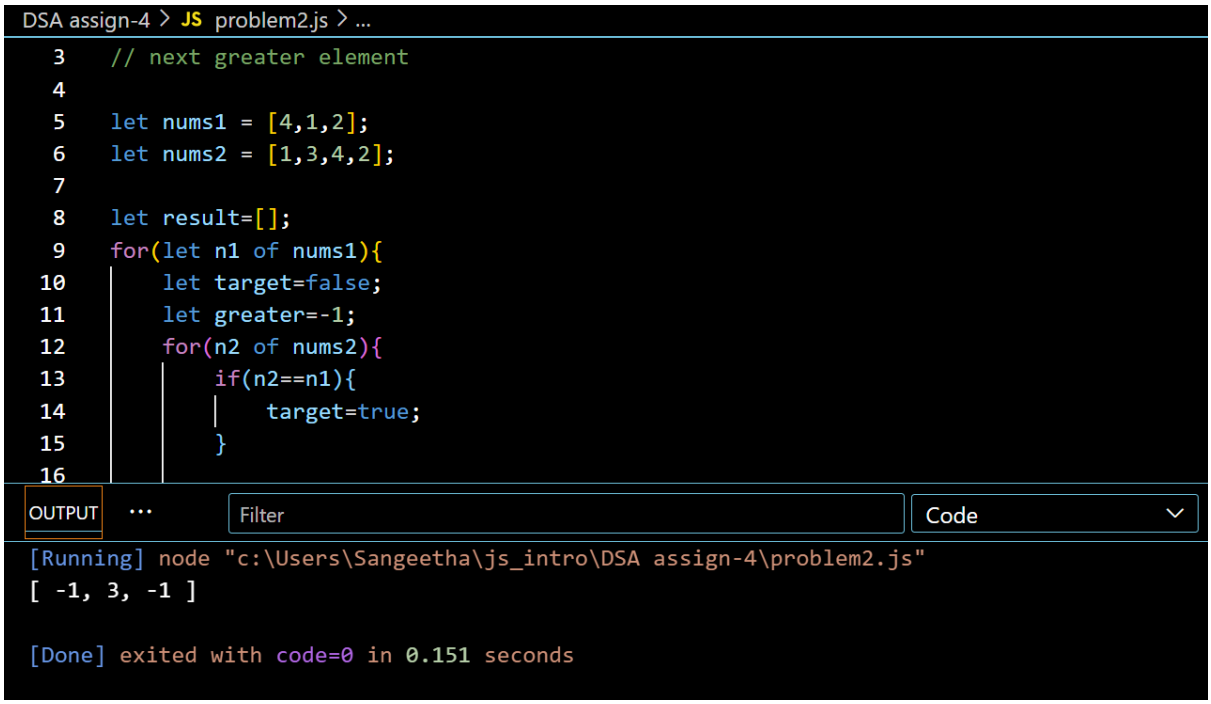
Question link

https://leetcode.com/problems/next-greater-element-i/description/

Code

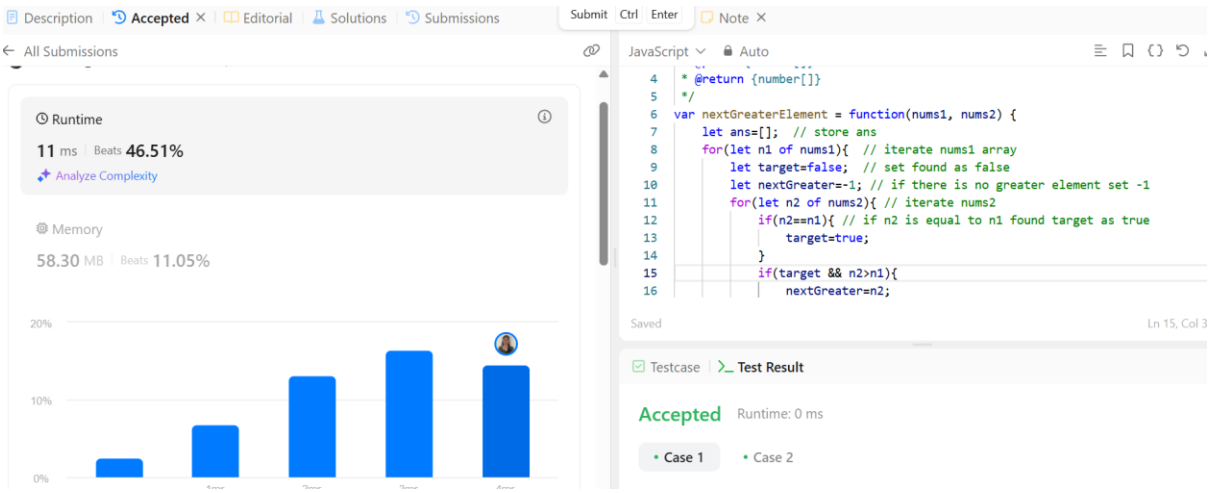
```
let nums1 = [4,1,2];

let nums2 = [1,3,4,2];

let result=[];

for(let n1 of nums1){

    let target=false;

    let greater=-1;

    for(n2 of nums2){

        if(n2==n1){

            target=true;
```

```
        }
        if(target && n2>n1){
            greater = n2;
            break;
        }
    }
    result.push(greater);
}
console.log(result);
```



Submission link

https://leetcode.com/problems/next-greater-element-i/submissions/1581032814/



Conclusion

Time complexity O(n*m)

- Iterating nums1 takes O(n) time complexity
- Iterating nums2 takes O(m)
- Overall time complexity is O(n*m)

Space complexity O(n)

The result array takes O(n) space complexity to run this code

Problem3

Remove-all-adjacent-duplicates

