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

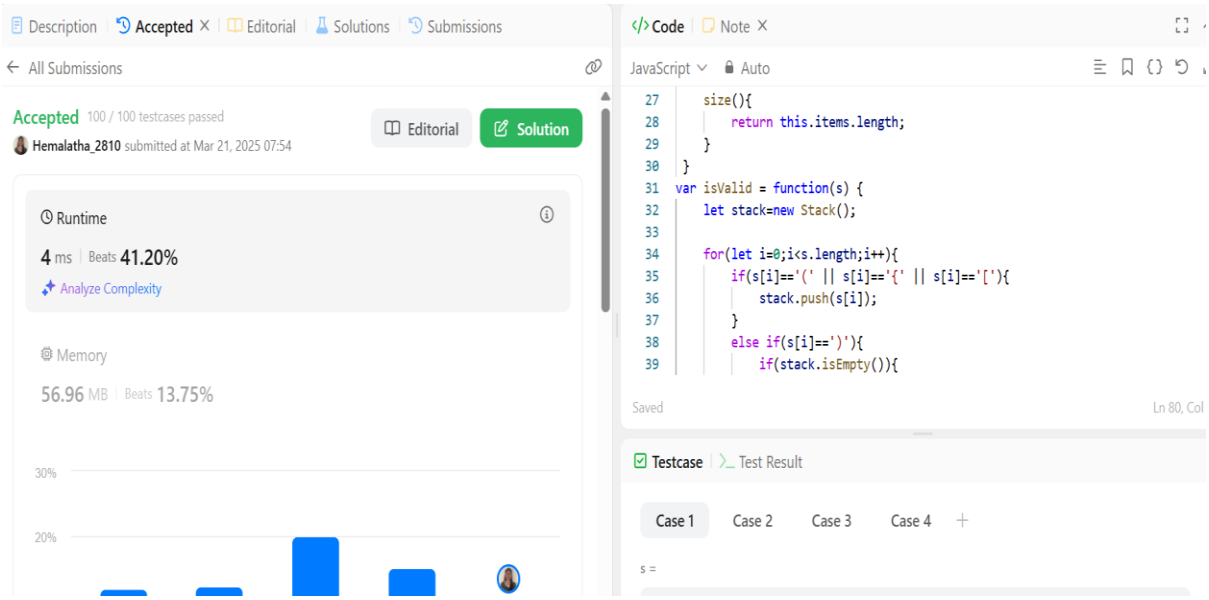
```js

// valid parenthesis

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

    push(element){
        this.items.push(element);
    }
    pop(){
        if(this.isEmpty()){
            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

Next greater elements

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

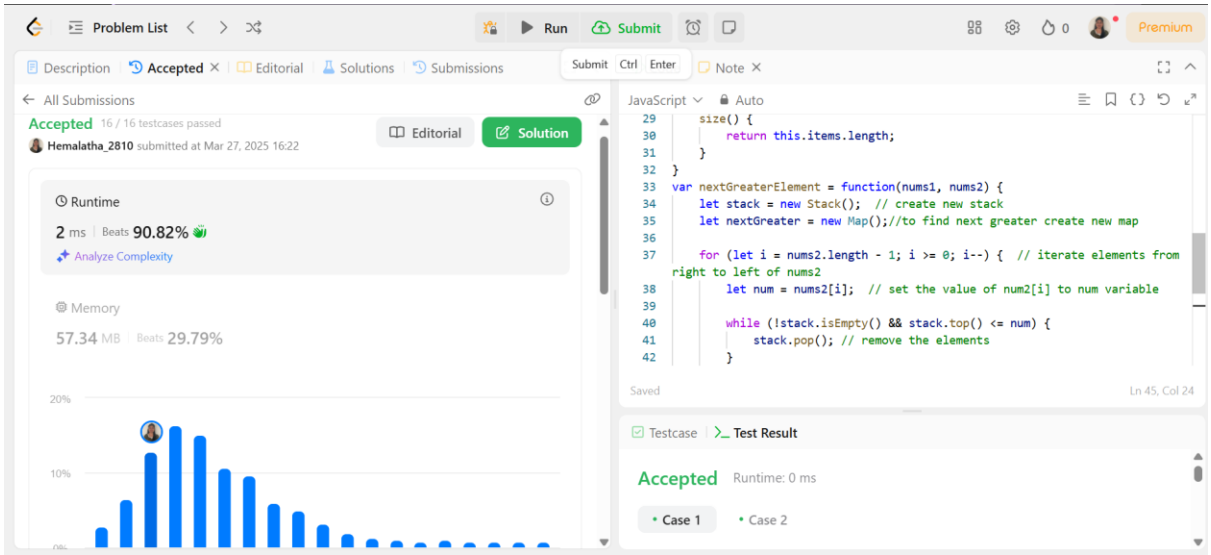
```
class Stack {
    constructor() {
        this.items = [];
    }
    push(element) {
        this.items.push(element);
    }
    pop() {
        if (this.isEmpty()) {
            return "underflow";
```

