# Implement two stacks in an array

Difficulty Level : Easy • Last Updated : 04 Jul, 2022

Create a data structure *twoStacks* that represents two stacks. Implementation of *twoStacks* should use only one array, i.e., both stacks should use the same array for storing elements.

*Following functions must be supported by twoStacks.*

- *push1(int x) -> pushes x to first stack*
- *push2(int x) -> pushes x to second stack*
- *pop1() -> pops an element from first stack and return the popped element*
- *pop2() -> pops an element from second stack and return the popped element*

*Implementation of twoStack should be space efficient.*



**Method 1 (Divide the space in two halves):**

A simple way to implement two stacks is to divide the array in two halves and assign the half space to two stacks, i.e., use arr[0] to arr[n/2] for stack1, and arr[(n/2) + 1] to arr[n-1] for stack2 where arr[] is the array to be used to implement two stacks and size of array be n. The problem with this method is inefficient use of array space. A stack push operation may result in stack overflow even if there is space available in arr[]. For example, say the array size is 6 and we push 3 elements to stack1 and do not push anything to second stack2. When we push 4th element to stack1, there will be overflow even if we have space for 3 more elements in array.

C++ Java Python3 C# Javascript

```python
# Python Script to Implement two stacks in a list
import math

class twoStacks:

    def __init__(self, n):      # constructor
        self.size = n
        self.arr = [None] * n
        self.top1 = math.floor(n/2) + 1
        self.top2 = math.floor(n/2)


    # Method to push an element x to stack1
    def push1(self, x):

        # There is at least one empty space for new element
        if self.top1 > 0:
            self.top1 = self.top1 - 1
            self.arr[self.top1] = x
        else:
            print("Stack Overflow by element : ", x)


    # Method to push an element x to stack2
    def push2(self, x):

        # There is at least one empty space for new element
```

```
        if self.top2 < self.size - 1:
            self.top2 = self.top2 + 1
            self.arr[self.top2] = x
        else :
            print("Stack Overflow by element : ", x)


    # Method to pop an element from first stack
    def pop1(self):
        if self.top1 <= self.size/2:
            x = self.arr[self.top1]
            self.top1 = self.top1 +1
            return x
        else:
            print("Stack Underflow ")
            exit(1)


    # Method to pop an element from second stack
    def pop2(self):
        if self.top2 >= math.floor(self.size/2) + 1:
            x = self.arr[self.top2]
            self.top2 = self.top2 - 1
            return x
        else:
            print("Stack Underflow ")
            exit(1)


# Driver program to test twoStacks class
ts = twoStacks(5)
ts.push1(5)
ts.push2(10)
ts.push2(15)
ts.push1(11)
ts.push2(7)

print("Popped element from stack1 is : " + str(ts.pop1()))
ts.push2(40)
print("Popped element from stack2 is : " + str(ts.pop2()))

# This code is contributed by Gautam goel
```



**Output**

```
Stack Overflow By element :7
Popped element from stack1 is  : 11
Stack Overflow By element :40

Popped element from stack2 is : 15
```

**Complexity Analysis:**

- **Time Complexity:**
  - **Push operation :** O(1)
  - **Pop operation :** O(1)
- **Auxiliary Space:** O(N).
  Use of array to implement stack so. It is not the space-optimized method as explained above.

**Method 2 (A space efficient implementation)** :

This method efficiently utilizes the available space. It doesn't cause an overflow if there is space available in arr[]. The idea is to start two stacks from two extreme corners of arr[]. stack1 starts from the leftmost element, the first element in stack1 is pushed at index 0. The stack2 starts from the rightmost corner, the first element in stack2 is pushed at index (n-1). Both stacks grow (or shrink) in opposite direction. To check for overflow, all we need to check is for space between top elements of both stacks. This check is highlighted in the below code.

