Implement Stack using Queues - GeeksforGeeks

Source: https://www.geeksforgeeks.org/implement-stack-using-queue/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Implement Stack using Queues Last Updated : 18 Sep, 2025 Implement a stack using queues. We are allowed to use only queue data structure. The stack should support the following operations: push(x): Insert element x onto the top of the stack. pop(): Remove the element from the top of the stack. If the stack is empty, do nothing. top(): Return the element at the top of the stack. If the stack is empty, return -1. size(): Return the number of elements in the stack. Try it on GfG Practice Table of Content [Approach 1] Using Two Queue - Push in O(n) and Pop() in O(1) [Approach 2] Using Two Queue - Push in O(1) and Pop() in O(n) [Approach 3] Using Single Queue - Push in O(n) and Pop() in O(1) [Approach 1] Using Two Queue - Push in O(n) and Pop() in O(1) We will be using two queues (q1 and q2) to implement the stack operations. The main idea is to always keep the newly inserted element at the front of q1, so that both pop() and top() can directly access it. Queue q2 acts as a helper to rearrange elements during push(). push(x): Enqueue the new element x into q2. Move all elements from q1 into q2, one by one. Swap the names of q1 and q2. After the swap, q1 contains the updated stack order with the newest element at the front. pop(): If q1 is empty, the stack is empty (underflow condition). Otherwise, dequeue from the front of q1, which represents the top of the stack. top(): If q1 is empty, return -1. Otherwise, return the front element of q1, since it represents the current top of the stack. size(): Simply return the number of elements in q1, which tracks the current stack size. C++ #include <iostream> #include <queue> using namespace std ; class myStack { queue < int > q1 , q2 ; public : void push (int x) { // Push x first in empty q2 q2 . push (x); // Push all the remaining // elements in q1 to q2. while (! q1 . empty ()) { q2 . push (q1 . front ()); q1 . pop (); } // swap the names of two queues swap (q1 , q2); } void pop () { // if no elements are there in q1 if (q1 . empty ()) return ; q1 . pop (); } int top () { if (q1 . empty ()) return -1 ; return q1 . front (); } int size () { return q1 . size (); } }; // Driver code int main () { myStack st ; st . push (1); st . push (2); st . push (3); cout << st . top () << endl ; st . pop (); cout << st . top () << endl ; st . pop (); cout << st . top () << endl ; cout << st . size () << endl ; return 0 ; } Java import java.util.LinkedList ; import java.util.Queue ; class MyStack { Queue < Integer > q1 = new LinkedList <> (); Queue < Integer > q2 = new LinkedList <> (); public void push (int x) { // Push x first in empty q2 q2 . add (x); // Push all the remaining // elements in q1 to q2. while (! q1 . isEmpty ()) { q2 . add (q1 . peek ()); q1 . remove (); } // swap the names of two queues Queue < Integer > t = q1 ; q1 = q2 ; q2 = t ; } public void pop () { // if no elements are there in q1 if (q1 . isEmpty ()) return ; q1 . remove (); } public int top () { if (q1 . isEmpty ()) return - 1 ; return q1 . peek (); } public int size () { return q1 . size (); } } public class GfG { public static void main (String [] args) { MyStack st = new MyStack (); st . push (1); st . push (2); st . push (3); System . out . println (st . top ()); st . pop (); System . out . println (st . top ()); st . pop (); System . out . println (st . top ()); System . out . println (st . size ()); } } Python from collections import deque class myStack : def __init__ (self): self . q1 = deque () self . q2 = deque () def push (self , x): # Push x first in empty q2 self . q2 . append (x) # Push all the remaining # elements in q1 to q2. while len (self . q1) != 0 : self . q2 . append (self . q1 [0]) self . q1 . popleft () # swap the names of two queues self . q1 , self . q2 = self . q2 , self . q1 def pop (self): # if no elements are there in q1 if len (self . q1) == 0 : return self . q1 . popleft () def top (self): if len (self . q1) == 0 : return - 1 return self . q1 [0] def size (self): return len (self . q1) if __name__ == '__main__' : st = myStack () st . push (1) st . push (2) st . push (3) print (st . top ()) st . pop () print (st . top ()) st . pop () print (st . top ()) print (st . size ()) C# using System ; using System.Collections.Generic ; public class myStack { Queue < int > q1 = new Queue < int > (); Queue < int > q2 = new Queue < int > (); public void