```
        }
        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;
    }
}

var nextGreaterElement = function(nums1, nums2) {
    let stack = new Stack();
    let nextGreaterMap = new Map();
    for (let i = nums2.length - 1; i >= 0; i--) {
        let num = nums2[i];
        while (!stack.isEmpty() && stack.top() <= num) {
            stack.pop();
        }
```

```
        if (stack.isEmpty()) {
            nextGreaterMap.set(num, -1);
        } else {
            nextGreaterMap.set(num, stack.top());
        }
        stack.push(num);
    }
    return nums1.map(num => nextGreaterMap.get(num));
};
console.log(nextGreaterElement([4,1,2], [1,3,4,2])); // Output: [-1, 3, -1]
```

Submission link

https://leetcode.com/problems/next-greater-element-i/submissions/1588019797/



Conclusion

Time complexity O(N+M)

- Iterate over nums2 **once** from right to left → **O(N)**. N is length of nums2
- M is length of nums1 and the result array by looking up values in nextGreater → **O(M)**
- O(N)+O(M) = O(N+M)

Space complexity O(N+M)

- Stack() and hashmap() takes O(N) space to store the elements
- Result array takes O(M) space to store M elements

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

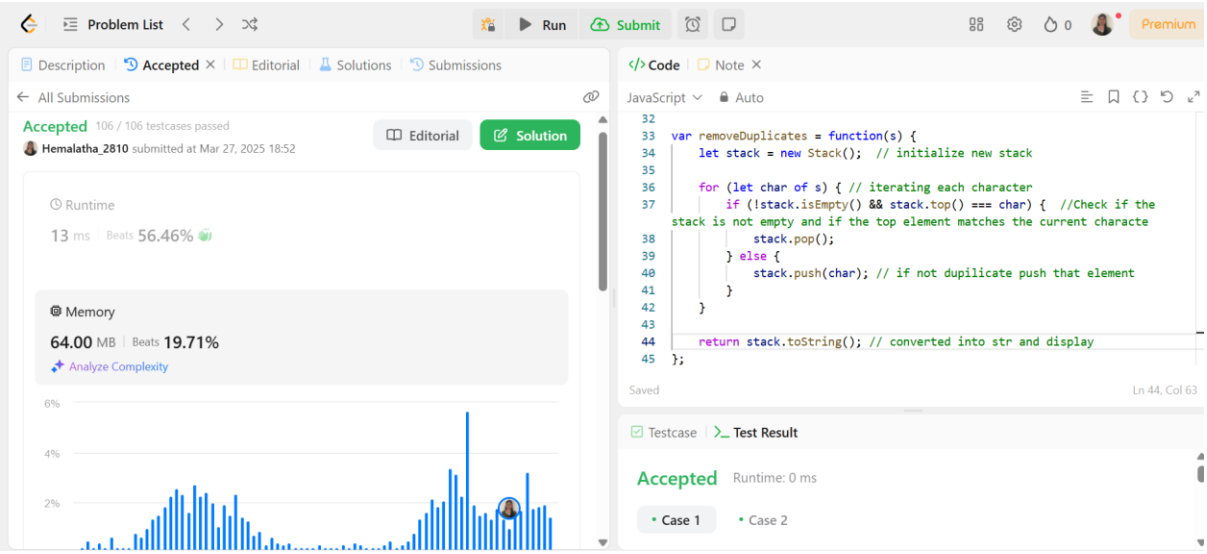
```
class Stack {
    constructor() {
        this.items = [];
    }
    push(element) {
        this.items.push(element);
    }
    pop() {
        if (!this.isEmpty()) {
            return this.items.pop();
```

