香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

# CSC3100 Data Structures Tutorial 5: Stack

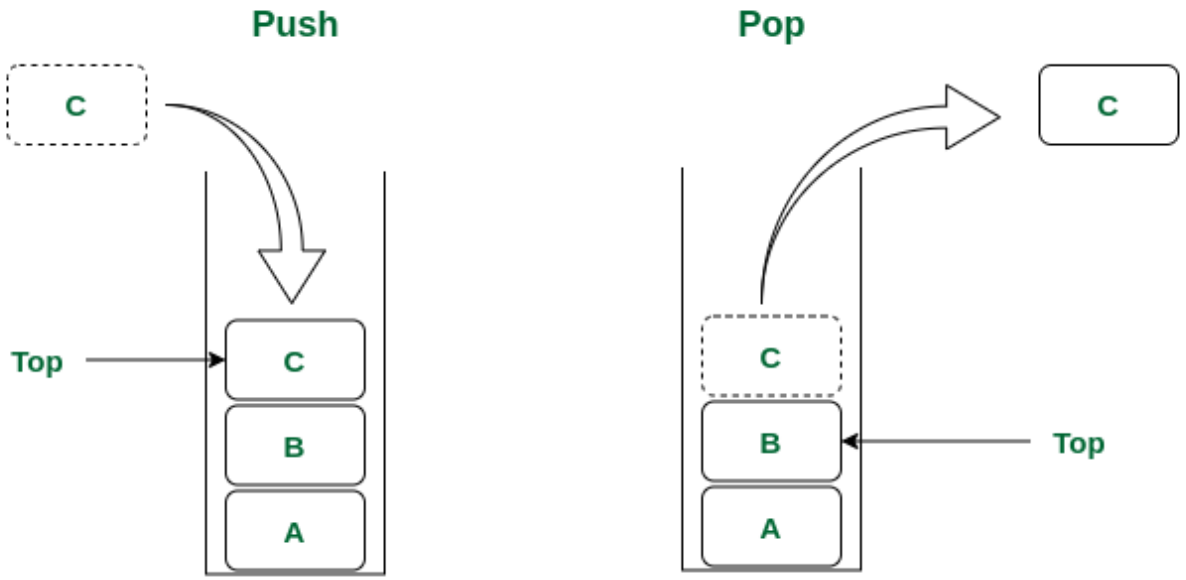
Wang Yaomin

# Contents

- Stack Concept
- Stack Application
- Stack Exercise

# Stack

## First-In, Last-Out (FILO)



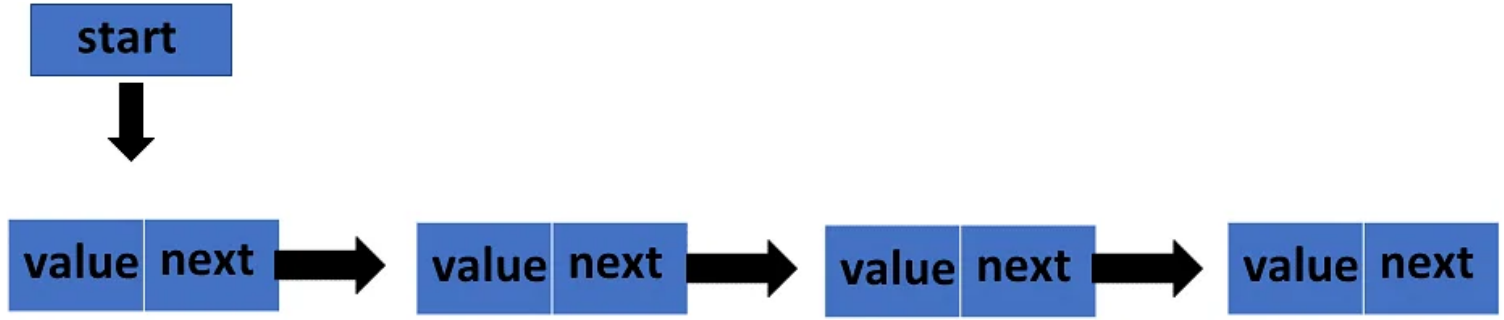

Stack Data Structure

## Basic operations

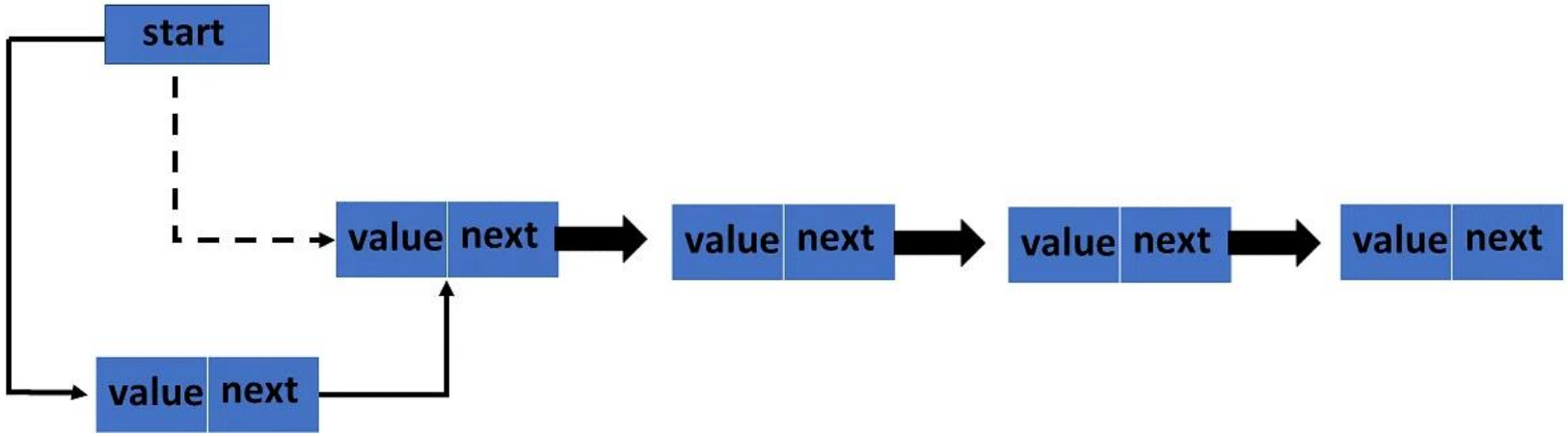
- pop ()
- push (i)
- top ()
- isEmpty()

