Stack with Minimum - GeeksforGeeks

Source:
https://www.geeksforgeeks.org/design-a-stack-that-supports-getmin-in-o1-time-and-o1-extra-space/

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 Stack with Minimum Last Updated : 16 Sep, 2025 Design a SpecialStack that supports push(x), pop(), peek(), and getMin() in O(1) time. push(x) → add element x pop() → remove top element peek() → return top element without removing; -1 if empty getMin() → return minimum element; -1 if empty All operations run in O(1). Example: Input: operations[] = [push(2), push(3), peek(), pop(), getMin(), push(1), getMin()] Output: [3, 2, 1] Explanation: push(2): Stack is [2] push(3): Stack is [2, 3] peek(): Top element is 3 pop(): Removes 3, stack is [2] getMin(): Minimum element is 2 push(1): Stack is [2, 1] getMin(): Minimum element is 1 Input: operations[] = [push(10), getMin(), push(5), getMin(), pop()] Output: [10, 5] Explanation: push(10): Stack is [10] getMin(): Minimum element is 10 push(5): Stack is [10, 5] getMin(): Minimum element is 5 pop(): Removes 5, stack is [10] Try it on GfG Practice Table of Content [Approach 1] Using an Auxiliary Stack - O(1) Time and O(n) Space [Approach 2] Using a Pair in Stack - O(1) Time and O(n) Space [Approach 3] Without Extra Space- O(1) Time and O(1) Space [Approach 1] Using an Auxiliary Stack - O(1) Time and O(n) Space Use two stacks: one to store actual stack elements and the other as an auxiliary stack to store minimum values. The idea is to do push() and pop() operations in such a way that the top of the auxiliary stack is always the minimum. Let us see how push() and pop() operations work. Push(int x) push x to the first stack (the stack with actual elements) compare x with the top element of the second stack (the auxiliary stack). Let the top element be y. If x is smaller than y then push x to the auxiliary stack. If x is greater than y then push y to the auxiliary stack. Pop() pop the top element from the auxiliary stack. pop the top element from the actual stack int getMin() Return the top element of the auxiliary stack. C++ #include <iostream> #include <stack> using namespace std ; class SpecialStack { stack < int > st ; stack < int > minStack ; public : void push (int x) { st . push (x); // If the minStack is empty or the new element is smaller than // the top of minStack, push it onto minStack if (minStack . empty () || x <= minStack . top ()) { minStack . push (x); } else { // Otherwise, push the top element of minStack // again to keep the minimum unchanged minStack . push (minStack . top ()); } } // Pop the top element from the stack void pop () { if (st . empty ()) { return ; } // Pop from both stacks st . pop (); minStack . pop (); } // Return the top element of the stack without removing it int peek () { if (st . empty ()) { return -1 ; } return st . top (); } // Get the minimum element in the stack int getMin () { if (minStack . empty ()) { return -1 ; } return minStack . top (); } }; int main () { SpecialStack st ; st . push (18); st . push (19); st . push (29); st . push (15); st . push (16); cout << st . getMin () << endl ; return 0 ; } Java import java.util.Stack ; class SpecialStack { Stack < Integer > st = new Stack <> (); Stack < Integer > minSt = new Stack <> (); public void push (int x) { st . push (x); // If the minSt is empty or the new element is smaller than // the top of minSt, push it onto minSt if (minSt . isEmpty () || x <= minSt . peek ()) { minSt . push (x); } else { // Otherwise, push the top element of minSt // again to keep the minimum unchanged minSt . push (minSt . peek ()); } } // Pop the top element from the stack public void pop () { if (st . isEmpty ()) { return ; } // Pop from both stacks st . pop (); minSt . pop (); } // Return the top element of the stack without removing it public int peek () { if (st . isEmpty ()) { return - 1 ; } return st . peek (); } // Get the minimum element in the stack public int getMin () { if (minSt . isEmpty ()) { return - 1 ; } return minSt . peek (); } } public class GfG { public static void main (String [] args) { SpecialStack st = new SpecialStack (); st . push (18); st . push (19); st . push (29); st . push (15); st . push (16); System . out . println (st . getMin ()); } } Python class SpecialStack : def __init__ (self): self . st = [] self . minSt = [] def push (self , x): self . st . append (x) # If minSt is

