Submitted To:

Engineer Sheharyar Khan

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 4th

Department: CPED

Data Structures and Algorithms

(DSA)  
Lab Report 7

Lab 7 : Stacks

# Example:

### Code:

**class Stack:**

**def \_\_init\_\_(self, capacity):**

**self.capacity = capacity**

**self.top = -1**

**self.stack = [None] \* capacity**

**def push(self, item):**

**if self.is\_full():**

**print("Stack overflow!")**

**return**

**self.top += 1**

**self.stack[self.top] = item**

**def pop(self):**

**if self.is\_empty():**

**print("Stack underflow")**

**return None**

**item = self.stack[self.top]**

**self.stack[self.top] = None**

**self.top -= 1**

**return item**

**def peek(self):**

**if self.is\_empty():**

**return None**

**return self.stack[self.top]**

**def is\_empty(self):**

**return self.top == -1**

**def is\_full(self):**

**return self.top == self.capacity - 1**

**# Example Usage**

**stack = Stack(5)**

**stack.push(1)**

**stack.push(2)**

**stack.push(3)**

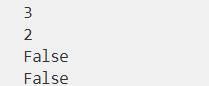
**print(stack.pop())**

**print(stack.peek())**

**print(stack.is\_empty())**

**print(stack.is\_full())**

### Output:



# Tasks:

## 1. Largest Rectangle in Histogram

You are given an array heights[] of size n, where each element represents the height of a bar

in a histogram. Each bar has a width of 1. Find the largest rectangular area that can be

formed in the histogram.

### Code:

**heights = [2, 1, 5, 6, 2, 3]**

**stack = []**

**max\_area = 0**

**for i in range(len(heights)):**

**while stack and heights[i] < heights[stack[-1]]:**

**height = heights[stack.pop()]**

**width = i if not stack else i - stack[-1] - 1**

**max\_area = max(max\_area, height \* width)**

**stack.append(i)**

**while stack:**

**height = heights[stack.pop()]**

**width = len(heights) if not stack else len(heights) - stack[-1] - 1**

**max\_area = max(max\_area, height \* width)**

**print(max\_area)**

### Output:



## 2. Trapping Rainwater Problem

Given an array heights[] of size n, where heights[i] represents the height of the building at

index i, determine the amount of rainwater trapped between the buildings after rainfall.

### Code:

**def trap(height):**

**n = len(height)**

**left\_max = [0] \* n**

**right\_max = [0] \* n**

**left\_max[0] = height[0]**

**right\_max[n - 1] = height[n - 1]**

**for i in range(1, n):**

**left\_max[i] = max(left\_max[i - 1], height[i])**

**for i in range(n - 2, -1, -1):**

**right\_max[i] = max(right\_max[i + 1], height[i])**

**water = 0**

**for i in range(n):**

**water += min(left\_max[i], right\_max[i]) - height[i]**

**return water**

**heights = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]**

**print(trap(heights))**

### Output:



## 3. Find Celebrity in a Party (Stack Approach)

You are given n people at a party, labeled as 0 to n-1. A celebrity is a person who:

1. Knows nobody at the party.

2. Is known by everyone at the party.

You are given a function knows(a, b) which returns True if a knows b, and False otherwise.

Find the celebrity in O(n) time complexity using a stack.

### Code:

**def knows(a, b):**

**M = [[0, 1, 1], [0, 0, 1], [0, 0, 0]]**

**return M[a][b]**

**def find\_celebrity(n, knows):**

**stack = list(range(n))**

**while len(stack) > 1:**

**a = stack.pop()**

**b = stack.pop()**

**if knows(a, b):**

**stack.append(b)**

**else:**

**stack.append(a)**

**candidate = stack.pop()**

**for i in range(n):**

**if i != candidate and (knows(candidate, i) or not knows(i, candidate)):**

**return -1**

**return candidate**

**n = 3**

### Output:

A close up of a computer screen

AI-generated content may be incorrect.

## 4. Design a Special Stack with Two Stacks

Design a stack that supports the following operations in O(1) time:

1. push(x): Push an element onto the stack.

2. pop(): Remove the top element.

3. get\_min(): Get the minimum element in the stack.

4. get\_max(): Get the maximum element in the stack.

### Code:

**class SpecialStack:**

**def \_\_init\_\_(self):**

**self.main\_stack = []**

**self.min\_stack = []**

**self.max\_stack = []**

**def push(self, x):**

**self.main\_stack.append(x)**

**if not self.min\_stack or x <= self.min\_stack[-1]:**

**self.min\_stack.append(x)**

**if not self.max\_stack or x >= self.max\_stack[-1]:**

**self.max\_stack.append(x)**

**def pop(self):**

**if self.main\_stack[-1] == self.min\_stack[-1]:**

**self.min\_stack.pop()**

**if self.main\_stack[-1] == self.max\_stack[-1]:**

**self.max\_stack.pop()**

**return self.main\_stack.pop()**

**def get\_min(self):**

**return self.min\_stack[-1]**

**def get\_max(self):**

**return self.max\_stack[-1]**

**s = SpecialStack()**

**s.push(5)**

**s.push(1)**

**s.push(3)**

**print(s.get\_min())  # 1**

**print(s.get\_max())  # 5**

### Output:

