Data Structures and Algorithms

(DSA)

Lab Report 7

Name: Iqra Fatima

Reg. Number: 23-CP-62

Semester: 4th

Department: CPED

Submitted To:

Engineer Sheharyar Khan

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:

```
3
2
False
False
```

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:

```
10
```

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:

```
6
```

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_L7\t3.py"
2
```

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:

```
1
5
```