HW-1

DAA.

Shivani Panchiwala.

UTA ID - 1001982478

Miracle

Pg.No :

Date : / /201

Q.1 Array containing integers from 1 to 10. but one number is missing (9 numbers in the array).

(A). Pesudo code to find the missing number

code.

```
def missing (arr):
    total = 0
    for i in arr:
        total = total + i
    return 55 - total
```

Pesudo steps:-

1. Define initial sum of elements in the array to be 0.
2. Apply a for loop to access every element in the array.
3. In every itteration add new element to total.
4. After loop ends return the answer by subtracting the total from the number 55.

As we have 10 elements,

$$n*(n+1)/2 = 10*11/2 = 55.$$

(B). What is the worst case runtime complexity of your suggested solution.

⇒ As we are iterating the array only ~~one~~ once the time complexity will always remain O(n) for all cases.

Q.2 Given an array of integers:

(A). write a pseudo code to find all pairs of numbers whose sum is equal to a particular num.

⇒ Code.

```
def pairs (n):
    Pairs = list ()
    for i in range (0, n+1):
    for j in range (0, n+1):
        if (i+j == n):
            pairs.append (tuple ([i, j]))
  return Pairs
```

Pseudo steps:-

① Create an empty list.
②. ~~Apply~~ Apply a for loop to access every elements in the array.
③. Apply a second for loop inside the previous loop to get every combination of elements.

4. Add every combination of elements and check if the sum is equal to the particular number.
5. If sum is equal to particular number then add the pair of the elements into the list.

(B). worst case run time complexity of your suggested solution.

⟹ As we are iterating the array twice ~~usin~~ using 2 loops the time complexity will always remain $O(n^2)$.

Q.3 (A). Pesudo code to remove duplicates from your array.

code:

```
def remove_dups (arr, n):
    mp = {i:0 for i in arr}
    for i in range(n):
        if mp[arr[i]] == 0:
            Print (arr[i], end = " ")
            mp[arr[i]] = 1
    return arr
```

Pesudo code steps:-

1. Define a hashmap to store all the elements which have appeared before.

2. Traverse the array.

Miracle Pg.No: Date: / /201

(3). check if the element is present in the hash map.

(4). If yes, continue traversing the array.

(5). Else print the element and store in hashmap.

(B). worst case run time complexity of your suggested solution.

→ As per my solution worst case runtime complexity is O(n).

Q.4 Given 2 sorted arrays:-

(A). Pseudo code to find the median of the two sorted arrays (combined)

Code:
```
def median (array x, array z)
    if array x or array z is empty
        return -1

    declare the final array final
    while (array x and array z have
                untraversed elements)
        if (x[0] > z[0])
            add z[0] to the end of array final
                Move to the next element
        else:
            add x[0] to the end of array
                final
```

move to the next element.

while (x has untraversed elements)
add x[0] to the end of final
move to the next element.

while (z has untraversed elements)
add z[0] to the end of final
move to the next element.

if (len(final) % 2 == 0)
median = (final [middle] +
final [middle +1])/2.
else
median = final [middle +1]

return median.

(B). worst case run time complexity of your suggested solution.

⇒ ~~O(m+n)~~ O(m+n).
where m is length of array 1
n is length of array 2.

Q.5

(A). when does the worst case of quick sort happen and what is the worst case run time complexity in term of big O?

⇒ It depends on the ~~strategy~~ strategy for choosing pivot. In early versions of Quick sort where the leftmost (or rightmost) element is choosen as a pivot, the worst occur in the following cases.

1. Array is already sorted in the same order.
2. Array is already sorted in reverse order.
3. All elements are the same. (a special case of case 1 & 2).

⇒ Therefore the time complexity of the Quick sort algorithm in worst case is

$$[N + (N-1) + (N-2) + (N-3) + \cdots + 2]$$

$$= \left[\frac{N(N+1)}{2} - 1\right] = O(N^2) = O(n^2)$$

(B). When does the best case of bubble sort happen and what is the best case run time complexity in terms of big O?

⇒ The best case for bubble sort occurs when the list is already sorted or nearly sorted. In the case where the list is already sorted, bubble sort will terminate after the first iteration, since no swaps were made.

→ best case runtime complexity is $O(n)$.

(6). What is the runtime complexity of Insertion sort in all 3 cases? Explain the situation which result is best, average & worst case complexity?

⇒ Worse case: $O(n^2)$

When we apply insertion sort on a reverse sorted array, it will insert each element at the beginning of the sorted sub-array, making it the worst time complexity of insertion sort.

⇒ Average case: $O(n^2)$

When the array elements are in random order, the average running time is $O(n^2/4) = O(n^2)$

→ Best case: $O(n)$

When we initiate insertion sort on an already sorted array, it will only compare each element to its predecessor, thereby requiring n steps to sort the already sorted array of n elements.