1762. Buildings With an Ocean View

Medium 839 112 Add to List Share

There are `n` buildings in a line. You are given an integer array `heights` of size `n` that represents the heights of the buildings in the line.

The ocean is to the right of the buildings. A building has an ocean view if the building can see the ocean without obstructions. Formally, a building has an ocean view if all the buildings to its right have a **smaller** height.

Return a list of indices **(0-indexed)** of buildings that have an ocean view, sorted in increasing order.

Example 1:

```
Input: heights = [4,2,3,1]
Output: [0,2,3]
Explanation: Building 1 (0-indexed) does not have an ocean view
because building 2 is taller.
```

Example 2:

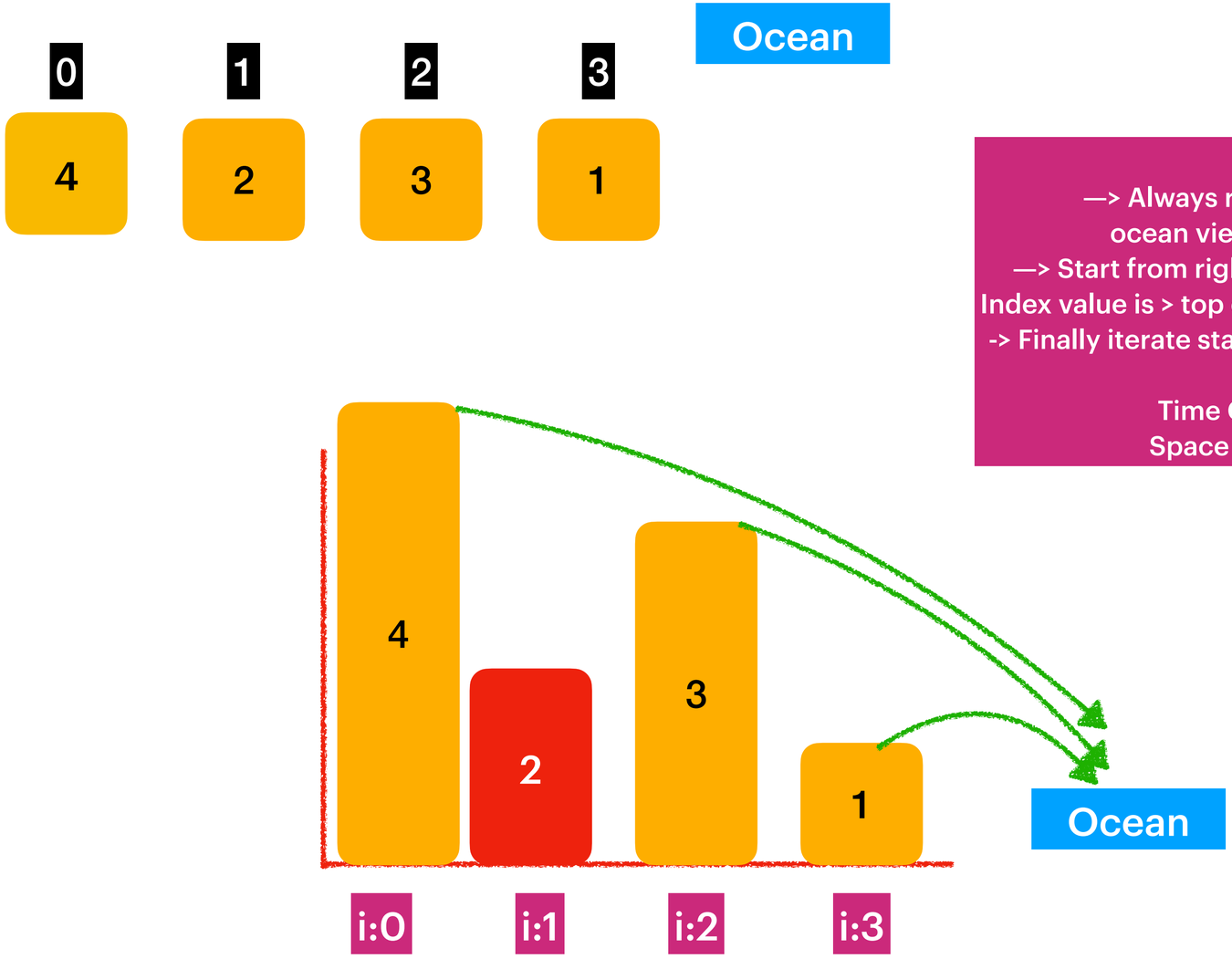
```
Input: heights = [4,3,2,1]
Output: [0,1,2,3]
Explanation: All the buildings have an ocean view.
```