You are given a string s consisting of lowercase English letters. A duplicate removal consists of choosing two adjacent and equal letters and removing them

We repeatedly make duplicate removals on s until we no longer can.

Return the final string after all such duplicate removals have been made. It can be proven that the answer is unique.

Example 1:

Input: s = "abbaca"

Output: "ca"

Question link

https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string/description/

Code

```
let res = [];
    for (let char of s) {
        if (res.length > 0 && res[res.length - 1] === char) {
            res.pop();
        } else {
            res.push(char);
        }
    }
    console.log(res.join(""));
```

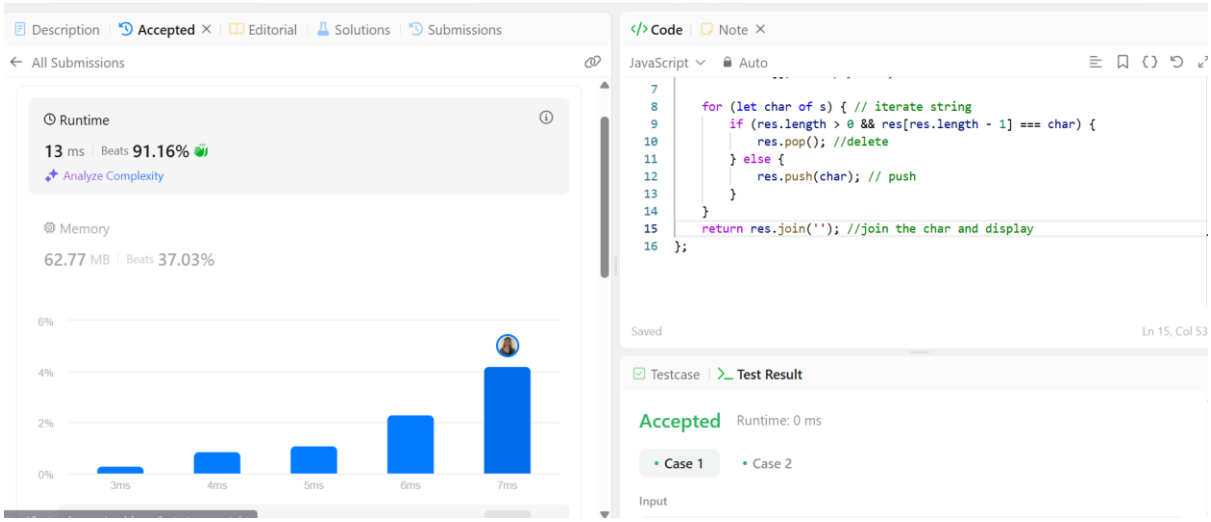
```
2
3
4   let s="abbaca";
5   let res = [];
6
7       for (let char of s) {
8           if (res.length > 0 && res[res.length - 1] === char) {
9               res.pop();
10          } else {
11              res.push(char);
12          }
13      }
14
15      console.log(res.join(""));
```

