



DATA STRUCTURE AND ALGORITHM

ASSIGNMENT- 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"
16        }
17    }
18}
19
20 true
21
22 [Done] exited with code=0 in 0.211 seconds
23
24 [Running] node "c:\Users\Sangeetha\js_intro\DSA assign-4\problem1.js"
25 false
```

Submission link

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

DescriptionAccepted XEditorialSolutionsSubmissions

All Submissions

Accepted 100 / 100 testcases passed
Hemalatha_2810 submitted at Mar 21, 2025 07:54

Runtime

4 ms | Beats 41.20%

Analyze Complexity

Memory

56.96 MB | Beats 13.75%

30%

20%

CodeNote X

JavaScriptAuto

27size(){
28return this.items.length;
29}
30}
31var isValid = function(s) {
32let stack=new Stack();
33
34for(let i=0;i<s.length;i++){
35if(s[i]=='(' || s[i]=='{' || s[i]=='['){
36stack.push(s[i]);
37}
38else if(s[i]==')' || s[i]=='}' || s[i]==']'){
39if(stack.isEmpty()){
40return false;
41}
42stack.pop();
43}
44}
45return stack.isEmpty();
46}
47}
48
49true
50

SavedLn 80, Col

TestcaseTest Result

Case 1Case 2Case 3Case 4+

s =

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 $0 \leq i < \text{nums1.length}$, 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]

```

The screenshot shows a VS Code editor window with a file named 'problem2.js'. The code defines a 'Stack' class with methods for constructor, push, pop, and top. It then uses this stack to calculate the next greater element for each element in an array [4, 1, 2] using a second array [1, 3, 4, 2]. The console output shows the result as [-1, 3, -1].

```

JS problem2.js > ...
2
3 // next greater element
4 class Stack {
5     constructor() {
6         this.items = [];
7     }
8     push(element) {
9         this.items.push(element);
10    }
11    pop() {
12        if (this.isEmpty()) {
13            return "underflow";
14        }
15        return this.items.pop();
16    }
17    top() {
18        if (this.isEmpty()) {

```

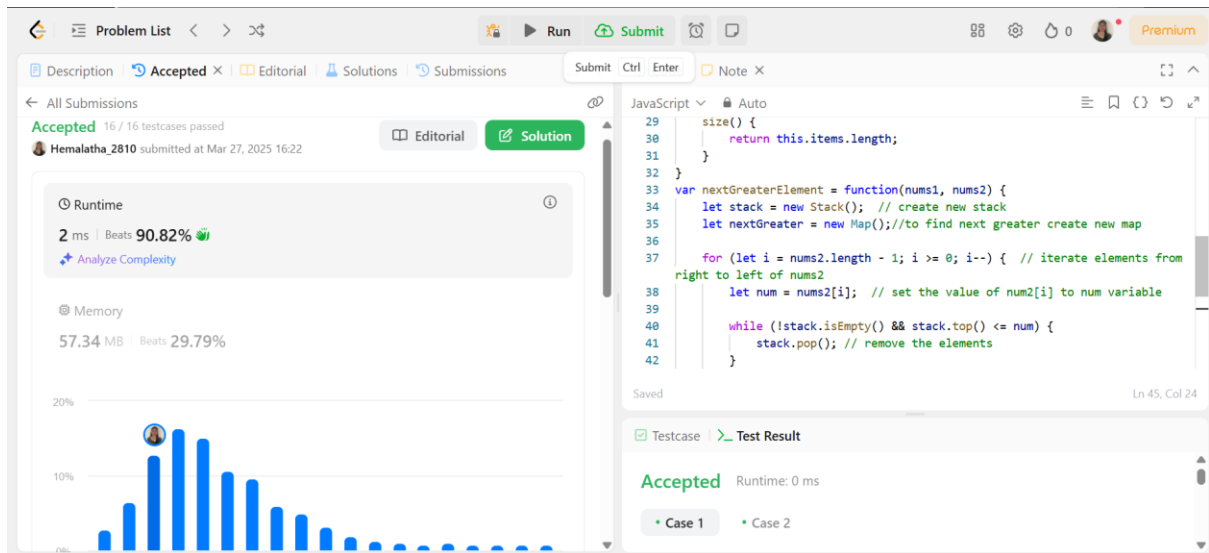
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter Code

[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-4\problem2.js"

[-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 $\rightarrow O(N)$. N is length of `nums2`
- M is length of `nums1` and the result array by looking up values in `nextGreater` $\rightarrow O(M)$
- $O(N)+O(M) = O(N+M)$

