Q1

class Solution:

def nextLargerElement(self,arr,n):

ans = [-1 for i in range(n)]

stack = []

for i in range(n):

if not stack:

stack.append(i)

else:

if arr[i] > arr[stack[-1]]:

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

val = stack.pop()

ans[val] = arr[i]

stack.append(i)

else:

stack.append(i)

return ans

Q2

def prevSmaller(self, A):

S = list()

res = []

for i in range(len(A)):

while (len(S) > 0 and S[-1] >= A[i]):

S.pop()

if (len(S) == 0):

res.append(-1)

else:

res.append(S[-1])

S.append(A[i])

return res

Q3

def push(x):

# global declaration

global queue\_1

global queue\_2

queue\_1.append(x)

# code here

#Function to pop an element from stack using two queues.

def pop():

# global declaration

global queue\_1

global queue\_2

if (len(queue\_1) == 0):

return -1

while(len(queue\_1) > 1):

queue\_2.append(queue\_1.pop(0))

out = queue\_1.pop(0)

while(len(queue\_2) != 0):

queue\_1.append(queue\_2.pop(0))

return out

Q4

from typing import List

class Solution:

def reverse(self,st):

#code here

if st==[]:

return st

else:

x = st[0]

st = self.reverse(st[1:])

st.append(x)

return st

Q5

from collections import deque

def solve(x):

s = deque()

for i in x:

# print(i)

s.append(i)

ans = ''

while(s):

ans +=s.pop()

return ans

def reverse(S):

return solve(S)

#Add code here

Q6

from collections import deque

def solve(x):

s = deque()

for i in x:

if ( i not in ['+', '-', '\*', '/']):

s.append(int(i))

else:

# i == \*,-,+,/

a = s.pop()

b = s.pop()

if(i == '+'):

s.append(a+b)

elif(i == '-'):

s.append(b-a)

elif(i =='\*'):

s.append(a\*b)

else:

s.append(b//a)

return s.pop()

class Solution:

def evaluatePostfix(self, S):

return solve(S)

Q7

class stack:

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

self.s=[]

self.minEle=None

def push(self,x):

#CODE HERE

self.s.append(x)

def pop(self):

#CODE HERE

if self.s:

return self.s.pop(-1)

else:

return -1

def getMin(self):

#CODE HERE

if (len(self.s) == 0):

return -1

return min(self.s)

Q8

def trap(self, height: List[int]) -> int:

result = 0

max\_left = height[0]

max\_right = height[-1]

start = 1

end = len(height) - 2

while start <= end:

max\_left = max(max\_left, height[start])

max\_right = max(max\_right, height[end])

if max\_left < max\_right:

result = result + max\_left - height[start]

start = start + 1

else:

result = result + max\_right - height[end]

end = end - 1

return result