**ASSIGNMENT-XV**

**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  
Explanation:  
In the array, the next larger element  
to 1 is 3 , 3 is 4 , 2 is 4 and for 4 ?  
since it doesn't exist, it is -1.

**Example 2:**

Input:  
N = 5, arr[] [6 8 0 1 3]  
Output:  
8 -1 1 3 -1  
Explanation:  
In the array, the next larger element to  
6 is 8, for 8 there is no larger elements  
hence it is -1, for 0 it is 1 , for 1 it  
is 3 and then for 3 there is no larger  
element on right and hence -1.

**Ans:** def nextGreaterElement(arr):

n = len(arr)

result = [-1] \* n

stack = []

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

while stack and stack[-1] <= arr[i]:

stack.pop()

if stack:

result[i] = stack[-1]

stack.append(arr[i])

return result

arr = [6, 8, 0, 1, 3]

result = nextGreaterElement(arr)

print(result)

**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 = 3  
a = {1, 6, 2}  
Output: -1 1 1  
Explaination: There is no number at the  
left of 1. Smaller number than 6 and 2 is 1.

**Example 2:**

Input: n = 6  
a = {1, 5, 0, 3, 4, 5}  
Output: -1 1 -1 0 3 4  
Explaination: Upto 3 it is easy to see  
the smaller numbers. But for 4 the smaller  
numbers are 1, 0 and 3. But among them 3  
is closest. Similary for 5 it is 4.

**Ans**: def nearestSmallerElements(arr):

n = len(arr)

result = [-1] \* n

stack = []

for i in range(n):

while stack and stack[-1] >= arr[i]:

stack.pop()

if stack:

result[i] = stack[-1]

stack.append(arr[i])

return result

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

result = nearestSmallerElements(arr)

print(result)

**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  
Explanation:  
push(2) the stack will be {2}  
push(3) the stack will be {2 3}  
pop() poped element will be 3 the  
 stack will be {2}  
push(4) the stack will be {2 4}  
pop() poped element will be 4  
  
**Example 2:**

Input:  
push(2)  
pop()  
pop()  
push(3)  
Output:2 –1  
**Ans:** from queue import Queue

class Stack:

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

self.q1 = Queue()

self.q2 = Queue()

def push(self, value):

self.q2.put(value)

while not self.q1.empty():

self.q2.put(self.q1.get())

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

def pop(self):

if self.q1.empty():

return -1

return self.q1.get()

stack = Stack()

stack.push(2)

stack.push(3)

print(stack.pop())

stack.push(4)

print(stack.pop())

**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}

**Example 2:**

Input:St = {4,3,9,6}  
Output:{6,9,3,4}

**Ans**: def reverse\_stack(stack):

if not stack:

return

temp = stack.pop()

reverse\_stack(stack)

insert\_at\_bottom(stack, temp)

def insert\_at\_bottom(stack, item):

if not stack:

stack.append(item)

return

temp = stack.pop()

insert\_at\_bottom(stack, item)

# Insert the popped element at the bottom

stack.append(temp)

stack = [3, 2, 1, 7, 6]

reverse\_stack(stack)

print(stack)

**Question 5**

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

**Example 1:**

Input: S="GeeksforGeeks"  
Output: skeeGrofskeeG

**Ans:** def reverse\_string(S):

stack = []

reversed\_str = ""

for char in S:

stack.append(char)

while stack:

reversed\_str += stack.pop()

return reversed\_str

S = "GeeksforGeeks"

reversed\_str = reverse\_string(S)

print(reversed\_str) # Output: skeeGrofskeeG

**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  
Explanation:  
After solving the given expression,  
we have -4 as result.  
  
**Example 2:**

Input: S = "123+\*8-"  
Output: -3  
Explanation:  
After solving the given postfix  
expression, we have -3 as result.

**Ans:** def evaluate\_postfix\_expression(S):

stack = []

for char in S:

if char.isdigit():

stack.append(int(char))

else:

operand2 = stack.pop()

operand1 = stack.pop()

result = perform\_operation(operand1, operand2, char)

stack.append(result)

return stack.pop()

def perform\_operation(operand1, operand2, operator):

if operator == '\*':

return operand1 \* operand2

elif operator == '/':

return operand1 / operand2

elif operator == '+':

return operand1 + operand2

elif operator == '-':

return operand1 - operand2

65S = "231\*+9-"

result = evaluate\_postfix\_expression(S)

print(result) # Output: -4

**Question 7**

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

* MinStack() initializes the stack object.
* void push(int val) pushes the element val onto the stack.
* void pop() removes the element on the top of the stack.
* int top() gets the top element of the stack.
* int getMin() retrieves the minimum element in the stack.

You must implement a solution with O(1) time complexity for each function.

**Example 1:**

Input

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

Output

[null,null,null,null,-3,null,0,-2]

Explanation

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top(); // return 0

minStack.getMin(); // return –2

**Ans:** class MinStack:

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

self.stack = []

self.min\_stack = []

def push(self, val):

self.stack.append(val)

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

self.min\_stack.append(val)

def pop(self):

if self.stack.pop() == self.min\_stack[-1]:

self.min\_stack.pop()

def top(self):

return self.stack[-1]

def getMin(self):

return self.min\_stack[-1]

minStack = MinStack()

minStack.push(-2)

minStack.push(0)

minStack.push(-3)

print(minStack.getMin())

minStack.pop()

print(minStack.top())

print(minStack.getMin())

**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

Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

Example 2:

Input: height = [4,2,0,3,2,5]

Output: 9

**Ans:** def trap(height):

left = 0

right = len(height) - 1

left\_max = 0

right\_max = 0

water = 0

while left <= right:

if height[left] <= height[right]:

left\_max = max(left\_max, height[left])

water += left\_max - height[left]

left += 1

else:

right\_max = max(right\_max, height[right])

water += right\_max - height[right]

right -= 1

return water

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

print(trap(height))

height = [4, 2, 0, 3, 2, 5]

print(trap(height))