```
        }
        return null;
    }
    top() {
        return this.isEmpty() ? null : this.items[this.items.length - 1];
    }
    isEmpty() {
        return this.items.length === 0;
    }
    size() {
        return this.items.length;
    }
    toString() {
        return this.items.join("");
    }
}

function removeAdjacentDuplicates(s) {
    let stack = new Stack();

    for (let char of s) {
        if (!stack.isEmpty() && stack.top() === char) {
            stack.pop();
        } else {
            stack.push(char);
        }
    }
```

```
    return stack.toString();
}


console.log("Unique elements:",removeAdjacentDuplicates("abbaca")); //
Output: "ca"
```

Submission link

https://leetcode.com/problems/remove-all-adjacent-duplicates-in-string/submissions/1588126834/



Conclusion

Time complexity O(n)

- We iterates through the input str once so the loop runs n times→ O(n)
- Each char of the str either pushed or popped. It takes O(1) for push and pop operation

Space complexity O(n)

- We takes extra space to store the unique characters .In the worst case, all n characters are stored in the stack →O(n)

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

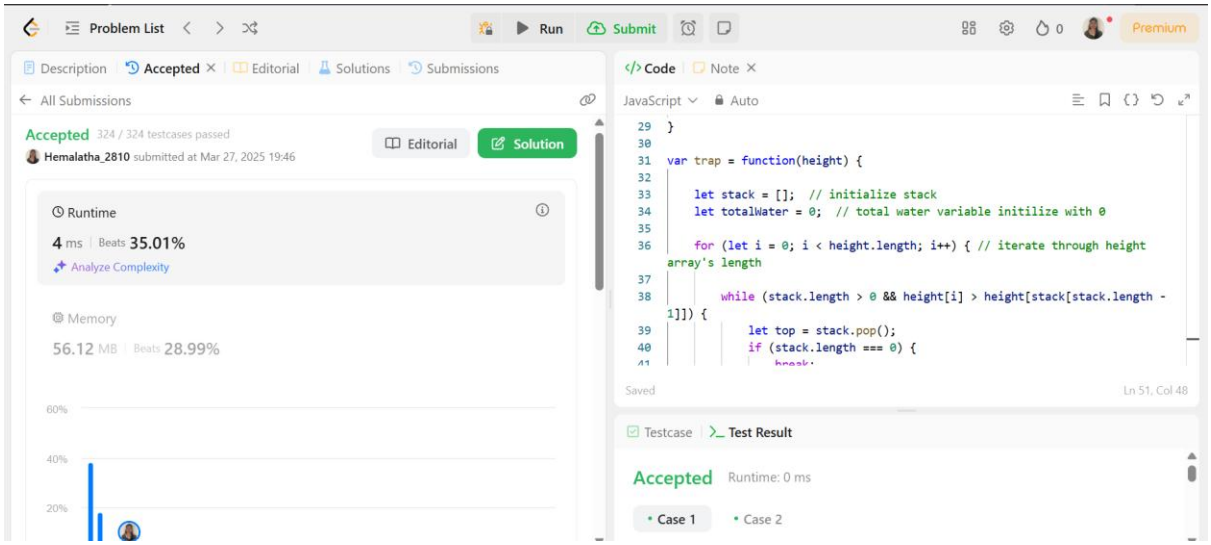
```
class Stack {
    constructor() {
        this.items = [];
    }
    push(element) {
        this.items.push(element);
    }
    pop() {
        if (!this.isEmpty()) {
            return this.items.pop();
        }
        return null;
```

