**Assignment Questions 16**

**Question 1**

Given an array, for each element find the value of the nearest element to the right which is having a frequency greater than that of the current element. If there does not exist an answer for a position, then make the value ‘-1’.

**Examples:**

Input: a[] = [1, 1, 2, 3, 4, 2, 1]

Output : [-1, -1, 1, 2, 2, 1, -1]

Explanation:

Given array a[] = [1, 1, 2, 3, 4, 2, 1]

Frequency of each element is: 3, 3, 2, 1, 1, 2, 3

Lets calls Next Greater Frequency element as NGF

1. For element a[0] = 1 which has a frequency = 3,

As it has frequency of 3 and no other next element

has frequency more than 3 so '-1'

2. For element a[1] = 1 it will be -1 same logic

like a[0]

3. For element a[2] = 2 which has frequency = 2,

NGF element is 1 at position = 6 with frequency

of 3 > 2

4. For element a[3] = 3 which has frequency = 1,

NGF element is 2 at position = 5 with frequency

of 2 > 1

5. For element a[4] = 4 which has frequency = 1,

NGF element is 2 at position = 5 with frequency

of 2 > 1

6. For element a[5] = 2 which has frequency = 2,

NGF element is 1 at position = 6 with frequency

of 3 > 2

7. For element a[6] = 1 there is no element to its

right, hence -1

Input : a[] = [1, 1, 1, 2, 2, 2, 2, 11, 3, 3]

Output : [2, 2, 2, -1, -1, -1, -1, 3, -1, -1]

**CODE:**

**def** findNearestGreaterFrequency(nums):

stack **=** []

frequencies **=** {}

result **=** []

**for** i **in** range(len(nums) **-** 1, **-**1, **-**1):

**while** stack **and** frequencies**.**get(nums[stack[**-**1]], 0) **<=** frequencies**.**get(nums[i], 0):

stack**.**pop()

**if** **not** stack:

result**.**append(**-**1)

**else**:

result**.**append(nums[stack[**-**1]])

stack**.**append(i)

frequencies[nums[i]] **=** frequencies**.**get(nums[i], 0) **+** 1

**return** result[::**-**1]

**Question 2**

Given a stack of integers, sort it in ascending order using another temporary stack.

**Examples:**

Input : [34, 3, 31, 98, 92, 23]

Output : [3, 23, 31, 34, 92, 98]

Input : [3, 5, 1, 4, 2, 8]

Output : [1, 2, 3, 4, 5, 8]

**CODE:**

**def** sortStack(stack):

tempStack **=** []

**while** stack:

temp **=** stack**.**pop()

**while** tempStack **and** tempStack[**-**1] **>** temp:

stack**.**append(tempStack**.**pop())

tempStack**.**append(temp)

*# Copy elements from tempStack back to stack*

**while** tempStack:

stack**.**append(tempStack**.**pop())

**return** stack

**Question 3**

Given a stack with **push()**, **pop()**, and **empty()** operations, The task is to delete the **middle** element \*\*\*\*of it without using any additional data structure.

Input  : Stack[] = [1, 2, 3, 4, 5]

Output : Stack[] = [1, 2, 4, 5]

Input  : Stack[] = [1, 2, 3, 4, 5, 6]

Output : Stack[] = [1, 2, 4, 5, 6]

**CODE:**

**def** deleteMiddle(stack):

**if** **not** stack:

**return** stack

slowPtr **=** stack

fastPtr **=** stack

prevPtr **=** **None**

**while** fastPtr **and** fastPtr**.**next:

fastPtr **=** fastPtr**.**next**.**next

prevPtr **=** slowPtr

slowPtr **=** slowPtr**.**next

**if** prevPtr:

prevPtr**.**next **=** slowPtr**.**next

**else**:

stack **=** slowPtr**.**next

**return** stack

**Question 4**

Given a Queue consisting of first **n** natural numbers (in random order). The task is to check whether the given Queue elements can be arranged in increasing order in another Queue using a stack. The operation allowed are:

1. Push and pop elements from the stack
2. Pop (Or Dequeue) from the given Queue.
3. Push (Or Enqueue) in the another Queue.

**Examples :**

Input : Queue[] = { 5, 1, 2, 3, 4 }

Output : Yes

Pop the first element of the given Queue

i.e 5. Push 5 into the stack.

Now, pop all the elements of the given Queue and push them to second Queue.

Now, pop element 5 in the stack and push it to the second Queue.