OUTPUT ··· Filter Code

```
[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-4\problem3.js"
ca

[Done] exited with code=0 in 0.168 seconds
```

Submission link

https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string/submissions/1581049552/



Conclusion

Time complexity O(n)

We iterates through the input str once so the loop runs n times

Each char of the str either pushed or popped.

Space complexity O(n)

We takes extra space to store the unique characters

Problem 4

Trapping rain water

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Question link

https://leetcode.com/problems/trapping-rain-water/description/

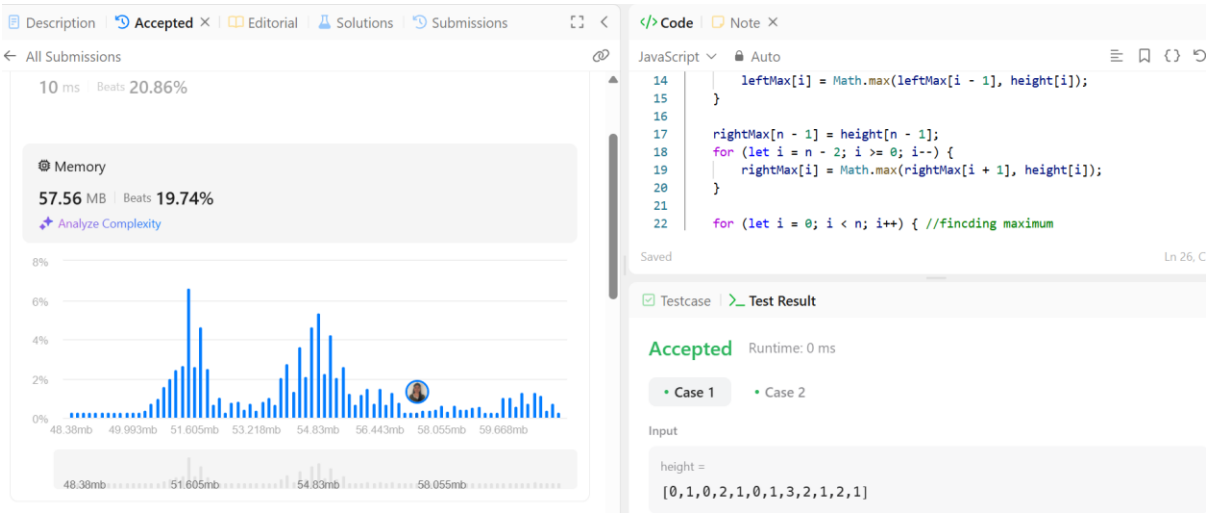
Code

```
function trap(height) {
    let n = height.length;
        let leftMax = new Array(n).fill(0);
    let rightMax = new Array(n).fill(0);
    let totalWater = 0;
    leftMax[0] = height[0];
    for (let i = 1; i < n; i++) {
        leftMax[i] = Math.max(leftMax[i - 1], height[i]);
    }
    rightMax[n - 1] = height[n - 1];
    for (let i = n - 2; i >= 0; i--) {
```

```
        rightMax[i] = Math.max(rightMax[i + 1], height[i]);
    }

    for (let i = 0; i < n; i++) {
        totalWater += Math.min(leftMax[i], rightMax[i]) - height[i];
    }

    return totalWater;
}
console.log("The amount of water trapped is :",trap([0,1,0,2,1,0,1,3,2,1,2,1]));
```

Submission link

https://leetcode.com/problems/trapping-rain-water/submissions/1583165167/



Conclusion

Time complexity O(n)

- Calculating left max,right max and amount of water trapped has taken N time to run this code

Space complexity O(n)

- Left max takes n space
- Right max takes n space

Problem 5

Largest rectangle in histogram

Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.

Example

Input: heights = [2,1,5,6,2,3]

Output: 10

Explanation: The above is a histogram where width of each bar is 1.

The largest rectangle is shown in the red area, which has an area = 10 units

Question link

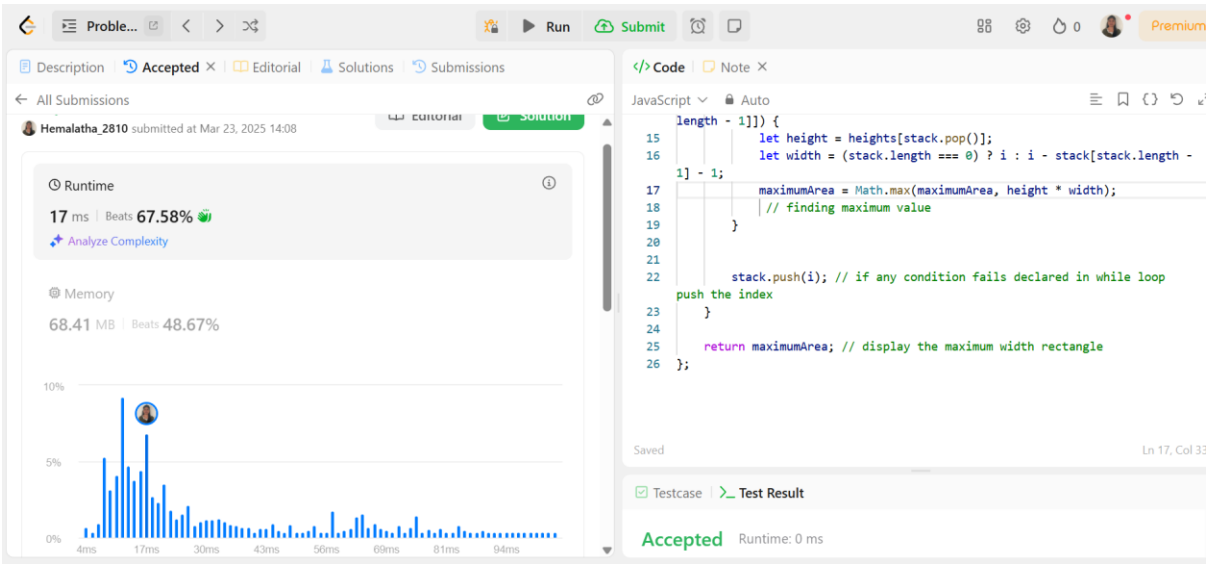
```
   for (let i = 0; i <= heights.length; i++) {
      let currentHeight = (i === heights.length) ? 0 : heights[i];
   while (stack.length > 0 && currentHeight < heights[stack[stack.length - 1]])
{
         let height = heights[stack.pop()];
         let width = (stack.length === 0) ? i : i - stack[stack.length - 1] - 1;
         maxArea = Math.max(maxArea, height * width);
      }
      stack.push(i);
   }
   return maxArea;
}
const heights = [2, 1, 5, 6, 2, 3];
console.log("The largest rectangle area is :"largestRectangleArea(heights));
```

```
2
3   // largerst rectangle
4
5   function largestRectangleArea(heights) {
6       let stack = [];
7       let maxArea = 0;
8
9
10      for (let i = 0; i <= heights.length; i++) {
11
12          let currentHeight = (i === heights.length) ? 0 : heights[i];
13
14
15          while (stack.length > 0 && currentHeight < heights[stack[stack.length - 1]]) {
```

```
OUTPUT ... Filter                                   Code
[Done] exited with code=0 in 0.153 seconds

[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-4\problem5.js"
The largest rectangle area is : 10

[Done] exited with code=0 in 0.143 seconds
```

Submission link

https://leetcode.com/problems/largest-rectangle-in-histogram/submissions/1583129515/



Conclusion

Time complexity

- Each bar (index) is pushed onto the stack once and popped from the stack once.
- The operation (push/pop) is O(1), the overall complexity is **O(n)**.

Space complexity O(n)

- The stack stores at most **n** elements