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’.

Soln:

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

    # Count the frequency of each element
    for num in arr:
        frequency[num] = frequency.get(num, 0) + 1

    # Iterate over the array from right to left
    for i in range(len(arr) - 1, -1, -1):
        while stack and frequency[arr[i]] >= frequency[arr[stack[-1]]]:
            stack.pop()

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

        stack.append(i)
    return result
```

Question 2

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

Soln:

```
def sortStack(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

Soln:

```
def deleteMiddle(stack, k):
    # Base case: If stack is empty or has only one element
    if len(stack) == 0 or k == 0:
        stack.pop()
        return

    # Remove the top element
    removed_element = stack.pop()
    # Recursively delete the middle element from the remaining stack
    deleteMiddle(stack, k - 1)

    # If the number of elements is odd, push the removed element back onto the stack
    if k != len(stack) + 1:
        stack.append(removed_element)

def deleteMiddleElement(stack):
    # Calculate the index of the middle element
    k = len(stack) // 2 + 1
    # Delete the middle element
    deleteMiddle(stack, k)
```

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.

Soln:

```
from queue import Queue



def canArrangeInIncreasingOrder(queue):

    stack = []

    second_queue = Queue()

    expected = 1



    while not queue.empty():

        front_element = queue.queue[0]



        if front_element == expected:

            second_queue.put(queue.get())

            expected += 1

        elif len(stack) == 0 or stack[-1] > front_element:

            stack.append(queue.get())

        else:

            return "No"
```

```
    while len(stack) > 0:
        top_element = stack.pop()
        second_queue.put(top_element)

        if top_element != expected:
            return "No"

        expected += 1

    return "Yes"
```

💡 **Question 5**

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

Soln:

```
def reverseNumber(number):
    stack = []
    number_str = str(number)

    # Push each digit onto the stack
    for digit in number_str:
        stack.append(digit)

    reversed_number_str = ""
```

```
    # Pop each digit from the stack and append it to the reversed number string
    while stack:
        reversed_number_str += stack.pop()

    reversed_number = int(reversed_number_str)
    return reversed_number
```

Question 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.

Soln:

```
from queue import Queue

def reverseKElements(queue, k):
    if k > queue.qsize():
        return "Invalid input: k is greater than the size of the queue"

    stack = []
```

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

    # Enqueue the remaining elements back into the queue
    while not queue.empty():
        queue.put(queue.get())

    # Enqueue the elements from the stack back into the queue in reversed order
    while stack:
        queue.put(stack.pop())

    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.

Soln:

```
def countWordsAfterDestruction(words):
    stack = []

    for word in words:
        if not stack or stack[-1] != word:
```

```
            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.**

Soln:

```
def maxDiffBetweenSmallerElements(arr):
    n = len(arr)
    leftStack = []
    rightStack = []
    leftSmaller = [0] * n
    rightSmaller = [0] * n

    # Calculate the nearest smaller element on the left side
    for i in range(n):
        while leftStack and leftStack[-1] >= arr[i]:
```

```
        leftStack.pop()
    if leftStack:
        leftSmaller[i] = leftStack[-1]
    leftStack.append(arr[i])

# Calculate the nearest smaller element on the right side
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])

rightSmaller.reverse()

maxDiff = 0
for i in range(n):
    diff = abs(leftSmaller[i] - rightSmaller[i])
    maxDiff = max(maxDiff, diff)

return maxDiff
```