empty or new element is smaller than # the top of minSt, push it if not self . minSt or x <= self . minSt [- 1]: self . minSt . append (x) else : # Otherwise, repeat the top of minSt self . minSt . append (self . minSt [- 1]) # Pop the top element def pop (self): if not self . st : return self . st . pop () self . minSt . pop () # Return top element def peek (self): if not self . st : return - 1 return self . st [- 1] # Get the minimum element def getMin (self): if not self . minSt : return - 1 return self . minSt [- 1] if __name__ == '__main__' : st = SpecialStack () st . push (18) st . push (19) st . push (29) st . push (15) st . push (16) print (st . getMin ()) C# using System ; using System.Collections.Generic ; class SpecialStack { Stack < int > st = new Stack < int > (); Stack < int > minSt = new Stack < int > (); public void push (int x) { st . Push (x); // If minSt is empty or new element is smaller than // the top of minSt, push it if (minSt . Count == 0 || x <= minSt . Peek ()) { minSt . Push (x); } else { // Otherwise, repeat the top of minSt minSt . Push (minSt . Peek ()); } } // Pop the top element public void pop () { if (st . Count == 0) { return ; } st . Pop (); minSt . Pop (); } // Return top element public int peek () { if (st . Count == 0) { return - 1 ; } return st . Peek (); } // Get the minimum element public int getMin () { if (minSt . Count == 0) { return - 1 ; } return minSt . Peek (); } } class GfG { static void Main () { SpecialStack st = new SpecialStack (); st . push (18); st . push (19); st . push (29); st . push (15); st . push (16); Console . WriteLine (st . getMin ()); } } JavaScript class SpecialStack { constructor () { this . st = []; this . minStack = []; } push (x) { this . st . push (x); // If the minStack is empty or the new element is smaller than // the top of minStack, push it onto minStack if (this . minStack . length === 0 || x <= this . minStack [this . minStack . length - 1]) { this . minStack . push (x); } else { // Otherwise, push the top element of minStack // again to keep the minimum unchanged this . minStack . push (this . minStack [this . minStack . length - 1]); } } // Pop the top element from the stack pop () { if (this . st . length === 0) { return ; } // Pop from both stacks this . st . pop (); this . minStack . pop (); } // Return the top element of the stack without removing it peek () { if (this . st . length === 0) { return - 1 ; } return this . st [this . st . length - 1]; } // Get the minimum element in the stack getMin () { if (this . minStack . length === 0) { return - 1 ; } return this . minStack [this . minStack . length - 1]; } } // Driver Code const st = new SpecialStack (); st . push (18); st . push (19); st . push (29); st . push (15); st . push (16); console . log (st . getMin ()); Output 15 Time Complexity: For insert operation: O(1) (As insertion 'push' in a stack takes constant time) For delete operation: O(1) (As deletion 'pop' in a stack takes constant time) For 'Get Min' operation: O(1) (As we have used an auxiliary stack which has it's top as the minimum element) Auxiliary Space: O(n) [Approach 2] Using a Pair in Stack - O(1) Time and O(n) Space This approach uses a stack where each element is stored as a pair: the element itself and the minimum value up to that point. When an element is pushed, the minimum is updated. The getMin() function directly accesses the minimum value from the top of the stack in constant time, ensuring that both push(), pop(), and getMin() operations are O(1). This approach efficiently tracks the minimum value without needing to traverse the stack. C++ #include <iostream> #include <stack> using namespace std ; class SpecialStack { private : stack < pair < int , int >> st ; public : SpecialStack () { } // Add an element to the top of stack void push (int x) { int newMin = st . empty () ? x : min (x , st . top (). second); st . push ({ x , newMin }); } // Remove the top element from the stack void pop () { if (! st . empty ()) { st . pop (); } } // Return top element of the stack int peek () { if (st . empty ()) { return -1 ; } return st . top (). first ; } // Find minimum element of the stack int getMin () { if (st . empty ()) { return -1 ; } return st . top (). second ; } }; int main () { SpecialStack st ; st . push (2); st . push (3); cout << st . peek () << " " ; st . pop (); cout << st . getMin () << " " ; st . push (1); cout << st . getMin () << " " ; } Java import java.util.Stack ; class SpecialStack { private Stack < int []> st ; public SpecialStack () { st = new Stack <> (); } // Add an element to the top of Stack public void push (int x) { int newMin = st . isEmpty () ? x : Math . min (x , st . peek () [1]); st . push (new int [] { x , newMin }); } // Remove the top element from the Stack public void pop () { if (! st . isEmpty ()) { st . pop (); } } // Returns top element of the Stack public int peek () { return st . isEmpty () ? - 1 : st . peek () [0] ; } // Finds minimum element of Stack public int getMin () { return st . isEmpty () ? - 1 : st . peek () [1] ; } public static void main (String [] args) { SpecialStack st = new SpecialStack (); // Function calls st . push (2); st . push (3); System . out . print (st . peek () + " "); st . pop (); System . out . print (st . getMin () + " "); st . push (1); System . out . print (st . getMin () + " "); } } Python class SpecialStack : def __init__ (self): self . st = [] # Add an element to the top of Stack def push (self , x): newMin = x if not self . st else min (x , self . st [- 1][1]) self . st . append ((x , newMin)) # Remove the top element from the Stack def pop (self): if self . st : self . st . pop () # Returns top element of the Stack def peek (self): return - 1 if not self . st else self . st [- 1][0] # Finds minimum element of Stack def getMin (self): return - 1 if not self . st else self . st [- 1][1] if __name__ == "__main__" : st = SpecialStack () st . push (2) st . push (3) print (st . peek (), end = " ") st . pop () print (st . getMin (), end = " ") st . push (1) print (st . getMin (), end = " ") C#

