ASSIGNMENT-XVI

**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]

**Ans**: def nearest\_greater\_frequency(arr):

frequency = {}

stack = []

result = [-1] \* len(arr)

for num in arr:

frequency[num] = frequency.get(num, 0) + 1

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

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

index = stack.pop()

result[index] = arr[i]

stack.append(i)

return result

**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]

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

temp\_stack = []

while stack:

temp = stack.pop()

while temp\_stack and temp\_stack[-1] > temp:

stack.append(temp\_stack.pop())

temp\_stack.append(temp)

while temp\_stack:

stack.append(temp\_stack.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]

**Ans:** def delete\_middle(stack, position):

if position == 1:

stack.pop()

return

temp = stack.pop()

delete\_middle(stack, position - 1)

stack.append(temp)

def delete\_middle\_element(stack):

size = len(stack)

position = (size // 2) + 1

delete\_middle(stack, position)

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.

**Ans**: def check\_queue\_order(queue):

stack = []

second\_queue = []

expected = 1

while queue:

if queue[0] == expected:

second\_queue.append(queue.pop(0))

expected += 1

elif stack and stack[-1] == expected:

stack.pop()

expected += 1

else:

stack.append(queue.pop(0))

while stack and stack[-1] == expected:

stack.pop()

expected += 1

if not queue and not stack:

return "Yes"

else:

return "No"

**Question 5**

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

**Examples:**

Input : 365  
Output : 563  
  
Input : 6899  
Output : 9986

**Ans:** def reverse\_number(num):

num\_str = str(num)

stack = []

for digit in num\_str:

stack.append(digit)

reversed\_str = ""

while stack:

reversed\_str += stack.pop()

reversed\_num = int(reversed\_str)

return reversed\_num

**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.

**Ans**: def reverse\_k\_elements(queue, k):

stack = []

temp\_queue = []

for \_ in range(k):

stack.append(queue.pop(0))

while queue:

temp\_queue.append(queue.pop(0))

while stack:

queue.append(stack.pop())

while temp\_queue:

queue.append(temp\_queue.pop(0))

return queue

**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.*

**Ans:** def count\_remaining\_words(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.

**Examples:**

Input : arr[] = {2, 1, 8}

Output : 1

Left smaller LS[] {0, 0, 1}

Right smaller RS[] {1, 0, 0}

Maximum Diff of abs(LS[i] - RS[i]) = 1

Input : arr[] = {2, 4, 8, 7, 7, 9, 3}

Output : 4

Left smaller LS[] = {0, 2, 4, 4, 4, 7, 2}

Right smaller RS[] = {0, 3, 7, 3, 3, 3, 0}

Maximum Diff of abs(LS[i] - RS[i]) = 7 - 3 = 4

Input : arr[] = {5, 1, 9, 2, 5, 1, 7}

Output : 1

**Ans:** def max\_absolute\_difference(arr):

n = len(arr)

LS = [0] \* n

RS = [0] \* n

stack = []

for i in range(n):

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

index = stack.pop()

RS[index] = arr[i]

stack.append(i)

stack.clear()

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

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

index = stack.pop()

LS[index] = arr[i]

stack.append(i)

max\_diff = 0

for i in range(n):

diff = abs(LS[i] - RS[i])

if diff > max\_diff:

max\_diff = diff

return max\_diff