**Assignment 11 Questions - Stacks | DSA**

**Question 1** Given an array **arr[ ]** of size **N** having elements, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element. If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

**Example 1 : Input : N = 4, arr[] = [1 3 2 4] Output : 3 4 4 -1**

**------------**

**Solution :** class Solution():

**------------** def findNextGreaterElements(input):

if not input:

         return

      for i in range(len(input)):

         next = -1

         for j in range(i + 1, len(input)):

             if input[j] > input[i]:

                 next = input[j]

                 break

         print(next, end=' ')

if \_\_name\_\_ == '\_\_main\_\_':

     input = [1, 3, 2, 4]

     findNextGreaterElements(input)

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(n).
* Space complexity : O(1).

=====================================x===================================

**Question 2** Given an array **a** of integers of length **n**, find the nearest smaller number for every element such that the smaller element is on left side.If no small element present on the left print -1.

**Example 1 : Input : n = 6 a = {1, 5, 0, 3, 4, 5} Output : -1 1 -1 0 3 4**

**------------**

**Solution :** class Stack():

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

self.stack = []

def isEmpty(self):

return len(self.stack) == 0

def push(self, num):

self.stack.append(num)

def pop(self):

if self.isEmpty():

raise Exception('Stack Underflow')

return self.stack.pop()

def peek(self):

if self.isEmpty():

return None

return self.stack[-1]

def nearestSmallerToLeft(arr):

stack = Stack()

result = []

for i in range(0, len(arr)):

if stack.isEmpty():

result.append(-1)

stack.push(arr[i])

elif not stack.isEmpty():

while(not stack.isEmpty() and arr[i] < stack.peek()):

stack.pop()

if stack.isEmpty():

result.append(-1)

else:

result.append(stack.peek())

stack.push(arr[i])

return result

arr = [**1, 5, 0, 3, 4, 5**]

print(nearestSmallerToLeft(arr))

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(n). As we use only single for loop and all the elements in the stack are push and popped at most once.
* Space complexity : O(n). As extra space is used for storing the elements of the stack.

=====================================x===================================

**Question 3** Implement a Stack using two queues **q1** and **q2**.

**Example 1:** Input : push(2) push(3) pop() push(4) pop() Output : 3 4

**------------**

**Solution :** class Stack():

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

self.q1 = deque()

self.q2 = deque()

def push(self, x):

self.q1.append(x)

def pop(self):

if (not self.q1):

return

while(len(self.q1) != 1):

self.q2.append(self.q1.popleft())

# swap the names of two queues

self.q1, self.q2 = self.q2, self.q1

def top(self):

if (not self.q1):

return

while(len(self.q1) != 1):

self.q2.append(self.q1.popleft())

top = self.q1[0]

self.q2.append(self.q1.popleft())

self.q1, self.q2 = self.q2, self.q1

return top

def size(self):

return len(self.q1)

if \_\_name\_\_ == '\_\_main\_\_':

s = Stack()

s.push(2)

s.push(3)

s.pop()

s.push(4)

s.pop()

print(s.top())

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity :
* Push operation :-  O(1), As, on each push operation the new element is added at the end of the Queue.
* Pop operation :- O(N), As, on each pop operation, all the elements are popped out from the Queue (q1) except the last element and pushed into the Queue (q2).
* Space complexity :  O(N) since 2 queues are used.

=====================================x===================================

**Question 4** You are given a stack **St**. You have to reverse the stack using recursion.

**Example 1 :** Input : St = {3,2,1,7,6} Output : {6,7,1,2,3}

**------------**

**Solution :** class Stack():

**------------** def insertAtBottom(stack, item):

if isEmpty(stack):

push(stack, item)

else:

temp = pop(stack)

insertAtBottom(stack, item)

push(stack, temp)

def reverse(stack):

if not isEmpty(stack):

temp = pop(stack)

reverse(stack)

insertAtBottom(stack, temp)

def createStack():

stack = []

return stack

def isEmpty(stack):

return len(stack) == 0

def push(stack, item):

stack.append(item)

def pop(stack):

if(isEmpty(stack)):

print("Stack Underflow ")

exit(1)

return stack.pop()

def prints(stack):

for i in range(len(stack)-1, -1, -1):

print(stack[i], end=' ')

stack = createStack()

push(stack, str(3))

push(stack, str(2))

push(stack, str(1))

push(stack, str(7))

push(stack, str(6))

reverse(stack)

prints(stack)

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(n2).
* Space complexity : O(N) use of Stack.

=====================================x===================================

**Question 5** You are given a string **S**, the task is to reverse the string using stack.

**Example 1: Input : S="GeeksforGeeks" Output : skeeGrofskeeG**

**------------**

**Solution :** class Stack():

**------------** def createStack():

stack = []

return stack

def size(stack):

return len(stack)

def isEmpty(stack):

if size(stack) == 0:

return true

def push(stack, item):

stack.append(item)

def pop(stack):

if isEmpty(stack):

return

return stack.pop()

def reverse(string):

n = len(string)

stack = createStack()

for i in range(0, n, 1):

push(stack, string[i])

string = ""

for i in range(0, n, 1):

string += pop(stack)

return string

string = " **GeeksforGeeks** "

string = reverse(string)

print("Reversed string is " + string)

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(n)
* Space complexity : O(n) for Stack.

=====================================x===================================

**Question 6** Given string **S** representing a postfix expression, the task is to evaluate the expression and find the final value. Operators will only include the basic arithmetic operators like \***, /, + and -**.

**Example 1: Input : S = "231\*+9-" Output : -4**

**------------**

**Solution :** class Evaluate():

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

self.top = -1

self.capacity = capacity

self.array = []

def isEmpty(self):

return True if self.top == -1 else False

def peek(self):

return self.array[-1]

def pop(self):

if not self.isEmpty():

self.top -= 1

return self.array.pop()

else:

return "$"

def push(self, op):

self.top += 1

self.array.append(op)

def evaluatePostfix(self, exp):

for i in exp:

if i.isdigit():

self.push(i)

else:

val1 = self.pop()

val2 = self.pop()

self.push(str(eval(val2 + i + val1)))

return int(self.pop())

if \_\_name\_\_ == '\_\_main\_\_':

exp = "231\*+9-"

obj = Evaluate(len(exp))

print("postfix evaluation: " (obj.evaluatePostfix(exp)))

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(n).
* Space complexity : O(n).

=====================================x===================================

**Question 8** Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

**Example 1 : Input : height = [0,1,0,2,1,0,1,3,2,1,2,1] Output : 6**

**------------**

**Solution :** def maxWater(arr, n):

**------------** res = 0

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

left = arr[i]

for j in range(i):

left = max(left, arr[j])

right = arr[i]

for j in range(i + 1, n):

right = max(right, arr[j])

res = res + (min(left, right) - arr[i])

return res

if \_\_name\_\_ == "\_\_main\_\_":

arr = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]

n = len(arr)

print(maxWater(arr, n))

**-------------------------**

**Complexity Analysis:**

**-------------------------**

* Time complexity : O(N2). There are two nested loops traversing the array.
* Space complexity : O(1). No extra space is required.

=====================================x===================================