Name : Prem Atul Jethwa

UTA ID : 1001861810

Assignment ID : 01

1. You have an array containing integers from 1 to 10 (not in order) but one number is missing (there is 9 numbers in the array).

a) Write a pseudo code to find the missing number.

b) What is the worst case run time complexity of your suggested solution.

Answer:

a) Pseudo code to find the missing number

```
Function Find_Missing_Number(array a):
    SumTotal= n*(n+1)/2
    array_Sum=0
    for i in range(list a)
        array_Sum=array_Sum + i
    return SumTotal – array_Sum
```

b) worst case run time complexity of your suggested solution

Since the above suggested solution has one for loop to traverse through all the elements in array the time complexity is O(n)

2. You are given an array of integers:

a) Write a pseudo code to find all pairs of numbers whose sum is equal to a particular number

b) What is the worst case run time complexity of your suggested solution

Answer:

a) pseudo code to find all pairs of numbers whose sum is equal to a particular number

Method 1: (without sorting the array)

Pseudo code

```
Function Find_Pair_Number(array a, n):
length=len(a)
For i in range(0, length)
        For j in range(i+1, length)
                If a[i] +a[j] ==n:
                        Print('pairs of numbers whose sum is equal to ', n, ' : ',a[i] ,a[j])
```

Method 2: (sorting the array)

```
Function Find_Pair_Number(array a, n):
a=merge_Sort(a)
i=0
j=len(a)-1
while(j>i):
        if (a[i] +a[j] == n):
                print('pairs of numbers whose sum is equal to ', n, ' : ',a[i] ,a[j])
        if (a[i] + a[j] > n):
                j =j-1
        else:
                i =i+1
Function merge(array a, array b)
array c
While (a and b have elements)
        if (a[0] >b[0])
        add b[0] to the end of c
        remove b[0] from b
else
```

```
        add a[0] to the end of c
        remove a[0] from a
While (a has elements)
        add a[0] to the end of c
        remove a[0] from a
While (b has elements)
        add b[0] to the end of c
        remove b[0] from b
Return c
Function merge_Sort(array a)
        if (n == 1) return a
        l1 = a[0] ... a[n/2]
        l2 = a[n/2+1] ... a[n]
        l1 = merge_Sort (l1)
        l2 = merge_Sort (l2)
Return merge (l1, l2)
```

b) what is the worst case run time complexity of your suggested solution

Method 1 has two for loop hence the worst case run time complexity is O(n^2)

Method 2 has two parts- sorting the array and find all pairs of numbers.

The respective complexity are – O(nlogn) and O(n)

Hence the worst case run time complexity of the method 2 solution is O(nlogn)

3. You are given an array of integers:

a) Write a pseudo code to remove duplicates from your array.

b) What is the worst case run time complexity of your suggested solution

Answer:

a) pseudo code to remove duplicates from your array

Pseudo Code

```
Function Remove_Duplicate(array a)
If len(a)==0 or len(a)==1
        Return a
Sorted_a=mergesort(a)
Dedup_a=[]
j=0
for i in range(1,len(a)-1)
        if a [i] != a [i+1]:
        Dedup_a [j] = a [i]
        j = j+ 1
Dedup_a [j] = a [i]
j=j+1
Function merge(array a, array b)
array c
While (a and b have elements)
        if (a[0] > b[0])
        add b[0] to the end of c
        remove b[0] from b
else
        add a[0] to the end of c
```

```
        remove a[0] from a
While (a has elements)
        add a[0] to the end of c
        remove a[0] from a
While (b has elements)
        add b[0] to the end of c
        remove b[0] from b
Return c
Function merge_Sort(array a)
        if (n == 1) return a
        l1 = a[0] ... a[n/2]
        l2 = a[n/2+1] ... a[n]
        l1 = merge_Sort (l1)
        l2 = merge_Sort (l2)
Return merge (l1, l2)
```

b) what is the worst case run time complexity of your suggested solution

The above solution consists of two parts: Sorting the array and removing duplicates.

The respective complexity are – O(nlogn) and O(n)

Hence the worst case run time complexity of the method 2 solution is O(nlogn)

4. You are given 2 sorted arrays:

a) Write a pseudo code to find the median of the two sorted arrays (combined).

b) What is the worst case run time complexity of your suggested solution

Answer:

a) pseudo code to find the median of the two sorted arrays

Below are the steps required to find the median of two sorted arrays:

1. Extend the two sorted array and sort them using merge sort.

2. Using the length of new array and calculate the median for the new array.

```
Function merge(array a, array b)
array c
While (a and b have elements)
        if (a[0] > b[0])
        add b[0] to the end of c
        remove b[0] from b
else
        add a[0] to the end of c
        remove a[0] from a
        While (a has elements)
        add a[0] to the end of c
        remove a[0] from a
        While (b has elements)
        add b[0] to the end of c
        remove b[0] from b
Return c
Function merge_Sort(array a)
        if (n == 1) return a
```

```
    l1 = a[0] ... a[n/2]

    l2 = a[n/2+1] ... a[n]

    l1 = merge_Sort (l1)

    l2 = merge_Sort (l2)

    Return merge (l1, l2)


Function Find_Median_Number(array a, array b)

    Sortedb=Merge_Sort(a.extend(b))

    len_of_Sortedb=len(Sortedb)

    If len_of_Sortedb %2==0

    Median= ( Sortedb[len_of_Sortedb/2] + Sortedb[len_of_Sortedb+1/2] ) / 2

    Else:

        Median= Sortedb[len_of_Sortedb/2]
```

b) worst case run time complexity of your suggested solution

The above solution has worst case complexity are – O(nlogn)

5. Answer the following questions:

a) When does the worst case of Quick sort happen and what is the worst case run time complexity in terms of big O?

Answer:

The worst-case complexity of Quick sort depends on the choice of pivot (first element, last element or random element of array/list).

When first or last element of an array is selected as pivot, the worst case occurs for below cases:

1. Array is sorted is same/reverse order (ascending or descending order)

2. All the elements of the array/list are same.

The worst case time complexity of quick sort is 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?

Answer:

The best case of bubble sort happens when the input array is already sorted.

The best case run time complexity of Bubble sort is O(n)

c) What is the runtime complexity of Insertion sort in all 3 cases? Explain the situation which results in best, average and worst case complexity.

Answer:

Below are the runtime complexities of Insertion sort in all 3 cases:

Algo	Time Complexity		
	Best	Average	Worst
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$

Examples of Best, Average and worst Case complexity:

Best Case:

If the input array is already in sorted order, insertion sort compares n elements and performs no swaps.

Example

Average Case/Worst Case:

The average/worst case for insertion sort will occur when the input list is in random/decreasing order.