push (int x) { // Push x first in empty q2 q2 . Enqueue (x); // Push all the remaining // elements in q1 to q2. while (q1 . Count > 0) { q2 . Enqueue (q1 . Peek ()); q1 . Dequeue (); } // swap the names of two queues Queue < int > t = q1 ; q1 = q2 ; q2 = t ; } public void pop () { // if no elements are there in q1 if (q1 . Count == 0) return ; q1 . Dequeue (); } public int top () { if (q1 . Count == 0) return - 1 ; return q1 . Peek (); } public int size () { return q1 . Count ; } } class GfG { public static void Main () { myStack st = new myStack (); st . push (1); st . push (2); st . push (3); Console . WriteLine (st . top ()); st . pop (); Console . WriteLine (st . top ()); st . pop (); Console . WriteLine (st . top ()); Console . WriteLine (st . size ()); } } JavaScript class myStack { constructor () { this . q1 = []; this . q2 = []; } push (x) { // Push x first in empty q2 this . q2 . push (x); // Push all the remaining // elements in q1 to q2. while (this . q1 . length > 0) { this . q2 . push (this . q1 [0]); this . q1 . shift (); } // swap the names of two queues let t = this . q1 ; this . q1 = this . q2 ; this . q2 = t ; } pop () { // if no elements are there in q1 if (this . q1 . length === 0) return ; this . q1 . shift (); } top () { if (this . q1 . length === 0) return - 1 ; return this . q1 [0]; } size () { return this . q1 . length ; } } // Driver code let st = new myStack (); st . push (1); st . push (2); st . push (3); console . log (st . top ()); st . pop (); console . log (st . top ()); st . pop (); console . log (st . top ()); console . log (st . size ()); Output 3 2 1 1 Time Complexity: push operation : O(n), as every new element must be inserted first into q2, then all elements from q1 are moved to q2. pop operation : O(1), since the top element is always at the front of q1. top operation : O(1), because the top can be accessed directly from the front of q1. size operation : O(1), as size is tracked by the queue. Auxiliary Space : O(n) [Approach 2] Using Two Queue - Push in O(1) and Pop() in O(n) We are implementing a stack using two queues (q1 and q2). The idea is to make the push(x) operation simple, and adjust the order during pop() and top() so that the stack behavior (Last-In-First-Out) is preserved. push(x): The element x is simply enqueued into q1. This keeps push() efficient. pop(): If q1 is empty, the stack is empty (underflow condition). Otherwise, move elements from q1 to q2 until only one element is left in q1. Remove this last element from q1 (which represents the top of the stack). Swap the names of q1 and q2 so that q1 again holds the current stack elements. top(): If q1 is empty, return -1. Otherwise, move elements from q1 to q2 until only one element is left. Store this last element (the top of the stack), then move it to q2 instead of discarding it. Swap q1 and q2 to restore order. Return the stored element. size(): Return the number of elements currently in q1, which represents the size of the stack. C++ #include <iostream> #include <queue> using namespace std ; class myStack { queue < int > q1 , q2 ; public : // insert element void push (int x) { q1 . push (x); } // remove top element void pop () { if (q1 . empty ()) return ; while (q1 . size () != 1) { q2 . push (q1 . front ()); q1 . pop (); } q1 . pop (); swap (q1 , q2); } // return top element int top () { if (q1 . empty ()) return -1 ; while (q1 . size () != 1) { q2 . push (q1 . front ()); q1 . pop (); } int temp = q1 . front (); q1 . pop (); q2 . push (temp); swap (q1 , q2); return temp ; } // return current size int size () { return q1 . size (); } }; int main () { myStack st ; st . push (1); st . push (2); st . push (3); cout << st . top () << endl ; st . pop (); cout << st . top () << endl ; st . pop (); cout << st . top () << endl ; cout << st . size () << endl ; return 0 ; } Java import java.util.LinkedList ; import java.util.Queue ; class myStack { Queue < Integer > q1 = new LinkedList <> (); Queue < Integer > q2 = new LinkedList <> (); // insert element void push (int x) { q1 . add (x); } // remove top element void pop () { if (q1 . isEmpty ()) return ; while (q1 . size () != 1) { q2 . add (q1 . peek ()); q1 . remove (); } q1 . remove (); Queue < Integer > temp = q1 ; q1 = q2 ; q2 = temp ; } // return top element int top () { if (q1 . isEmpty ()) return - 1 ; while (q1 . size () != 1) { q2 . add (q1 . peek ()); q1 . remove (); } int temp = q1 . peek (); q1 . remove (); q2 . add (temp); Queue < Integer > t = q1 ; q1 = q2 ; q2 = t ; return temp ; } // return current size int size () { return q1 . size (); } } public class GfG { public static void main (String [] args) { myStack st = new myStack (); st . push (1); st . push (2); st . push (3); System . out . println (st . top ()); st . pop (); System . out . println (st . top ()); st . pop (); System . out . println (st . top ()); System . out . println (st . size ()); } } Python from collections import deque class myStack : def __init__ (self): self . q1 = deque () self . q2 = deque () # insert element def push (self , x): self . q1 . append (x) # remove top element def pop (self): if not self . q1 : return while len (self . q1) != 1 : self . q2 . append (self . q1 . popleft ()) self . q1 . popleft () self . q1 , self . q2 = self . q2 , self . q1 # return top element def top (self): if not self . q1 : return - 1 while len (self . q1) != 1 : self . q2 . append (self . q1 . popleft ()) temp = self . q1 . popleft () self . q2 . append (temp) self . q1 , self . q2 = self . q2 , self . q1 return temp # return current size def size (self): return len (self . q1) if __name__ == '__main__' : st = myStack () st . push (1) st . push (2) st . push (3) print (st . top ()) st . pop () print (st . top ()) st . pop () print (st . top ()) print (st . size ()) C# using System ; using System.Collections.Generic ; class myStack { Queue < int > q1 = new Queue < int > (); Queue < int > q2 = new Queue < int > (); // insert element public void push (int x) { q1 . Enqueue (x); } // remove top element public void pop () { if (q1 . Count == 0) return ; while (q1 . Count != 1) { q2 . Enqueue (q1 . Dequeue ()); } q1 . Dequeue (); var

temp = q1 ; q1 = q2 ; q2 = temp ; } // return top element public int top () { if (q1 . Count == 0) return - 1 ; while (q1 . Count != 1) { q2 . Enqueue (q1 . Dequeue ()); } int temp = q1 . Dequeue (); q2 . Enqueue (temp); var t = q1 ; q1 = q2 ; q2 = t ; return temp ; } // return current size public int size () { return q1 . Count ; } } class GfG { static void Main () { myStack st = new myStack (); st . push (1); st . push (2); st . push (3); Console . WriteLine (st . top ()); st . pop (); Console . WriteLine (st . top ()); st . pop (); Console . WriteLine (st . top ()); Console . WriteLine (st . size ()); } } JavaScript class myStack { constructor () { this . q1 = []; this . q2 = []; } // insert element push (x) { this . q1 . push (x); } // remove top element pop () { if (this . q1 . length === 0) return ; while (this . q1 . length !== 1) { this . q2 . push (this . q1 . shift ()); } this . q1 . shift (); [this . q1 , this . q2] = [this . q2 , this . q1]; } // return top element top () { if (this . q1 . length === 0) return - 1 ; while (this . q1 . length !== 1) { this . q2 . push (this . q1 . shift ()); } let temp = this . q1 . shift (); this . q2 . push (temp); [this . q1 , this . q2] = [this . q2 , this . q1]; return temp ; } // return current size size () { return this . q1 . length ; } } // Driver Code const st = new myStack (); st . push (1); st . push (2); st . push (3); console . log (st . top ()); st . pop (); console . log (st . top ()); st . pop (); console . log (st . top ()); console . log (st . size ()); Output 3 2 1 1 Time Complexity: push operation : O(1), as the element is directly added to q1. pop operation : O(n), since all elements except the last need to be moved to q2 before removal. top operation : O(n), as we need to move elements to access the last inserted one. size operation : O(1), because queue maintains its size. Auxiliary Space: O(n) [Approach 3] Using Single Queue - Push in O(n) and Pop() in O(1) In this approach, we implement a stack using only one queue. The main idea is to always keep the most recently pushed element at the front of the queue, so that top() and pop() work in O(1) time. push(x): Insert the new element at the back of the queue. Rotate the queue by repeatedly removing the front element and pushing it back until the new element comes to the front. After rotation, the newest element will be at the front, acting as the stack top. size(): Return the size of the queue. top(): Return the element at the