

[Courses](#)[Practice](#)[Roadmap](#)[Pro](#)

Algorithms and Data Structures for Beginners

5 / 35

About

0 Introduction FREE

Arrays

1 RAM FREE

2 Static Arrays

3 Dynamic Arrays

4 **Stacks**

Linked Lists

4 - Stacks

04:26






Mark Lesson Complete

[View Code](#)[Prev](#)[Next](#)

5 Singly Linked
Lists

FREE

Suggested Problems

Status	Star	Problem ↕	Difficulty ↕	Video Solution	Code
<input type="checkbox"/>	☆	Baseball Game	Easy		C++
<input type="checkbox"/>	☆	Valid Parentheses	Easy		C++
<input type="checkbox"/>	☆	Min Stack	Medium		C++

Stacks

A stack is a data structure that contains a collection of elements where you can add and delete elements from just one end (called the top of the stack). In the physical world, a stack can be conceptualized by thinking of plates at a dinner party buffet. When you go to take a plate, you can only remove from the top and similarly when you finish your meal, the stack of plates can only be built by adding them on top of each other — this is exactly what a stack in the software world does.

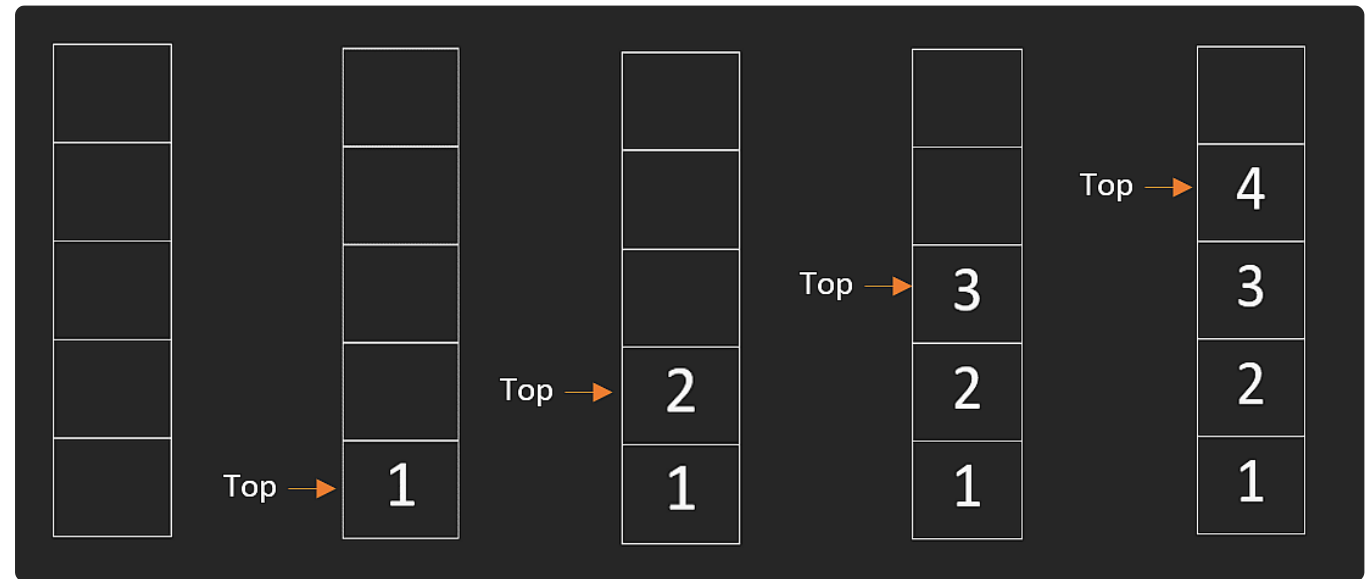
Stacks are a dynamic data structure that operate on a LIFO (Last In First Out) manner. The last element placed inside is the first element that comes out. The

stack supports three operations - **push** , **pop** , **peek** .

Push

Push operation adds an element to the top of the stack, which in dynamic array terms would be appending an element to the end. This is an efficient $O(1)$ operation as discussed in the previous chapters. It helps to visualize a stack as an array that is vertical. The pseudocode demonstrates the concept, along with the visual where we add numbers from 1 to 4 to the top. The top pointer updates to point at the last item added. The following pseudocode and visual demonstrates this.

```
fn push(n):  
    // using the pushback function from dynamic arrays to add to the st  
    stack.pushback(n)
```



Stack, as a data structure, is just an abstract interface and it does not really matter how you implement it - the characteristics are just that you should be able to add and remove elements from the same end.