using System ; using System.Collections.Generic ; class SpecialStack { private Stack < (int , int) > st ; public SpecialStack () { st = new Stack < (int , int) > (); } // Add an element to the top of Stack public void push (int x) { int newMin = st . Count == 0 ? x : Math . Min (x , st . Peek (). Item2); st . Push ((x , newMin)); } // Remove the top element from the Stack public void pop () { if (st . Count > 0) { st . Pop (); } } // Returns top element of the Stack public int peek () { return st . Count == 0 ? - 1 : st . Peek (). Item1 ; } // Finds minimum element of Stack public int getMin () { return st . Count == 0 ? - 1 : st . Peek (). Item2 ; } public static void Main () { SpecialStack st = new SpecialStack (); st . push (2); st . push (3); Console . Write (st . peek () + " "); st . pop (); Console . Write (st . getMin () + " "); st . push (1); Console . Write (st . getMin () + " "); } } JavaScript class SpecialStack { constructor () { this . st = []; } // Add an element to the top of Stack push (x) { let newMin = this . st . length === 0 ? x : Math . min (x , this . st [this . st . length - 1][1]); this . st . push ([x , newMin]); } // Remove the top element from the Stack pop () { if (this . st . length > 0) { this . st . pop (); } } // Returns top element of the Stack peek () { return this . st . length === 0 ? - 1 : this . st [this . st . length - 1][0]; } // Finds minimum element of Stack getMin () { return this . st . length === 0 ? - 1 : this . st [this . st . length - 1][1]; } } // Driver Code const st = new SpecialStack (); st . push (2); st . push (3); console . log (st . peek () + " "); st . pop (); console . log (st . getMin () + " "); st . push (1); console . log (st . getMin () + " "); Output 3 2 1 [Approach 3] Without Extra Space- O(1) Time and O(1) Space The idea is to use a variable minEle to track the minimum element in the stack. Instead of storing the actual value of minEle in the stack, we store a modified value when pushing an element smaller than minEle. Push(x) If the stack is empty, push x and set minEle = x. If x >= minEle, push x normally. If x < minEle, push 2*x - minEle and update minEle = x (this encodes the previous min). Pop() Remove the top element. If the removed element is >= minEle, no change in minEle. If the removed element is < minEle, update minEle = 2*minEle - top (decoding the previous min). Peek() Returns minEle if the top is modified (encoded) or top otherwise. getMin() Returns minEle, the current minimum in O(1) time. How this approach works: When pushing a new element x that is smaller than minEle, we store 2*x - minEle in the stack and update minEle = x. This encoded value is always less than x, signaling that minEle has changed. Why 2*x - minEle < x: Since x < minEle, x - minEle < 0 Adding x on both sides: 2*x - minEle < x So, the stack stores a “flag” value less than x to remember the previous minimum. Restoring previous min when popping: Let y be the popped value (y = 2*x - prevMinEle). Current minEle = x. Restore previous minimum: prevMinEle = 2*minEle - y = 2*x - (2*x - prevMinEle) = prevMinEle This way, we can always track the current minimum in O(1) time using only one stack. C++ #include <iostream> #include <stack> using namespace std ; class SpecialStack { private : stack < int > st ; int minEle ; public : SpecialStack () { minEle = -1 ; } // Add an element to the top of stack void push (int x) { if (st . empty ()) { minEle = x ; st . push (x); } // If new number is less than minEle else if (x < minEle) { st . push (2 * x - minEle); minEle = x ; } else { st . push (x); } } // Remove the top element from the stack void pop () { if (st . empty ()) return ; int top = st . top (); st . pop (); // Minimum will change if min element is removed if (top < minEle) { minEle = 2 * minEle - top ; } } // Return top element of the stack int peek () { if (st . empty ()) return -1 ; int top = st . top (); return (minEle > top) ? minEle : top ; } // Return minimum element of the stack int getMin () { if (st . empty ()) return -1 ; return minEle ; } }; int main () { SpecialStack st ; st . push (2); st . push (3); cout << st . peek () << " " ; st . pop (); cout << st . getMin () << " " ; st . push (1); cout << st . getMin () << " " ; } Java import java.util.Stack ; class