# Implementations

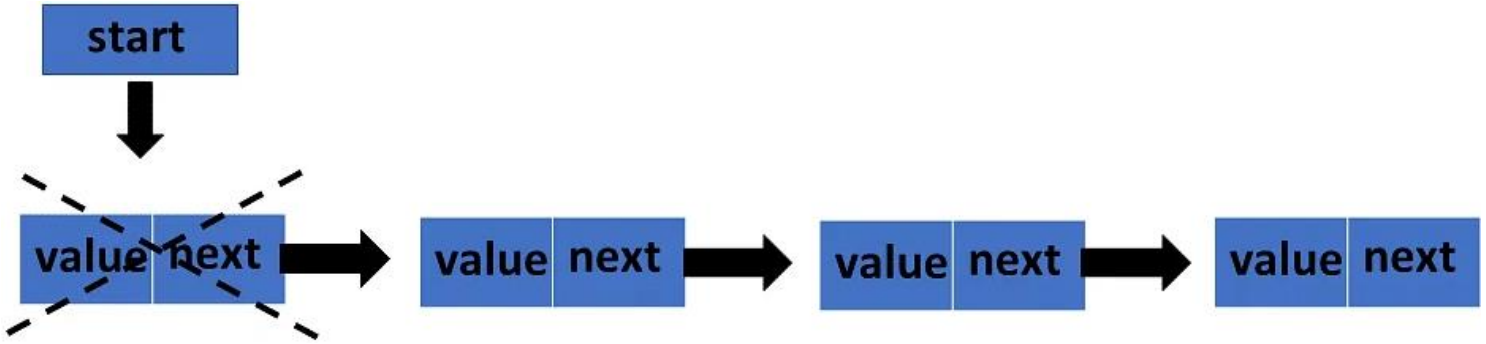
- Linked list




push()

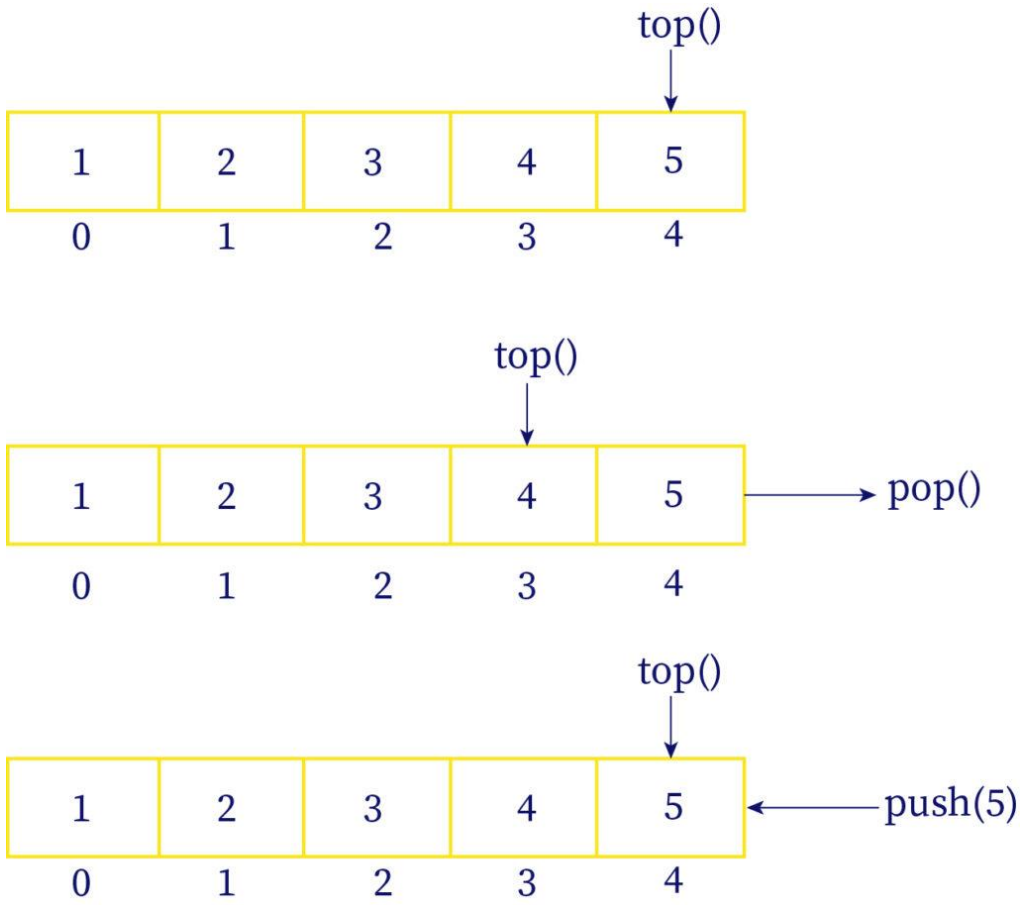



pop()



# Implementations

- Arrays




Comparison of two implementations：

·Using linked list saves space

(No waste memory allocation)

·Using array is faster

(Continuous memory allocation and load)

# Application 1:

·Parentheses Matching

((((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)

·Balanced symbol

({{}}) (){([])}]

# Application 2:

Using 2 stacks to sort a sequence of numbers(Monotonic Stack).

# Exercise 1: Remove K Digits (LeetCode P402)

Given string num representing a non-negative integer num, and an integer k, return the smallest possible integer after removing k digits from num

**Example 1:**

```
Input: num = "1432219", k = 3
Output: "1219"
Explanation: Remove the three digits 4, 3, and 2 to form the new number 1219 which is the smallest.
```

**Example 2:**

```
Input: num = "10200", k = 1
Output: "200"
Explanation: Remove the leading 1 and the number is 200. Note that the output must not contain leading zeroes.
```

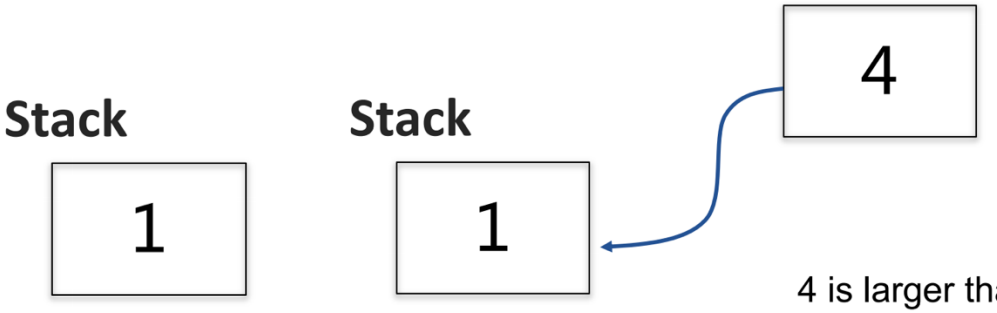
**Example 3:**

```
Input: num = "10", k = 2
Output: "0"
Explanation: Remove all the digits from the number and it is left with nothing which is 0.
```

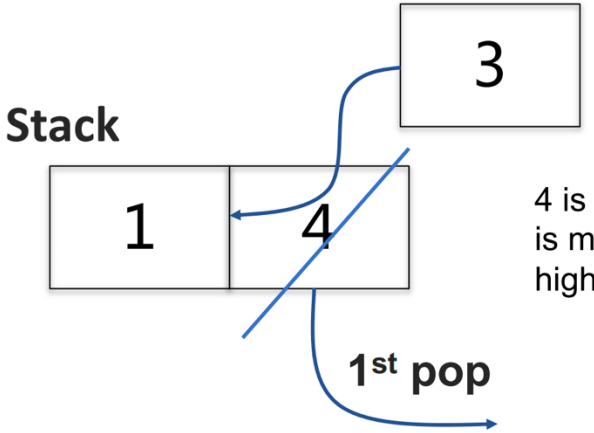
https://leetcode.com/problems/remove-k-digits/

# Solution 1: Remove K Digits

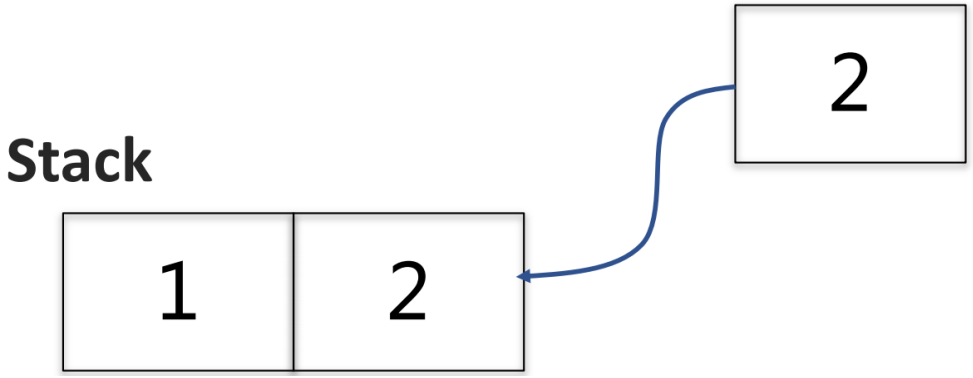
Example 1: Input: num = "1432219", k = 4 Output: "119"