```
    }
    top() {
        return this.isEmpty() ? null : this.items[this.items.length - 1];
    }
    isEmpty() {
        return this.items.length === 0;
    }
    size() {
        return this.items.length;
    }

}

function trap(height) {
    let stack = [];
    let totalWater = 0;

    for (let i = 0; i < height.length; i++) {

        while (stack.length > 0 && height[i] > height[stack[stack.length - 1]]) {
            let top = stack.pop();
            if (stack.length === 0) break;

            let distance = i - stack[stack.length - 1] - 1;
            let boundedHeight = Math.min(height[i], height[stack[stack.length - 1]])
- height[top]; // Effective height
            totalWater += distance * boundedHeight;
        }
```

```
        stack.push(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/1588179377/



Conclusion

Time complexity O(n)

- Each element goes into push or pop operation atleast once and for loop iterates through height array length → O(n)

Space complexity O(n)

- Stack method used in this problem →O(n)

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

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

Code

```
class Stack {
    constructor() {
        this.items = [];
    }
    push(element) {
        this.items.push(element);
    }
    pop() {
        if (!this.isEmpty()) {
            return this.items.pop();
        }
        return null;
    }
    top() {
```

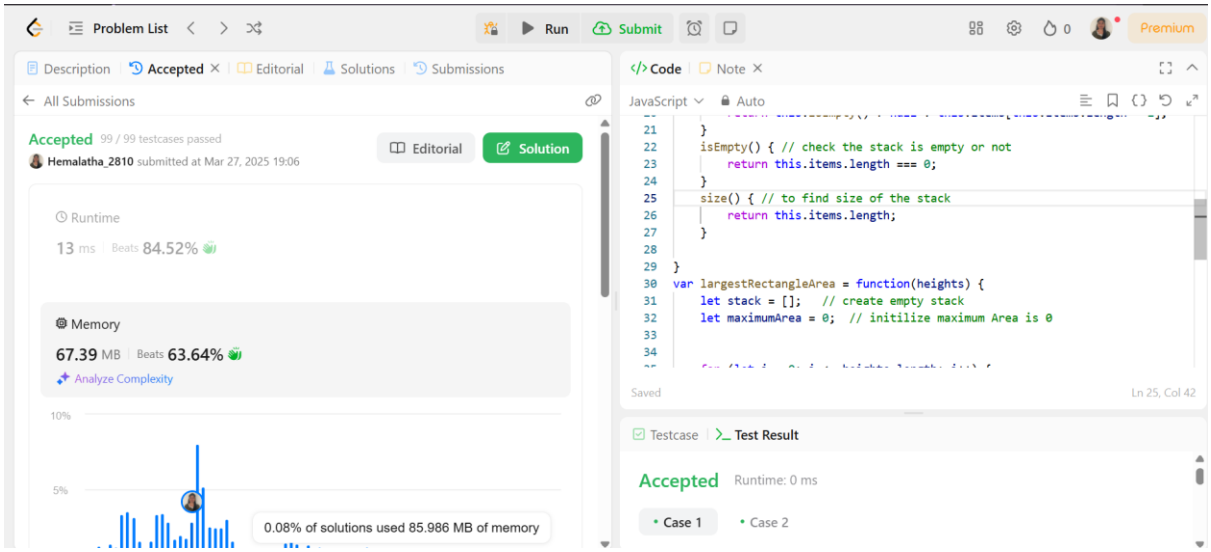
```
        return this.isEmpty() ? null : this.items[this.items.length - 1];
    }
    isEmpty() {
        return this.items.length === 0;
    }
    size() {
        return this.items.length;
    }

}

    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));
```

```javascript

// largerst rectangle

function largestRectangleArea(heights) {
    let stack = [];
    let maxArea = 0;


    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]]) {
```

```
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/1588140453/



Conclusion

- ✓ **Solved this problem by using stack**

Time complexity O(n)

- 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 and Worst-case scenario when all elements are pushed into the stack