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"
```

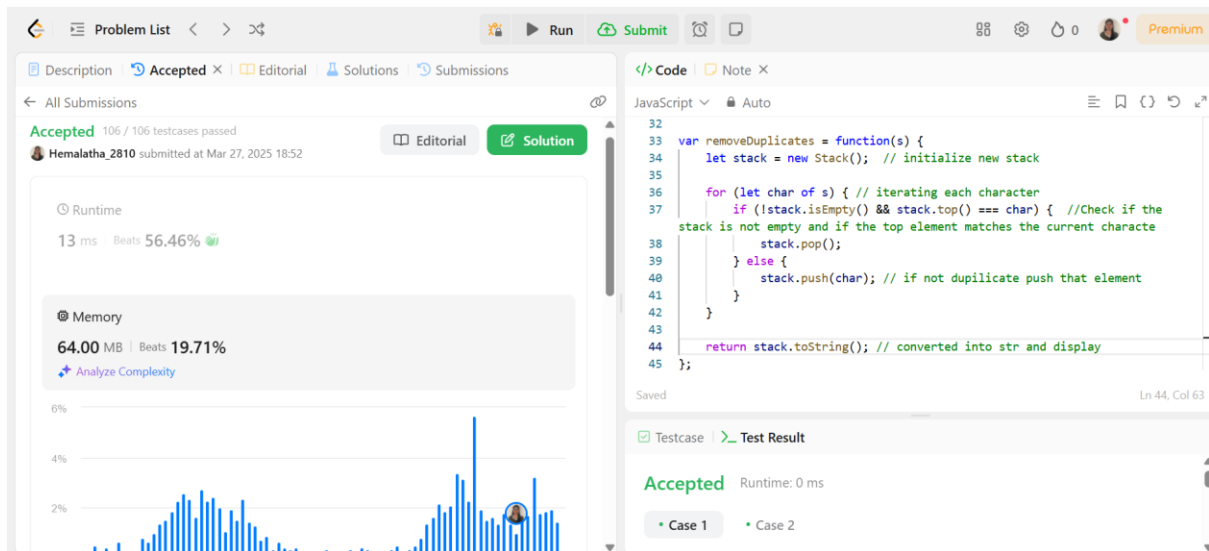


The screenshot shows a VS Code editor window with a file named 'problem3.js'. The code defines a 'Stack' class with methods: 'constructor()' (initializes 'this.items' to an empty array), 'push(element)' (adds an element to the stack), 'pop()' (removes and returns the top element, returning null if empty), and 'top()' (returns the top element or null if empty). The 'top()' method in the image contains a typo: 'return this.isEmpty() ? null : this.items[this.items.length - 1];'. Below the code editor, the 'OUTPUT' tab is selected, showing the console output: 'Unique elements: ca'. At the bottom, a status bar message says '[Done] exited with code=0 in 0.165 seconds'.

```
JS problem3.js > ...  
1  
2  
3 class Stack {  
4   constructor() {  
5     this.items = [];  
6   }  
7   push(element) {  
8     this.items.push(element);  
9   }  
10  pop() {  
11    if (!this.isEmpty()) {  
12      return this.items.pop();  
13    }  
14    return null;  
15  }  
16  top() {  
17    return this.isEmpty() ? null : this.items[this.items.length - 1];  
18  }  
19 }  
20  
21 console.log("Unique elements:",removeAdjacentDuplicates("abbaca")); //  
22  
23 Output: "ca"  
[Done] exited with code=0 in 0.165 seconds
```

Submission link

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



Conclusion

Time complexity $O(n)$

- We iterate through the input string once so the loop runs n times $\rightarrow O(n)$
- Each character of the string is either pushed or popped. It takes $O(1)$ for push and pop operation

Space complexity $O(n)$

- We take extra space to store the unique characters. In the worst case, all n characters are stored in the stack $\rightarrow O(n)$

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;
        }
    }
}

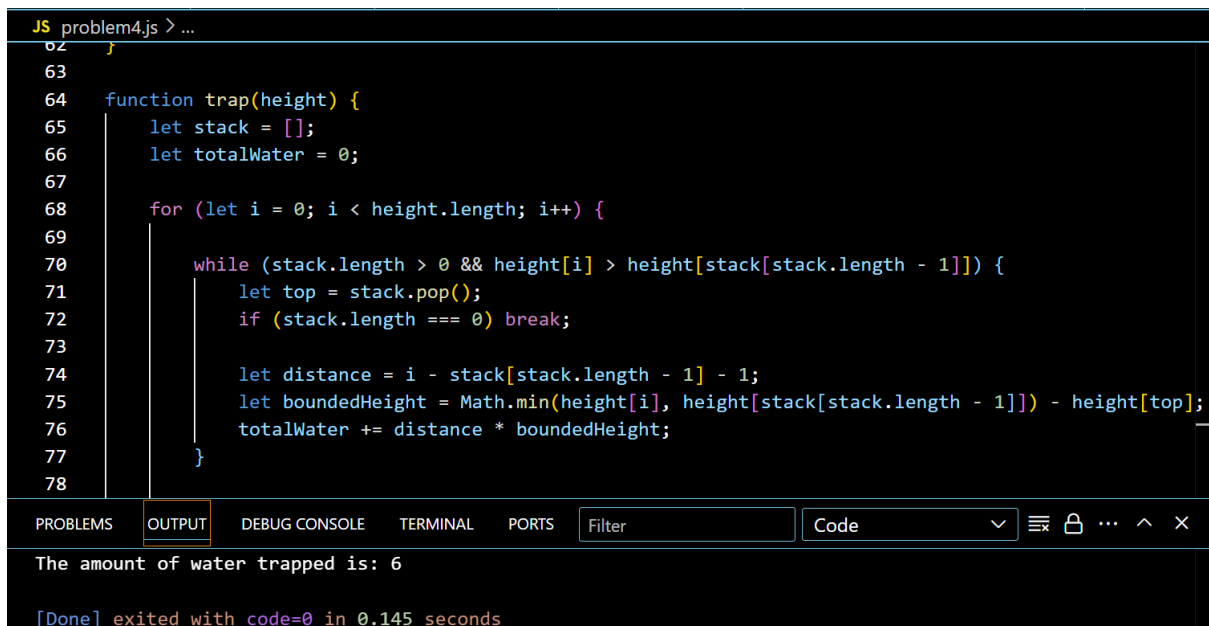
```



```
        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]));
```



```
JS problem4.js > ...
62 }
63
64 function trap(height) {
65     let stack = [];
66     let totalWater = 0;
67
68     for (let i = 0; i < height.length; i++) {
69
70         while (stack.length > 0 && height[i] > height[stack[stack.length - 1]]) {
71             let top = stack.pop();
72             if (stack.length === 0) break;
73
74             let distance = i - stack[stack.length - 1] - 1;
75             let boundedHeight = Math.min(height[i], height[stack[stack.length - 1]]) - height[top];
76             totalWater += distance * boundedHeight;
77         }
78     }
79 }