SpecialStack { private Stack < Integer > st ; private int minEle ; public SpecialStack () { st = new Stack <> (); minEle = - 1 ; } // Add an element to the top of stack public void push (int x) { if (st . isEmpty ()) { minEle = x ; st . push (x); } // If new number is less than minEle else if (x < minEle) { st . push (2 * x - minEle); minEle = x ; } else { st . push (x); } } // Remove the top element from the stack public void pop () { if (st . isEmpty ()) return ; int top = st . pop (); // Minimum will change if min element is removed if (top < minEle) { minEle = 2 * minEle - top ; } } // Return top element of the stack public int peek () { if (st . isEmpty ()) return - 1 ; int top = st . peek (); // If minEle > top, minEle stores value of top return (minEle > top) ? minEle : top ; } // Return minimum element of the stack public int getMin () { if (st . isEmpty ()) return - 1 ; // variable minEle stores the minimum element return minEle ; } public static void main (String [] args) { SpecialStack st = new SpecialStack (); st . push (2); st . push (3); System . out . print (st . peek () + " "); st . pop (); System . out . print (st . getMin () + " "); st . push (1); System . out . print (st . getMin () + " "); } } Python class SpecialStack : def __init__ (self): self . st = [] self . minEle = - 1 # Add an element to the top of stack def push (self , x): if not self . st : self . minEle = x self . st . append (x) # If new number is less than minEle elif x < self . minEle : self . st . append (2 * x - self . minEle) self . minEle = x else : self . st . append (x) # Remove the top element from the stack def pop (self): if not self . st : return top = self . st . pop () # Minimum will change if min element is removed if top < self .

minEle : self . minEle = 2 * self . minEle - top # Return top element of the stack def peek (self): if not self . st : return - 1 top = self . st [- 1] # If minEle > top, minEle stores value of top return self . minEle if self . minEle > top else top # Return minimum element of the stack def getMin (self): if not self . st : return - 1 return self . minEle if __name__ == '__main__' : st = SpecialStack () st . push (2) st . push (3) print (st . peek (), end = " ") st . pop () print (st . getMin (), end = " ") st . push (1) print (st . getMin (), end = " ") C# using System ; using System.Collections.Generic ; class SpecialStack { private Stack < int > st ; private int minEle ; public SpecialStack () { st = new Stack < int > (); minEle = - 1 ; } // Add an element to the top of stack public void push (int x) { if (st . Count == 0) { minEle = x ; st . Push (x); } // If new number is less than minEle else if (x < minEle) { st . Push (2 * x - minEle); minEle = x ; } else { st . Push (x); } } // Remove the top element from the stack public void pop () { if (st . Count == 0) return ; int top = st . Pop (); // Minimum will change if min element is removed if (top < minEle) { minEle = 2 * minEle - top ; } } // Return top element of the stack public int peek () { if (st . Count == 0) return - 1 ; int top = st . Peek (); // If minEle > top, minEle stores value of top return (minEle > top) ? minEle : top ; } // Return minimum element of the stack public int getMin () { if (st . Count == 0) return - 1 ; return minEle ; } static void Main () { SpecialStack st = new SpecialStack (); // Function calls st . push (2); st . push (3); Console . Write (st . peek () + " "); st . pop (); Console . Write (st . getMin () + " "); st . push (1); Console . Write (st . getMin () + " "); } } JavaScript class SpecialStack { constructor () { this . st = []; this . minEle = - 1 ; } // Add an element to the top of stack push (x) { if (this . st . length === 0) { this . minEle = x ; this . st . push (x); } // If new number is less than minEle else if (x < this . minEle) { this . st . push (2 * x - this . minEle); this . minEle = x ; } else { this . st . push (x); } } // Remove the top element from the stack pop () { if (this . st . length === 0) return ; let top = this . st . pop (); // Minimum will change if min element is removed if (top < this . minEle) { this . minEle = 2 * this . minEle - top ; } } // Returns top element of the stack peek () { if (this . st . length === 0) return - 1 ; let top = this . st [this . st . length - 1]; // If minEle > top, minEle stores value of top return this . minEle > top ? this . minEle : top ; } // Returns minimum element of the stack getMin () { if (this . st . length === 0) return - 1 ; return this . minEle ; } } // Driver Code let st = new SpecialStack (); st . push (2); st . push (3); console . log (st . peek (), " "); st . pop (); console . log (st . getMin (), " "); st . push (1); console . log (st . getMin (), " "); Output 3 2 1 Comment Article Tags: Article Tags: Stack DSA Microsoft Amazon Adobe Flipkart Goldman Sachs VMWare Snapdeal Paytm Sapient SAP Labs FactSet Kuliza STL GreyOrange + 12 More