ASSIGNMENT 16

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'.

```
def find_nearest_greater_frequency(arr):
    stack = []
    frequency = {}
    result = [-1] * len(arr)

    for i in range(len(arr) - 1, -1, -1):
        while stack and frequency[arr[stack[-1]]] <= frequency[arr[i]]:
            stack.pop()

        if stack:
            result[i] = arr[stack[-1]]

        stack.append(i)
        frequency[arr[i]] = frequency.get(arr[i], 0) + 1

    return result
```

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

```
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
```

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.

```
def delete_middle(stack):
    if len(stack) <= 1:
        return stack
    mid = stack.pop()

    stack = delete_middle(stack)

    if len(stack) % 2 == 0:
        stack.append(mid)

    return stack
```

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.

```
from queue import Queue

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

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

      if front == expected:
         expected += 1
      elif stack and stack[-1] == expected:
         other_queue.put(stack.pop())
         expected += 1

      stack.append(front)

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

   for i in range(1, expected):
      if other_queue.get() != i:
         return False

   return True
```

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

```
def reverse_number(number):
   stack = []

   # Extract digits and push them onto the stack
   while number != 0:
      digit = number % 10
      stack.append(digit)
      number //= 10

   # Build the reversed number by popping digits from the stack
   reversed_number = 0
   multiplier = 1
   while stack:
      digit = stack.pop()
      reversed_number += digit * multiplier
      multiplier *= 10

   return reversed_number
```

6. 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.

```
from queue import Queue

def reverse_k_elements(queue, k):
  stack = []

  # Dequeue the first k elements and push them onto the stack
  for _ in range(k):
    stack.append(queue.get())

  # Dequeue the remaining elements and enqueue them back to the queue
  while not queue.empty():
    queue.put(queue.get())

  # Pop the elements from the stack and enqueue them back to the queue
  while stack:
    queue.put(stack.pop())

  return queue
```

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.

```
def count_remaining_words(sequence):
  stack = []
  for word in sequence:
    if stack and stack[-1] == word:
      stack.pop()  # Destroy the similar words
    else:
      stack.append(word)
  return len(stack)
```

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.

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

    # Find nearest left smaller element for each element
    for i in range(n):
        while stack and arr[stack[-1]] >= arr[i]:
            stack.pop()
        left_smaller[i] = stack[-1] if stack else 0
        stack.append(i)

    stack.clear()

    # Find nearest right smaller element for each element
    for i in range(n - 1, -1, -1):
        while stack and arr[stack[-1]] >= arr[i]:
            stack.pop()
        right_smaller[i] = stack[-1] if stack else n - 1
        stack.append(i)

    max_diff = 0

    # Calculate maximum absolute difference
    for i in range(n):
        max_diff = max(max_diff, abs(arr[i] - arr[left_smaller[i]]))
        max_diff = max(max_diff, abs(arr[i] - arr[right_smaller[i]]))

    return max_diff
```