Example 3:

```
Input: heights = [1,3,2,4]
Output: [3]
Explanation: Only building 3 has an ocean view.
```

Constraints:

- `1 <= heights.length <= 10^5`
- `1 <= heights[i] <= 10^9`



Algo :
—> Always rightmost index has the ocean view so push into stack.
—> Start from rightMostindex-1 , if the current Index value is > top of the stack then push to stack.
-> Finally iterate stack elements then add to array.

Time Complexity : O(n)
Space Complexity : O(n)

The Flat Index have ocean view are {0,2,3}

1249. Minimum Remove to Make Valid Parentheses

Medium 4919 85 Add to List Share

Given a string `s` of `'('` , `')'` and lowercase English characters.

Your task is to remove the minimum number of parentheses (`'('` or `')'`, in any positions) so that the resulting *parentheses string* is valid and return **any** valid string.

Formally, a *parentheses string* is valid if and only if:

- It is the empty string, contains only lowercase characters, or
- It can be written as `AB` (`A` concatenated with `B`), where `A` and `B` are valid strings, or
- It can be written as `(A)`, where `A` is a valid string.

Example 1:

```
Input: s = "lee(t(c)o)de)"
Output: "lee(t(c)o)de"
Explanation: "lee(t(co)de)" , "lee(t(c)ode)" would also be accepted.
```

Example 2:

```
Input: s = "a)b(c)d"
Output: "ab(c)d"
```

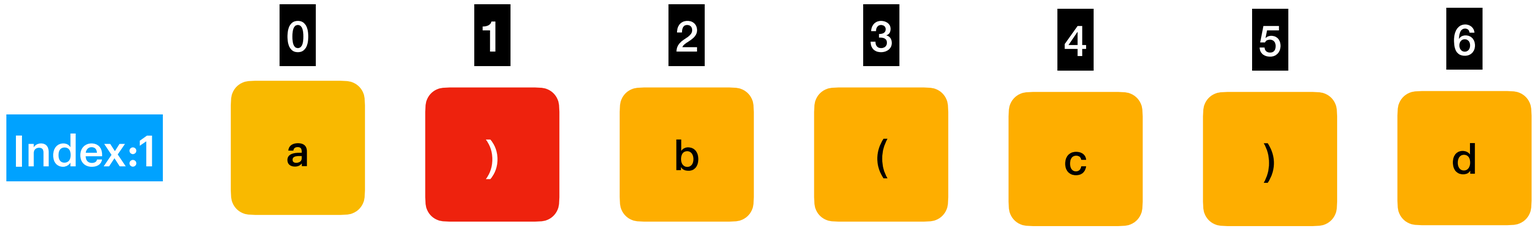
Example 3:

```
Input: s = "))(("
Output: ""
Explanation: An empty string is also valid.
```

Constraints:

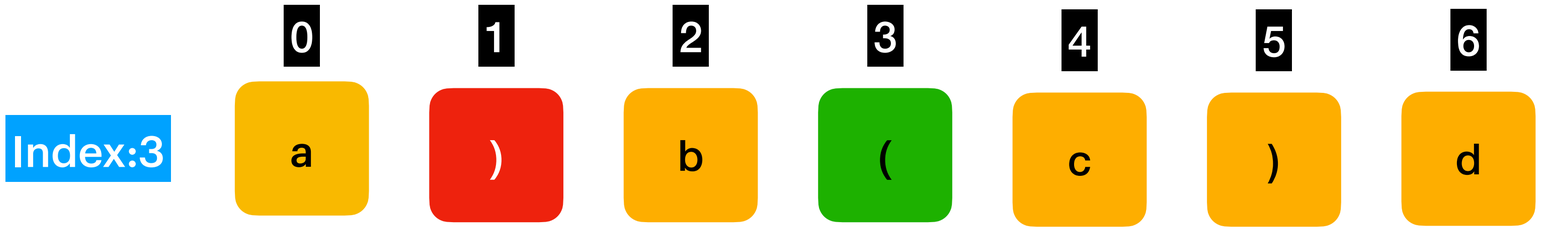
- `1 <= s.length <= 10^5`
- `s[i]` is either `'('` , `')'`, or lowercase English letter `.`