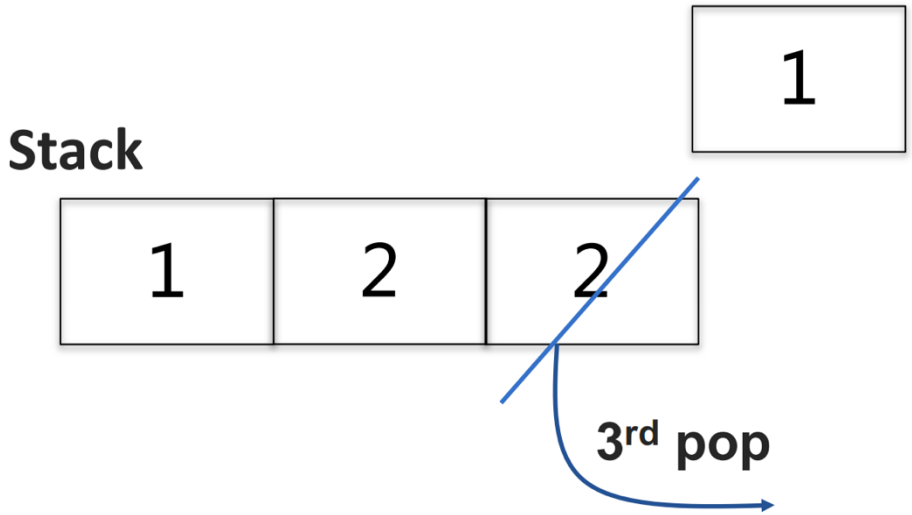



4 is larger than 1, change the high position will not produce any benefit



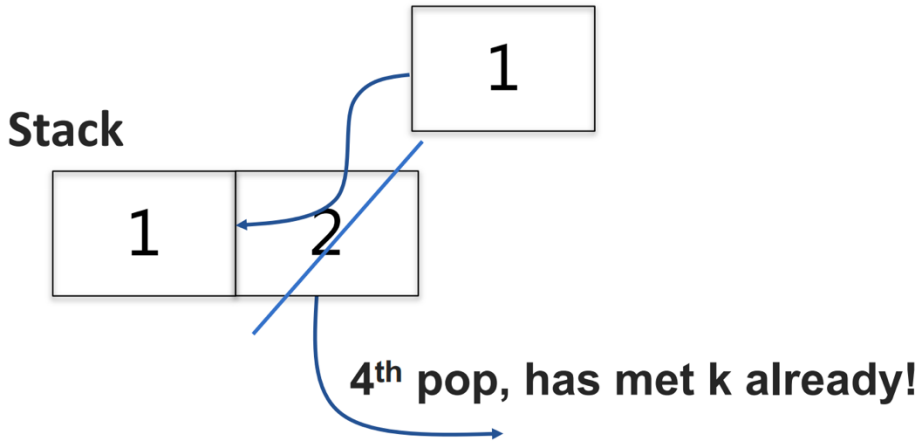

4 is larger than 3. For current step, 3 is more proper to take the second highest position.