C++ Java **Python** C# PHP Javascript

```
# Python Script to Implement two stacks in a list
class twoStacks:

    def __init__(self, n):     # constructor
        self.size = n
        self.arr = [None] * n
        self.top1 = -1
        self.top2 = self.size

    # Method to push an element x to stack1
    def push1(self, x):

        # There is at least one empty space for new element
```

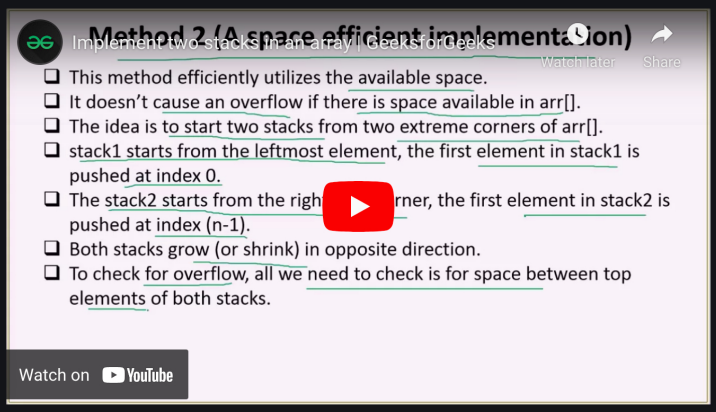
```
            if self.top1 < self.top2 - 1 :
                self.top1 = self.top1 + 1
                self.arr[self.top1] = x

            else:
                print("Stack Overflow ")
                exit(1)

        # Method to push an element x to stack2
        def push2(self, x):

            # There is at least one empty space for new element
            if self.top1 < self.top2 - 1:
                self.top2 = self.top2 - 1
                self.arr[self.top2] = x

            else:
                print("Stack Overflow ")
                exit(1)

        # Method to pop an element from first stack
        def pop1(self):
            if self.top1 >= 0:
                x = self.arr[self.top1]
                self.top1 = self.top1 -1
                return x
            else:
                print("Stack Underflow ")
                exit(1)

        # Method to pop an element from second stack
        def pop2(self):
            if self.top2 < self.size:
                x = self.arr[self.top2]
                self.top2 = self.top2 + 1
                return x
            else:
                print("Stack Underflow ")
                exit()

    # Driver program to test twoStacks class
    ts = twoStacks(5)
    ts.push1(5)
    ts.push2(10)
    ts.push2(15)
    ts.push1(11)
    ts.push2(7)

    print("Popped element from stack1 is " + str(ts.pop1()))
    ts.push2(40)
    print("Popped element from stack2 is " + str(ts.pop2()))

    # This code is contributed by Sunny Karira
```

**Output:**

```
Popped element from stack1 is 11
Popped element from stack2 is 40
```

**Complexity Analysis:**

- **Time Complexity:**
  - **Push operation :** O(1)
  - **Pop operation :** O(1)
- **Auxiliary Space :** O(N).
  Use of array to implement stack so it is a space-optimized method.



Like 186

Previous

Find whether an array is subset of another array

Next

Find relative complement of two sorted arrays



## RECOMMENDED ARTICLES

Page : 1 2 3

01 How to efficiently implement k stacks in a single array?
22, Aug 14

02 Implement Dynamic Multi Stack (K stacks) using only one Data Structure
09, Dec 21

03 Infix to Prefix conversion using two stacks
17, Jun 18

04 Add two numbers represented by Stacks
20, Aug 20

05 Reversing a Stack using two empty Stacks
09, Jun 21

06 Maximum count of integers to be chosen from given two stacks having sum at most K
24, Jan 22

07 Merging and Sorting Two Unsorted Stacks
20, May 18

08 Iterative Postorder Traversal | Set 1 (Using Two Stacks)
23, Feb 13

**Article Contributed By :**

GeeksforGeeks

**Vote for difficulty**

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : sidsm009, shrikanth13, rathbhupendra, bidibaaz123, aashish1995

avanitrachhadiya2155, krishna_97, umadevi9616, rkbhola5, Rajput-Ji, simmytarika5, gautamgoel962, hardikkoriintern

**Article Tags :** 24*7 Innovation Labs, Accolite, Microsoft, Samsung, Snapdeal, Arrays, Stack

**Practice Tags :** 24*7 Innovation Labs, Accolite, Microsoft, Samsung, Snapdeal, Arrays, Stack

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments