Q.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]

Q. 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]

Q. 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]

Q. 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.

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

Input : 365

Output : 563

Input : 6899

Output : 9986

Q. 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.

Q. <aside> 💡 **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.*

Q. 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.

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: Sure! I'll provide solutions for each of the questions you asked.

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’.

```python

def findNearestGreaterFrequency(arr):

frequency = {}

result = []

# Count the frequency of each element

for num in arr:

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

# Find the nearest element with greater frequency

for i in range(len(arr)):

found = False

for j in range(i + 1, len(arr)):

if frequency[arr[j]] > frequency[arr[i]]:

result.append(arr[j])

found = True

break

if not found:

result.append(-1)

return result

```

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

```python

def sortStack(stack):

tempStack = []

while stack:

temp = stack.pop()

while tempStack and tempStack[-1] > temp:

stack.append(tempStack.pop())

tempStack.append(temp)

return tempStack

```

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.

```python

def deleteMiddle(stack):

if not stack:

return stack

size = len(stack)

deleteMiddleUtil(stack, size, 0)

def deleteMiddleUtil(stack, size, curr):

if curr == size // 2:

stack.pop()

return

temp = stack.pop()

deleteMiddleUtil(stack, size, curr + 1)

stack.append(temp)

```

4. Given a Queue consisting of the first n natural numbers (in random order), check whether the given Queue elements can be arranged in increasing order in another Queue using a stack.

```python

def canArrangeQueue(queue):

stack = []

target = 1

while queue:

if queue[0] == target:

queue.pop(0)

target += 1

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

stack.pop()

target += 1

else:

stack.append(queue.pop(0))

while stack:

if stack[-1] == target:

stack.pop()

target += 1

else:

return False

return True

```

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

```python

def reverseNumber(num):

stack = []

while num != 0:

stack.append(num % 10)

num //= 10

reverse = 0

multiplier = 1

while stack:

reverse += stack.pop() \* multiplier

multiplier \*= 10

return reverse

```

6. Given an integer k and a queue of integers, reverse the order of the first k elements of the queue, leaving the other elements in the same relative order.

```python

def reverseK(queue, k):

stack = []

for \_ in range(k):

stack.append(queue.pop(0))

while stack:

queue.append(stack.pop())

for \_ in range(len(queue) - k):

queue.append(queue.pop(0))

```

7. Given a sequence of n strings, 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.

```python

def countWordsLeft(sequence):

stack = []

for word in sequence:

if stack and stack[-1] == word:

stack.pop()

else:

stack.append(word)

return len(stack)

```

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

```python

def maxDifference(arr):

n = len(arr)

LS = [0] \* n

RS = [0] \* n

# Find nearest smaller element on the left side

stack = []

for i in range(n):

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

stack.pop()

if stack:

LS[i] = stack[-1]

stack.append(arr[i])

# Clear the stack

stack.clear()

# Find nearest smaller element on the right side

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

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

stack.pop()

if stack:

RS[i] = stack[-1]

stack.append(arr[i])

# Find the maximum absolute difference

maxDiff = 0

for i in range(n):

maxDiff = max(maxDiff, abs(LS[i] - RS[i]))

return maxDiff

```

These are the solutions to the questions you asked. Let me know if you need any further assistance!