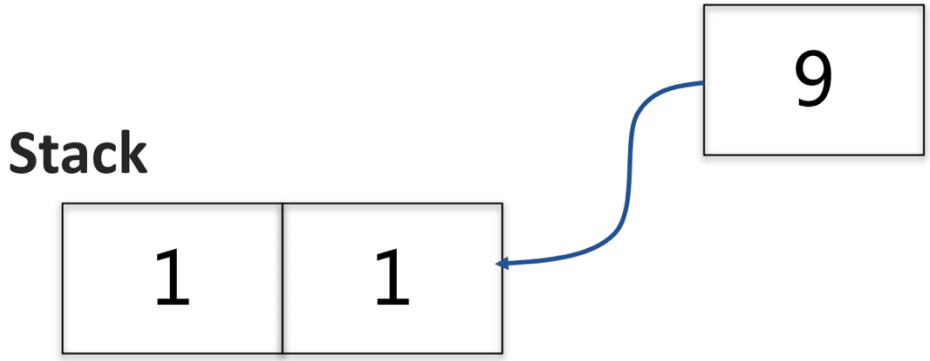


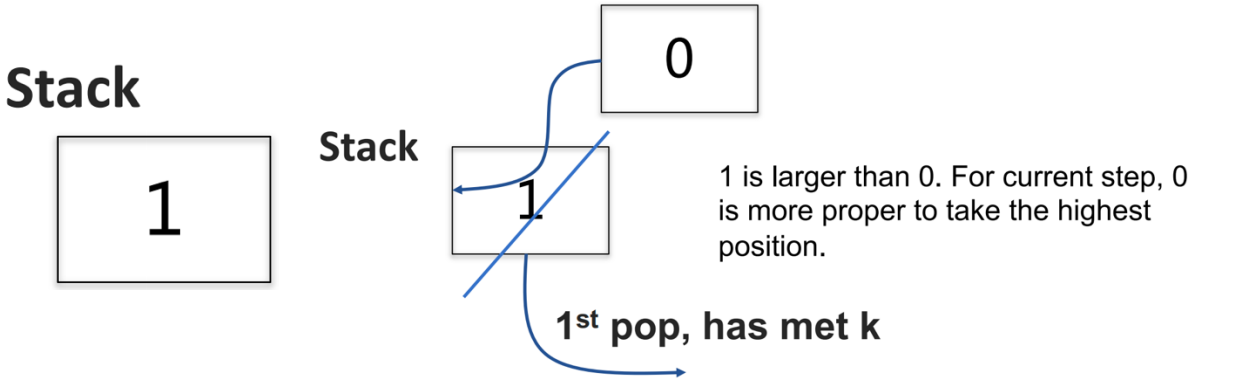
# Solution 1: Remove K Digits

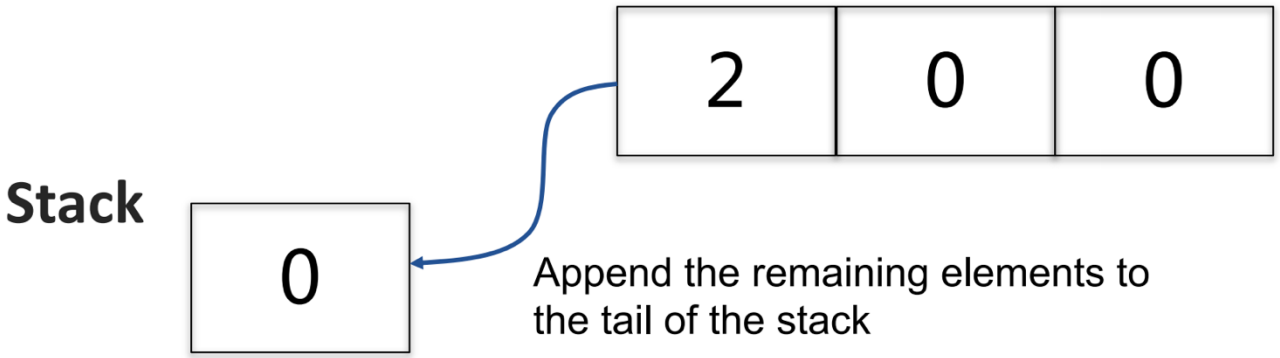
Example 1: Input: num = "1432219", k = 4 Output: "119"




Example 2: Input: num = "10200", k = 1 Output: "200"