Input: a)b(c)d



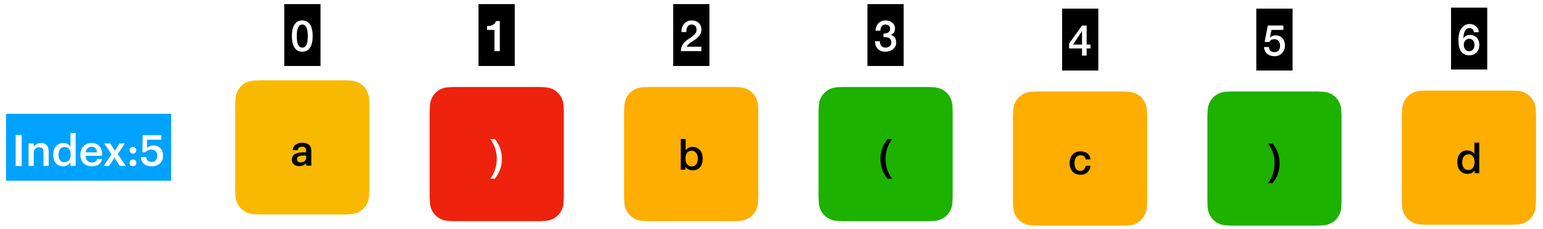
Stack: [ ]

InvalidParamsSet: [1]



Stack: [( ]

InvalidParamsSet: [1]



Stack: []

InvalidParamsSet: [1]

Input: a)b(c((

0	1	2	3	4	5	6
a	)	b	(	c	(	(

Index:1

Stack: [ ]

InvalidParamsSet: [1]

Algo : Clue
—> Copy invalid ‘)’ into Set.
-> If the stack is not empty then it has invalid ‘(‘ indexes so copy into Set
—> Using set eliminate invalid index’s then form a new String

0	1	2	3	4	5	6
a	)	b	(	c	(	(

Index:3

Stack: [3]

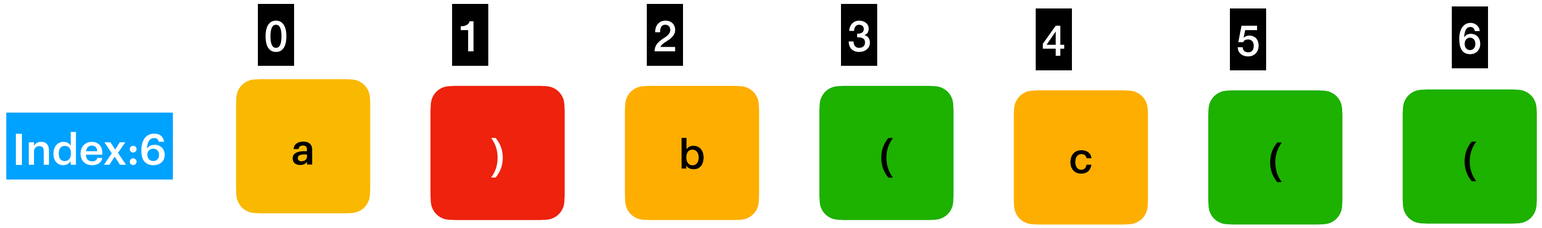
InvalidParamsSet: [1]

0	1	2	3	4	5	6
a	)	b	(	c	(	(

Index:5

Stack: [3,5]

InvalidParamsSet: [1]



Stack: [3,5,6]

InvalidParamsSet: [1]

As stack is not Empty —> add all the index’s to the InvalidParamsSet

InvalidParamsSet: [1,3,5,6]

After not considering invalid indexes then the String will be. —> abc

Time Complexity : O(n)

Space Complexity : O(n)