School of Computing Science & Engineering

Subject: Advanced Data Structures and Algorithms (E2UC503B)

Programming Assignment 1

Last Date: September 15, 2024

Write algorithms, program of following problems. Also find the time and space complexity of each algorithm.

1. Find the largest element in a given array.
2. Reverse a given array.
3. Find the second largest element in a given array
4. Check if a given array is sorted
5. Remove duplicates from a given array
6. Rotate a given array
7. Find the frequency of elements in a given array
8. Merge two sorted arrays

Mritunjay Mishra
22SCSE1410005
Section 1(AI)

Name :- Mritunjay Mishra

Section :- 1 (AI)

Ad. No :- 22SCSE1410005

ADSA

Date ..../..../........

Q.1 Find the Largest Element in a Given Array.

Sol

Algorithm:

1. Initialize a variable max_element to the first element of the array.
2. Itrate through the array.
3. For each element, update max_element if the current element is greater that max_element.
4. Return max_elements.

Code:

```
def find_largest (arr):
    if not arr:
        return None
max_element = arr [0]
 for num in arr:
     if num > max_element:
         max_element = num
return max_element
```

Time Complexity :- O(n)

Space Complexity :- O (1)

Q2) Reverse a given array.

Sol

Algorithm:

1. Initialize two Pointers: One at the beginning (start) and one at the end (end) af the array.
2. Swap the elements at these pointers.
3. Move the (start) pointer forward and the (end) Pointer backward.

4. Repeat until (start) is greater than or equal to (end).

Code:

```
def reverse- array (arr):
    Start, end = 0, len (arr)- 1
while Start < end:
    arr [start], arr[end] = arr[end], arr
                                        [Start]

    Start += 1
    end -= 1
```

Time complexity: O(n)
Space complexity; O (1)

Q.3 Find the second Largest Element in a given Array.

Sol

Algorithm:

1. Initialize two variables: largest and second-largest. Set both to negative infinity.
2. Iterate through the array.
3. Update largest and second-largest accordingly based on the current element.
4. Return second-largest.

Code:

```
def find_second_largest (arr):
    if len (arr) < 2:
        return None
largest = second_largest = float ('-inf')
for num in arr:
    if num > largest:
        second_largest = largest
        largest = num
    elif largest > num > second_largest:
        second_largest = num
  return second_largest
```

Time complexity :- O(n)

Space complexity :- O (1)

Q.4 Check if a given Array is Sorted.

Ans

Algorithm:

1. Iterate through the array from the beginning to the end.
2. Check if each element is less than or equal to the next element.
3. Return True if all elements satisfy this condition otherwise, return False.

Code :

```
def is-Sorted (arr):
    for i in range (len (arr) - 1):
        if arr [i] > arr [i + 1]:
            return False
    return True.
```

Time Complexity : O(n)

Space Complexity : O(1)

Q.5 Remove Duplicates from a given Array.

Sol **Algorithm:**

1. Use a set to track unique elements.
2. Iterate through the array, adding each element to the set.
3. Convert the set back to a list.

Code:

```
def remove_duplicates (arr):
    return list (set (arr))
```

Time complexity: O(n)
Space complexity: O(n)

Q.6 Rotate a Given Array.

Sl :- Algorithm:

1. Determine the number af position to rotate K.
2. Use slicing to rearrange the elements.

Code :

```
def rotate_array (arr, k):
    k = k % len(arr)
    return arr[-k:] + arr [:-k]
```

Time Complexity : O(n)
Space Complexity: O(n)

Q.7 Find the Frequency af Elements in a Given Array.

Sol :- Algorithm :-

1. Use a dictionary to count occurrences af each element.
2. Iterate through the array and update counts in the dictionary.

Q.7 Find the Frequency of Elements in a Given Array.

Sol :- **Algorithm :-**

1. Use a dictionary to count occurrences of each element.
2. Iterate through the array and update counts in the dictionary.

Code :

```
def find- frequency (arr):
      freq = { }
      for num in freq:
            freq [num] += 1
     else:
          freq [num] = 1
 return freq
```

Time Complexity: O(n)
Space Complexity: O(n)

Q.8 Merge Two Sorted Arrays.

Algorithm:

1. Use two pointers, one for each array.
2. Compare elements from both arrays and insert the smaller element into the result array.
3. Continue until all elements from both arrays are processed.

Code :

```
def merge-Sorted-arrays (arr1, arr2):
    merged = []
    i, j = 0,0
  while
      i < len (arr1) and j < len (arr2):
    if arr1[i] < arr2[j]:
    merged. append (arr1 [i])
```

```
            i += 1
    else:
        merged.append (arr2 [j])
            j += 1

      merged.extend (arr1 [i:])
      merged.extend (arr2 [j:])
      return merged
```

Time complexity: O (n+m)
Space complexity: O (n+m)