Input : Queue[] = { 5, 1, 2, 6, 3, 4 }

Output : No

Push 5 to stack.

Pop 1, 2 from given Queue and push it to another Queue.

Pop 6 from given Queue and push to stack.

Pop 3, 4 from given Queue and push to second Queue.

Now, from using any of above operation, we cannot push 5 into the second Queue because it is below the 6 in the stack.

**CODE:**

**from** queue **import** Queue

**def** checkOrder(queue):

stack **=** []

sortedQueue **=** Queue()

stack**.**append(queue**.**get())

**while** **not** queue**.**empty():

element **=** queue**.**get()

**while** stack **and** element **<** stack[**-**1]:

sortedQueue**.**put(stack**.**pop())

stack**.**append(element)

**while** stack:

sortedQueue**.**put(stack**.**pop())

*# Check if the elements in sortedQueue are in increasing order*

previous **=** sortedQueue**.**get()

**while** **not** sortedQueue**.**empty():

current **=** sortedQueue**.**get()

**if** current **<** previous:

**return** "No"

previous **=** current

**return** "Yes"

**Question 5**

Given a number , write a program to reverse this number using stack.

**Examples:**

Input : 365

Output : 563

Input : 6899

Output : 9986

**CODE:**

**def** reverseNumber(num):

stack **=** []

*# Extract each digit and push into the stack*

**while** num **>** 0:

digit **=** num **%** 10

stack**.**append(digit)

num **//=** 10

reversedNum **=** 0

*# Pop each digit from the stack and build the reversed number*

**while** stack:

digit **=** stack**.**pop()

reversedNum **=** reversedNum **\*** 10 **+** digit

**return** reversedNum

**Question 6**

Given an integer k and a [**queue**](https://www.geeksforgeeks.org/queue-data-structure/) of integers, The task is to reverse the order of the first **k** elements of the queue, leaving the other elements in the same relative order.

Only following standard operations are allowed on queue.

* **enqueue(x) :** Add an item x to rear of queue
* **dequeue() :** Remove an item from front of queue
* **size() :** Returns number of elements in queue.
* **front() :** Finds front item. </aside>

**CODE:**

**def** reverseFirstK(queue, k):

stack **=** []

**for** \_ **in** range(k):

stack**.**append(queue**.**pop(0))

reversedQueue **=** stack[::**-**1] **+** queue

**return** reversedQueue

**Question 7**

Given a sequence of n strings, the task is to check if any two similar words come together and then destroy each other then print the number of words left in the sequence after this pairwise destruction.

**Examples:**

Input : ab aa aa bcd ab

Output : 3

*As aa, aa destroys each other so,*

*ab bcd ab is the new sequence.*

Input :  tom jerry jerry tom

Output : 0

*As first both jerry will destroy each other.*

*Then sequence will be tom, tom they will also destroy*

*each other. So, the final sequence doesn’t contain any*

*word.*

**CODE:**

**def** countWordsLeft(sequence):

stack **=** []

**for** word **in** sequence:

**if** **not** stack **or** word **!=** stack[**-**1]:

stack**.**append(word)

**else**:

stack**.**pop()

**return** len(stack)

**Question 8**

Given an array of integers, the task is to find the maximum absolute difference between the nearest left and the right smaller element of every element in the array.

**Note:** If there is no smaller element on right side or left side of any element then we take zero as the smaller element. For example for the leftmost element, the nearest smaller element on the left side is considered as 0. Similarly, for rightmost elements, the smaller element on the right side is considered as 0.

**CODE:**

**def** findMaxAbsDifference(arr):

n **=** len(arr)

leftStack **=** []

rightStack **=** []

leftSmaller **=** [0] **\*** n

rightSmaller **=** [0] **\*** n

*# Find nearest left smaller elements*

**for** i **in** range(n):

**while** leftStack **and** leftStack[**-**1] **>=** arr[i]:

leftStack**.**pop()

**if** leftStack:

leftSmaller[i] **=** leftStack[**-**1]

leftStack**.**append(arr[i])

*# Find nearest right smaller elements*

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

**while** rightStack **and** rightStack[**-**1] **>=** arr[i]:

rightStack**.**pop()

**if** rightStack:

rightSmaller[i] **=** rightStack[**-**1]

rightStack**.**append(arr[i])

maxDiff **=** 0

**for** i **in** range(n):

diff **=** abs(leftSmaller[i] **-** rightSmaller[i])

maxDiff **=** max(maxDiff, diff)

**return** maxDiff