front of the queue, since it represents the top of the stack. pop(): Simply dequeue the front element of the queue. C++ #include <iostream> #include <queue> using namespace std ; class myStack { queue < int > q ; public : // push element on top void push (int x) { q . push (x); int sz = q . size (); for (int i = 0 ; i < sz - 1 ; i ++) { q . push (q . front ()); q . pop (); } } // remove top element void pop () { if (! q . empty ()) q . pop (); } // return top element int top () { if (q . empty ()) return -1 ; return q . front (); } // return current size int size () { return q . size (); } }; int main () { myStack st ; st . push (1); st . push (2); st . push (3); cout << st . top () << endl ; st . pop (); cout << st . top () << endl ; st . pop (); cout << st . top () << endl ; cout << st . size () << endl ; return 0 ; } Java import java.util.LinkedList ; import java.util.Queue ; class myStack { private Queue < Integer > q = new LinkedList <> (); // push element on top public void push (int x) { q . add (x); int sz = q . size (); for (int i = 0 ; i < sz - 1 ; i ++) { q . add (q . peek ()); q . poll (); } } // remove top element public void pop () { if (! q . isEmpty ()) q . poll (); } // return top element public int top () { if (q . isEmpty ()) return - 1 ; return q . peek (); } // return current size public int size () { return q . size (); } } public class GfG { public static void main (String [] args) { myStack st = new myStack (); st . push (1); st . push (2); st . push (3); System . out . println (st . top ()); st . pop (); System . out . println (st . top ()); st . pop (); System . out . println (st . top ()); System . out . println (st . size ()); } } Python from collections import deque class myStack : def __init__ (self): self . q = deque () # push element on top def push (self , x): self . q . append (x) sz = len (self . q) for i in range (sz - 1): self . q . append (self . q [0]) self . q . popleft () # remove top element def pop (self): if self . q : self . q . popleft () # return top element def top (self): if not self . q : return - 1 return self . q [0] # return current size def size (self): return len (self . q) if __name__ == '__main__' : st = myStack () st . push (1) st . push (2) st . push (3) print (st . top ()) st . pop () print (st . top ()) st . pop () print (st . top ()) print (st . size ()) C# using System ; using System.Collections.Generic ; class myStack { private Queue < int > q = new Queue < int > (); // push element on top public void push (int x) { q . Enqueue (x); int sz = q . Count ; for (int i = 0 ; i < sz - 1 ; i ++) { q . Enqueue (q . Peek ()); q . Dequeue (); } } // remove top element public void pop () { if (q . Count > 0) q . Dequeue (); } // return top element public int top () { if (q . Count == 0) return - 1 ; return q . Peek (); } // return current size public int size () { return q . Count ; } } class GfG { static void Main () { myStack st = new myStack (); st . push (1); st . push (2); st . push (3); Console . WriteLine (st . top ()); st . pop (); Console . WriteLine (st . top ()); st . pop (); Console . WriteLine (st . top ()); Console . WriteLine (st . size ()); } } JavaScript class myStack { constructor () { this . q = []; } // push element on top push (x) { this . q . push (x); let sz = this . q . length ; for (let i = 0 ; i < sz - 1 ; i ++) { this . q . push (this . q [0]); this . q . shift (); } } // remove top element pop () { if (this . q . length > 0) this . q . shift (); } // return top element top () { if (this . q . length === 0) return - 1 ; return this . q [0]; } // return current size size () { return this . q . length ; } } // Driver Code const st = new myStack (); st . push (1); st . push (2); st . push (3); console

. log (st . top ()); st . pop (); console . log (st . top ()); st . pop (); console . log (st . top ()); console . log (st . size ()); Output 3 2 1 1 Time Complexity : push operation : O(n), since after inserting the element, all previous elements must be rotated behind it. pop operation : O(1), as the top element is always at the front of the queue. top operation : O(1), because the most recent element is always at the front. size operation : O(1), queue keeps track of size. Auxiliary Space: O(n) Comment Article Tags: Article Tags: Stack Queue DSA Amazon Adobe Oracle D-E-Shaw Snapdeal Accolite CouponDunia Kritikal Solutions Grofers + 8 More