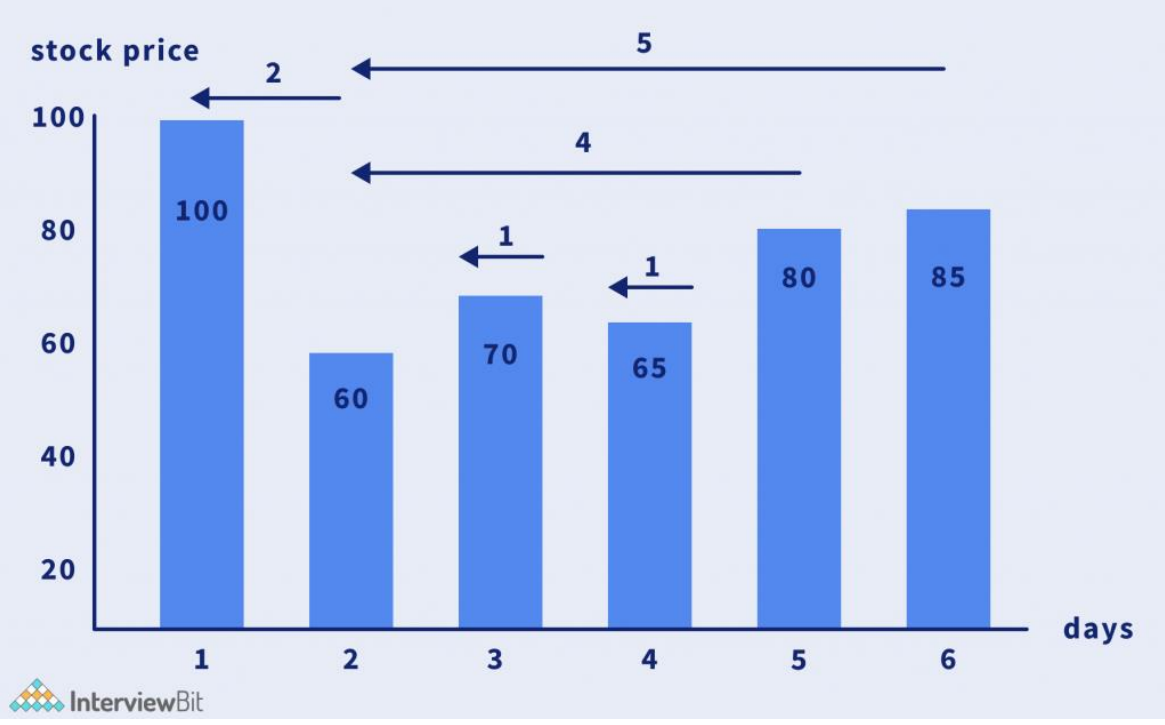


# Exercise 2: Stock Span Problem (LeetCode P901)

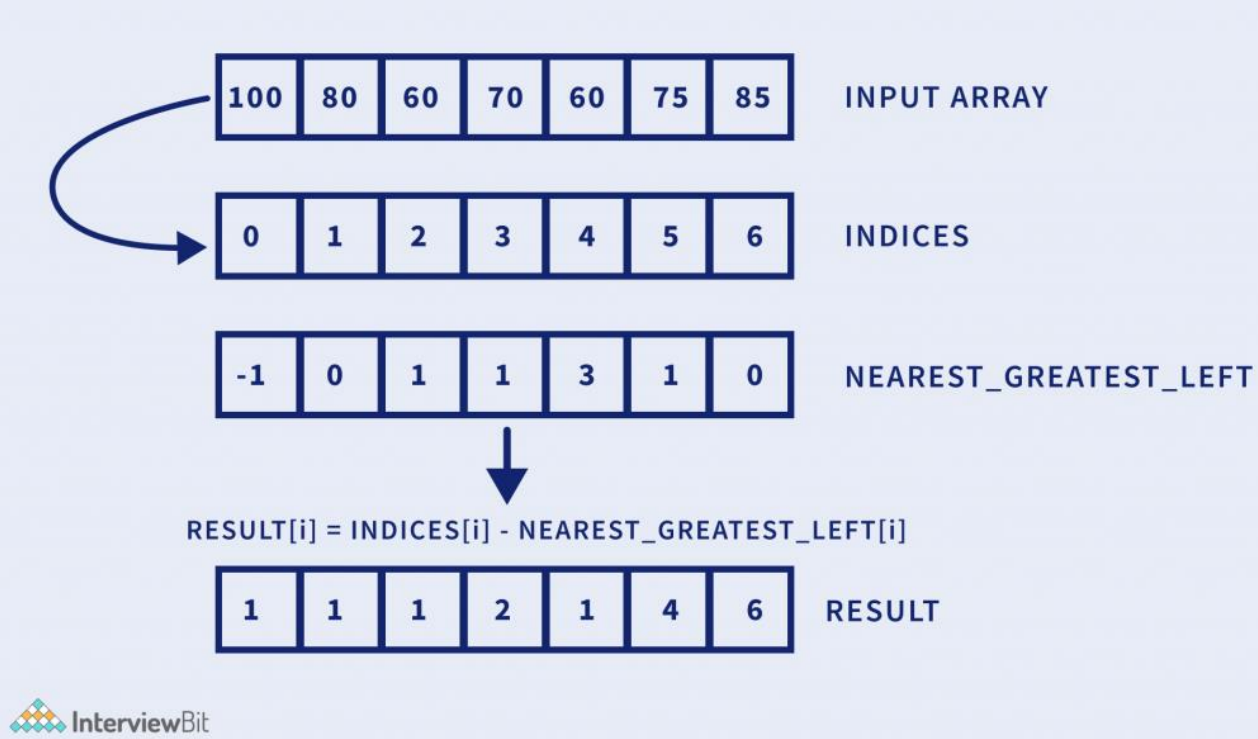
Given a list of prices of a stock for **N** days. The task is to find the stock span for each day.

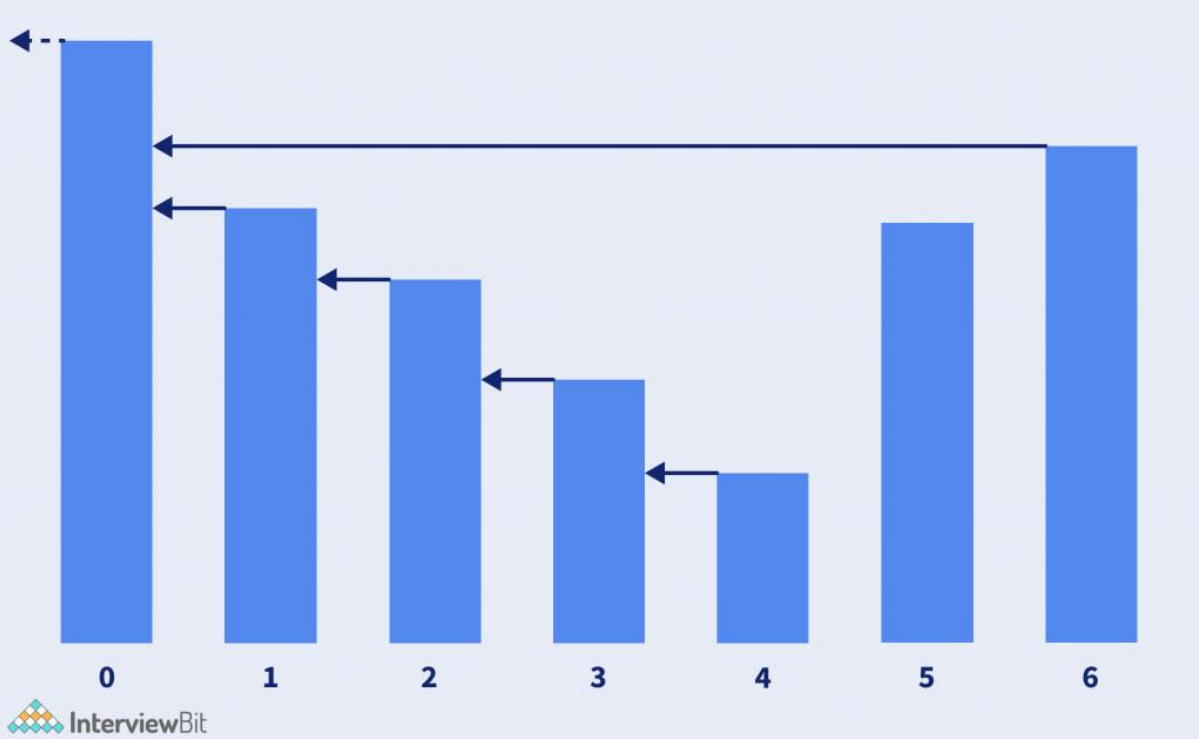
**Stock span** can be defined as the number of consecutive days before the current day where the price of the stock was equal to or less than the current price.




**Examples:**
**Input: A[]** = [100,60,70,65,80,85]
**Output:** [1,1,2,1,4,5]

# Solution 2: Stock Span Problem



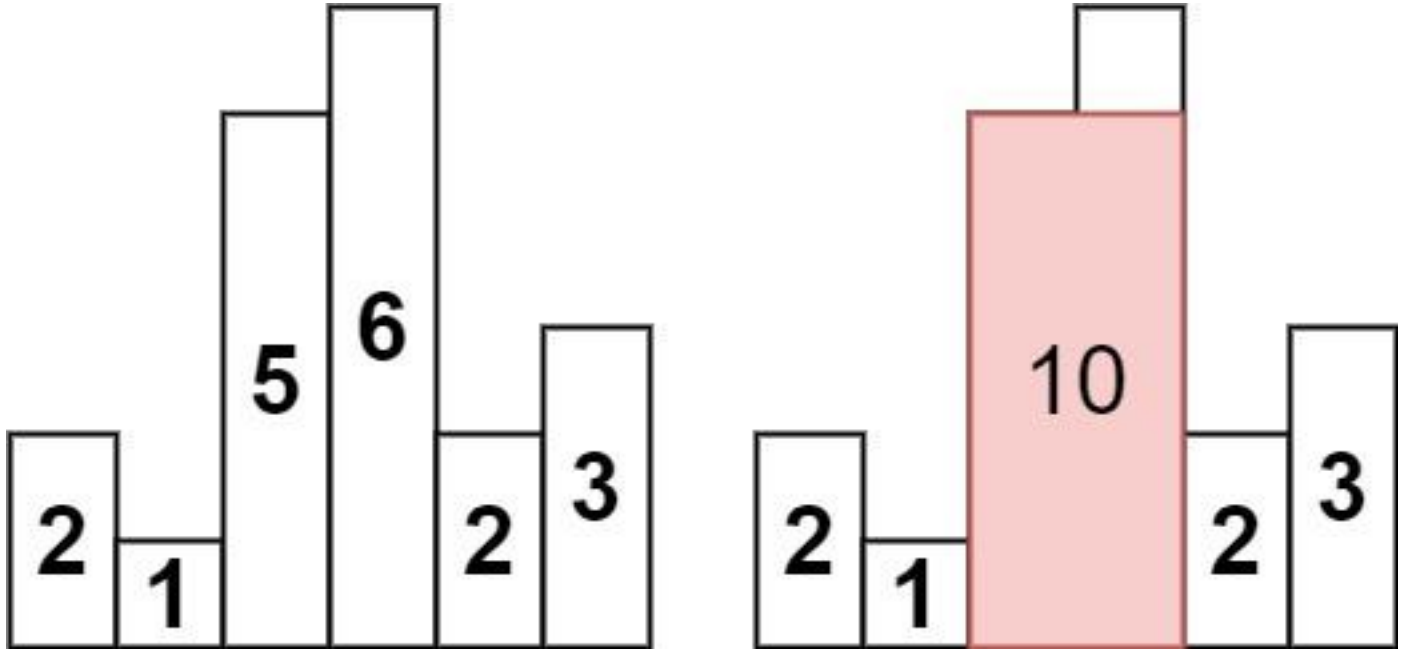
# Solution 2: Stock Span Problem

**Algorithm**

- Initialize an array **S[0] = 1** for day 0.
- Initialize a stack and push the **index** of the first day into the stack.
- Traverse a loop from **1** to **N** and for each iteration, do the following:
  - If the stack is not **empty** and **price** of current element is greater than the top of stack, pop the element.
  - Else if, stack is not empty then, subtract index from **S[i]**, i.e. **S[i] = i – S.top()**.
  - Else, **S[i] = i + 1**
- Push the current index **i** into the stack

# Exercise 3: Largest Rectangular Area in a Histogram using Stack (LeetCode P84)

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.

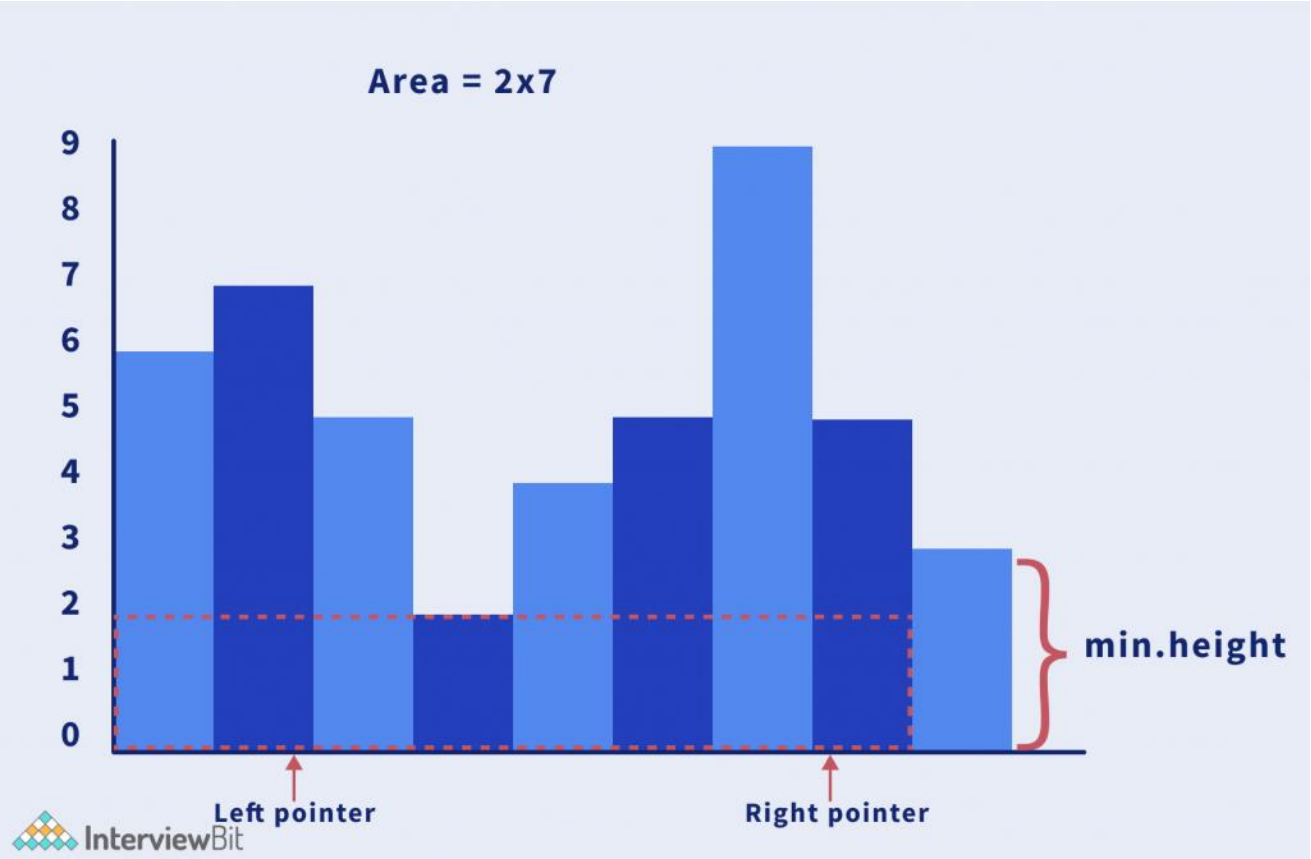



**Input:** heights = [2,1,5,6,2,3]
**Output:** 10

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

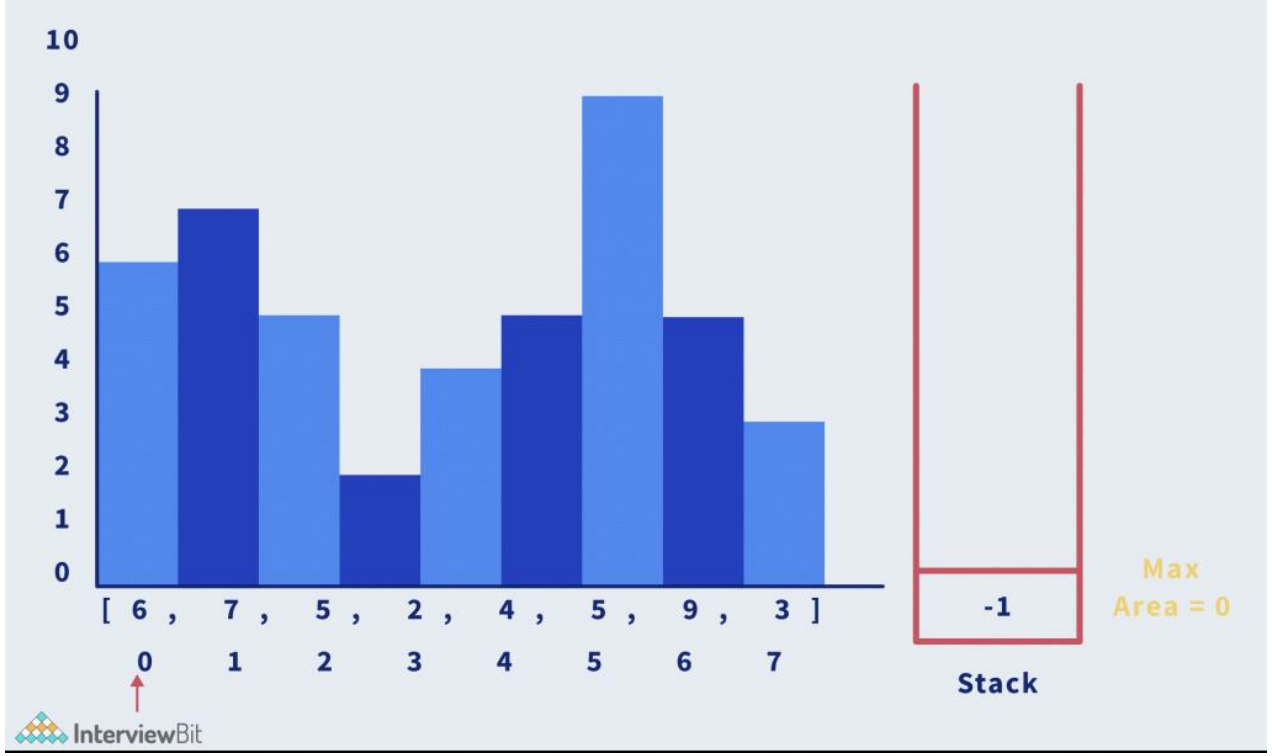
# Solution 3: Largest Rectangular Area in a Histogram using Stack

- **Approach 1: Brute Force**



# Solution 3: Largest Rectangular Area in a Histogram using Stack

- **Approach 2: Stack**




**Algorithm:**

•Initialise a stack **S**.

•Push the first index of **A[]** into the stack.

•Traverse through the array **A[]** and compare the height of **A[i]** with the height at the top of the stack.

•If the height is:

- Greater than **A[S.top()]**, push it into the stack.
- Less than **A[S.top()]**, keep popping the elements until **A[i] >= A[S.top()].**

•Keep maximizing the area while popping the elements from the stack.

•Push the index **i** for each element.

•Return the maximum element.

# Thanks for your Attention!

# Q&A