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’.

Solution :

def find\_nearest\_greater\_elements(arr):

frequency = {}

stack = []

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

# Step 1: Build frequency dictionary

for num in arr:

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

# Step 2 and 3: Find nearest greater elements

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

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

Solution :

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)

# Transfer sorted elements back to original stack

while temp\_stack:

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

return stack

Given a stack with **push()**, **pop()**, and **empty()** operations, The task is to delete the **middle**elementof it without using any additional data structure.

Solution :

def delete\_middle(stack):

if len(stack) == 0:

return

middle\_index = len(stack) // 2

def delete\_middle\_recursive(stk, current\_index):

if current\_index == middle\_index:

stk.pop()

return

temp = stk.pop()

delete\_middle\_recursive(stk, current\_index + 1)

stk.append(temp)

delete\_middle\_recursive(stack, 0)

return stack

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.

Solution :

from queue import Queue

def check\_arrangement\_possible(queue):

n = queue.qsize()

expected\_number = 1

stack = []

auxiliary\_queue = Queue()

while not queue.empty():

front\_element = queue.queue[0]

if front\_element == expected\_number:

auxiliary\_queue.put(front\_element)

queue.get()

expected\_number += 1

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

auxiliary\_queue.put(stack.pop())

expected\_number += 1

elif front\_element > stack[-1]:

return False

else:

stack.append(queue.get())

while stack:

if stack[-1] == expected\_number:

auxiliary\_queue.put(stack.pop())

expected\_number += 1

else:

return False

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

if auxiliary\_queue.get() != i:

return False

return True

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

Solution :

def reverse\_number(number):

number\_str = str(number)

stack = []

# Push each character onto the stack

for char in number\_str:

stack.append(char)

reversed\_str = ""

# Pop each character from the stack and append to reversed string

while stack:

reversed\_str += stack.pop()

reversed\_number = int(reversed\_str)

return reversed\_number

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.

Solution :

from queue import Queue

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

if k <= 0 or k > queue.qsize():

return queue

stack = []

for \_ in range(k):

stack.append(queue.get())

while stack:

queue.put(stack.pop())

for \_ in range(queue.qsize() - k):

queue.put(queue.get())

return queue

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.

Solution :

def count\_remaining\_words(sequence):

stack = []

for word in sequence:

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

stack.pop()

else:

stack.append(word)

remaining\_words = len(stack)

return remaining\_words

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.

Solution :

def max\_absolute\_difference(arr):

n = len(arr)

stack = []

left\_smaller = [0] \* n

right\_smaller = [0] \* n

# Find nearest smaller element on the right

for i in range(n):

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

right\_smaller[stack.pop()] = i

stack.append(i)

# Clear the stack

stack.clear()

# Find nearest smaller element on the left

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

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

left\_smaller[stack.pop()] = i

stack.append(i)

max\_difference = 0

# Calculate maximum absolute difference

for i in range(n):

difference = abs(right\_smaller[i] - left\_smaller[i])

max\_difference = max(max\_difference, difference)

return max\_difference