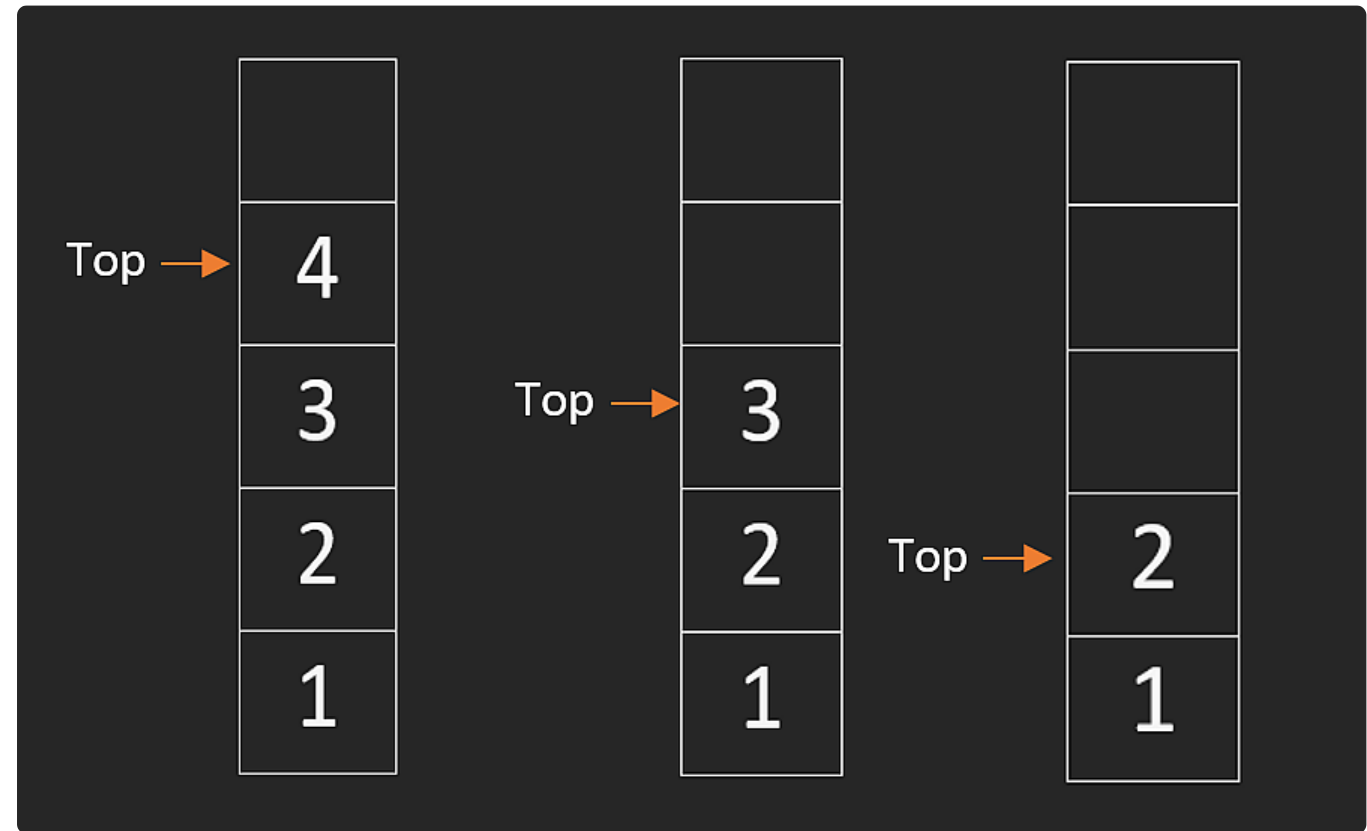
Since a stack will remove elements in the reverse order that it inserted them in, it can be used to reverse sequences - such as a string, which is just a sequence of characters.

Pop

Pop operation removes the last element from top of the stack, which in dynamic array terms would be retrieving the last element. This is also an efficient $O(1)$

operation as discussed in the previous chapters. Taking the previous example, let's say we wish to pop 4 and 5. The pseudocode below demonstrates the concept, along with the visual where we remove 4 and 5 from the top. Again, the top pointer updates to point at the last item.

```
fn pop():  
    // retrieve the last element before removing  
    top = stack.get(lastIndex, lengthOfArray)  
  
    // remove the last element  
    stack.popback()  
  
    // return the last element  
    return top
```



In most languages, before popping, it is a good measure to check if the stack is empty to avoid errors.

Peek

Peek is the simplest of three. It just returns, without removing, the top most element.

```
fn peek():  
    // retrieve the last element
```

```
top = stack.get(lastIndex, lengthOfArray)
return top
```

Closing Notes

Operation	Big-O Time Complexity	Notes
Push	$O(1)$	
Pop	$O(1)^*$	Check if the stack is empty first
Peek/Top	$O(1)^*$	Retrieves without removing

Copyright © 2023 NeetCode.io All rights reserved.

Contact: neetcodebusiness@gmail.com

[Github](#) [Privacy](#) [Terms](#)