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.

Soln:

```
def nextGreaterElements(arr):
    stack = []
    result = [-1] * len(arr)

    for i in range(len(arr) - 1, -1, -1):
        while stack and stack[-1] <= arr[i]:
            stack.pop()

        if stack:
            result[i] = stack[-1]

        stack.append(arr[i])

    return 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.

Soln:

```
def nearestSmallerElements(a):
    stack = []
    result = [-1] * len(a)

    for i in range(len(a)):
        while stack and stack[-1] >= a[i]:
            stack.pop()
        if stack:
            result[i] = stack[-1]

        stack.append(a[i])

    return result
```

Question 3

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

Soln:

```
class Stack:
    def __init__(self):
        self.q1 = []
        self.q2 = []

    def push(self, x):
        self.q1.append(x)

    def pop(self):
        if self.isEmpty():
            return -1

        while len(self.q1) > 1:
            self.q2.append(self.q1.pop(0))

        popped_element = self.q1.pop(0)

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

        return popped_element

    def top(self):
        if self.isEmpty():
            return -1

        while len(self.q1) > 1:
            self.q2.append(self.q1.pop(0))
        top_element = self.q1[0]

        self.q2.append(self.q1.pop(0))

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

        return top_element

    def isEmpty(self):
        return len(self.q1) == 0
```

Question 4

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

Soln:

```
def insertAtBottom(St, item):
    if len(St) == 0:
        St.append(item)
    else:
        temp = St.pop()
        insertAtBottom(St, item)
        St.append(temp)

def reverseStack(St):
    if len(St) <= 1:
        return

    temp = St.pop()
    reverseStack(St)
    insertAtBottom(St, temp)
```

Question 5

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

Soln:

```
def reverseString(S):
    stack = []
    reversed_str = ""

    # Push each character onto the stack
    for char in S:
        stack.append(char)

    # Pop each character from the stack and append it to reversed_str
    while stack:
        reversed_str += stack.pop()

    return reversed_str
```

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 -**.

Soln:

```
def evaluatePostfixExpression(S):
    stack = [ ]

    # Iterate through each character in the string
    for char in S:
        # If character is an operand, push it onto the stack
        if char.isdigit():
            stack.append(int(char))
        # If character is an operator, perform the corresponding operation
        else:
            operand2 = stack.pop()
            operand1 = stack.pop()

            if char == '*':
                result = operand1 * operand2
            elif char == '/':
                result = operand1 / operand2
            elif char == '+':
                result = operand1 + operand2
            elif char == '-':
                result = operand1 - operand2

            stack.append(result)
    return stack.pop()
```

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.

Soln:

```
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:
            popped_element = self.stack.pop()
            if popped_element == self.min_stack[-1]:
                self.min_stack.pop()

    def top(self):
        if self.stack:
            return self.stack[-1]
        return None

    def getMin(self):
        if self.min_stack:
            return self.min_stack[-1]
        return None
```

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.

Soln:

```
def maxWater(arr, n):

    # To store the maximum water
    # that can be stored
    res = 0

    # For every element of the array
    for i in range(1, n - 1):

        # Find the maximum element on its left
        left = arr[i]
        for j in range(i):
            left = max(left, arr[j])

        # Find the maximum element on its right
        right = arr[i]

        for j in range(i + 1, n):
            right = max(right, arr[j])

        # Update the maximum water
        res = res + (min(left, right) - arr[i])

    return res


# Driver code
if __name__ == "__main__":

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

```
n = len(arr)

print(maxWater(arr, n))
```