```

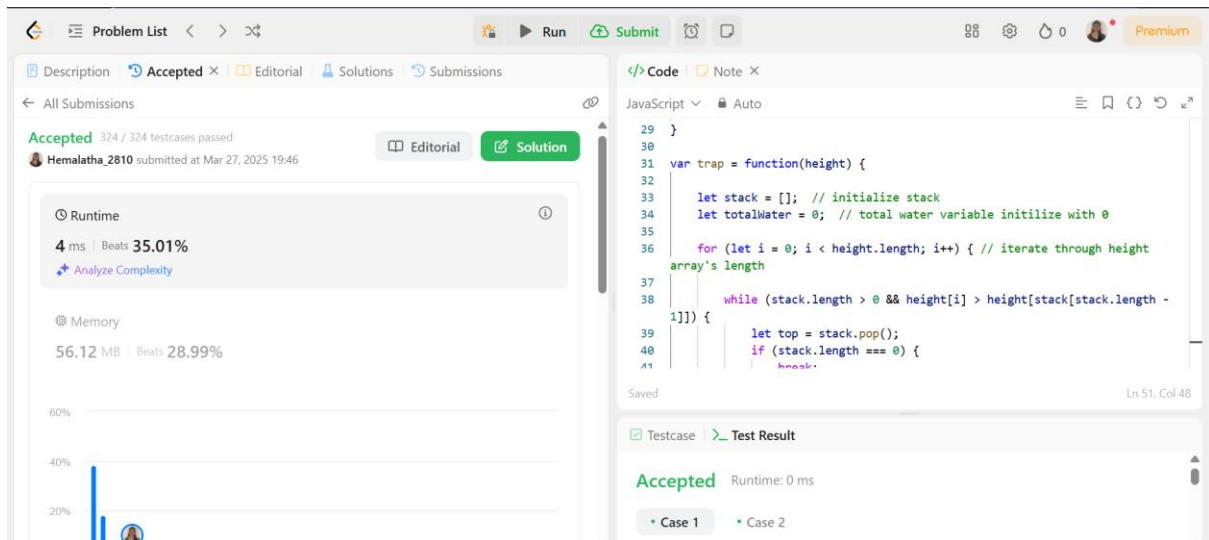
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter Code

The amount of water trapped is: 6

[Done] exited with code=0 in 0.145 seconds

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 $\rightarrow O(n)$

Space complexity $O(n)$

- Stack method used in this problem $\rightarrow O(n)$

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));

```

```
2
3 // largest 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]]) {
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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/>

Problem List < > 🔍

Run Submit 🕒 📄

Description Accepted x Editorial Solutions Submissions

← All Submissions

Accepted 99 / 99 testcases passed

Hemalatha_2810 submitted at Mar 27, 2025 19:06

Editorial Solution

Runtime

13 ms Beats 84.52%

Memory

67.39 MB Beats 63.64%

Analyze Complexity

0.08% of solutions used 85,986 MB of memory

JavaScript

```
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Saved Ln 25, Col 42

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

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