# Assignment 16 Solutions

## Q1. 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]

In [18]:
```
def find_nearest_greater_frequency(arr):
    frequency = {}
    for num in arr:
        frequency[num] = frequency.get(num, 0) + 1

    result = [-1] * len(arr)

    for i in range(len(arr) - 1, -1, -1):
        current_freq = frequency[arr[i]]
        for j in range(i + 1, len(arr)):
            if frequency[arr[j]] > current_freq:
                result[i] = arr[j]
                break

    return result
```

In [19]:
```
arr1 = [1, 1, 2, 3, 4, 2, 1]
print(find_nearest_greater_frequency(arr1))

arr2 = [1, 1, 1, 2, 2, 2, 2, 11, 3, 3]
print(find_nearest_greater_frequency(arr2))
```

```
[-1, -1, 1, 2, 2, 1, -1]
[2, 2, 2, -1, -1, -1, -1, 3, -1, -1]
```

## Q2 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]

In [26]:
```
def sort_stack(stack):
    temp_stack = []

    while stack:
        current = stack.pop()

        while temp_stack and temp_stack[-1] > current:
            stack.append(temp_stack.pop())

        temp_stack.append(current)

    return temp_stack[::-1]
```

In [27]:
```
stack1 = [34, 3, 31, 98, 92, 23]
print(sort_stack(stack1))

stack2 = [3, 5, 1, 4, 2, 8]
print(sort_stack(stack2))
```

```
[98, 92, 34, 31, 23, 3]
[8, 5, 4, 3, 2, 1]
```

## Q3 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]

In [36]:
```
def delete_middle_util(stack, k):
    if k == 1:
        stack.pop()
        return

    temp = stack.pop()
    delete_middle_util(stack, k - 1)
    stack.append(temp)


def delete_middle(stack):
    stack_size = len(stack)
    mid = (stack_size // 2) + 1

    delete_middle_util(stack, mid)
```

In [37]:
```
stack1 = [1, 2, 3, 4, 5]
delete_middle(stack1)
print(stack1)

stack2 = [1, 2, 3, 4, 5, 6]
delete_middle(stack2)
print(stack2)
```

```
[1, 2, 4, 5]
[1, 2, 4, 5, 6]
```

## Q4 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.

In [58]:
```
from queue import Queue

def check_queue_order(queue):
    stack = []
    second_queue = Queue()
    expected = 1

    while not queue.empty():
        num = queue.get()

        if num == expected:
            second_queue.put(num)
            expected += 1
        elif stack and stack[-1] == expected:
            second_queue.put(stack.pop())
            stack.append(num)
        else:
            stack.append(num)

    while stack:
        second_queue.put(stack.pop())

    for i in range(1, second_queue.qsize() + 1):
        if second_queue.get() != i:
            return "No"

    return "Yes"
```

In [59]:
```
# Example usage:
queue1 = Queue()
queue1.put(5)
queue1.put(1)
queue1.put(2)
queue1.put(3)
queue1.put(4)
print(check_queue_order(queue1))

queue2 = Queue()
queue2.put(5)
queue2.put(1)
queue2.put(2)
queue2.put(6)
queue2.put(3)
queue2.put(4)
print(check_queue_order(queue2))
```

```
Yes
No
```

## Q5 Given a number , write a program to reverse this number using stack.

**Examples:**

Input : 365

Output : 563

Input : 6899

Output : 9986

In [70]:
```python
def reverse_number(number):
    stack = []
    number_str = str(number)

    for char in number_str:
        stack.append(char)

    reversed_number_str = ""

    while stack:
        reversed_number_str += stack.pop()

    reversed_number = int(reversed_number_str)
    return reversed_number
```

In [71]:
```python
number1 = 365
print(reverse_number(number1))

number2 = 6899
print(reverse_number(number2))
```

```
563
9986
```

## Q6 Given an integer k and a **queue** 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.

In [82]:
```python
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)

    def is_empty(self):
        return len(self.items) == 0

    def size(self):
        return len(self.items)

    def front(self):
        if not self.is_empty():
            return self.items[0]


def reverse_k_elements(queue, k):
    if queue.size() <= 1 or k <= 0 or k > queue.size():
        return queue

    stack = []

    for _ in range(k):
        stack.append(queue.dequeue())

    while stack:
        queue.enqueue(stack.pop())

    for _ in range(queue.size() - k):
        queue.enqueue(queue.dequeue())

    return queue
```

In [83]:
```python
# Example usage:
queue = Queue()
```

```
queue.enqueue(1)
queue.enqueue(2)
queue.enqueue(3)
queue.enqueue(4)
queue.enqueue(5)

k = 3
reversed_queue = reverse_k_elements(queue, k)

# Print the reversed queue
while not reversed_queue.is_empty():
    print(reversed_queue.dequeue(), end=" ")  # Output: 3 2 1 4 5
```

```
3 2 1 4 5
```

## Q7 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.*

In [94]:
```
def count_remaining_words(sequence):
    stack = []

    for word in sequence:
        if not stack:
            stack.append(word)
        elif word == stack[-1]:
            stack.pop()
        else:
            stack.append(word)

    return len(stack)
```

In [95]:
```
sequence1 = ["ab", "aa", "aa", "bcd", "ab"]
print(count_remaining_words(sequence1))

sequence2 = ["tom", "jerry", "jerry", "tom"]
print(count_remaining_words(sequence2))
```

```
3
0
```

## Q8 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

In [118…

```python
def max_absolute_difference(arr):
    n = len(arr)
    left_smaller = [0] * n
    right_smaller = [0] * n
    diff_list = []

    stack = []
    for i in range(n):
        while stack and stack[-1] >= arr[i]:
            stack.pop()
        if stack:
            left_smaller[i] = stack[-1]
        stack.append(arr[i])

    stack = []
    for i in range(n - 1, -1, -1):
        while stack and stack[-1] >= arr[i]:
            stack.pop()
        if stack:
            right_smaller[i] = stack[-1]
        stack.append(arr[i])

    max_diff = 0
    for i in range(n):
        diff = abs(left_smaller[i] - right_smaller[i])
        max_diff = max(max_diff, diff)

    return max_diff
```

In [119…

```python
# Example usage:
arr1 = [2, 1, 8]
print(max_absolute_difference(arr1))

arr2 = [2, 4, 8, 7, 7, 9, 3]
print(max_absolute_difference(arr2))

arr3 = [5, 1, 9, 2, 5, 1, 7]
print(max_absolute_difference(arr3))
```

```
1
4
1
```

In [ ]: