Design and Analysis of Algorithms

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.

We have 10 elements, so we'll find sum of 10 elements first,

i. e n*(n+1)/2

= 10*11/2

= 55

```
function  remaining_no (array arrayData)

initialize total_sum = 0
initialize k=0
for k in arrayData:

total_sum += arrayData [k]

#return result

 return 55 – total_sum
```

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

The worst-case runtime 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 (10 points).

Define a List of numbers which maybe unsorted/sorted.

List = [1,2,3,4,5,6,7,8,9,0]

Sum = 10 #We initialize ‘sum = particular number’ (For example: 10)

For a in list # Initializing a for loop: To traverse the numbers a in the list

For b in list #Initializing another loop within the initial for
loop to traverse the numbers b in the list.

If a+b == sum #Now Adding a+b and Comparing with ‘sum’

display (a,b)

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

The worst-case runtime complexity is: **$O(n^2)$.**

3. You are given an array of integers:

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

Taking an unsorted list that includes duplicates.

A Function to sort the list

Def MergeSort of (list):

If length of list is > 1 then,

Mid equals length of list

Left = list (first left element to mid element)

Right = list (mid element to right most element)

Recursively call MergeSort for (left sub list) and (right sub list)

Initialize k,j,n = 0

While k is less than length(left) and j is less than length(right)

If left(k) is less than right(j)

list(n) = left(k)

k=k+1

else

List(n) is equal to right(j)

j=j+1

n=n+1

While k is less than length(left)

list(n) = left(k)

k = k+1

n = n+1

While j is less than length(right)

list(n) = right(j)

j= j+1

n=n+1

We call the function MergeSort(list)

Once the list is sorted

We initialise a temporary list called 'updated_list'

For k in range length (list -1)

If list(k) is not equal to list (k+1)

Add to existing list the list(k) to temporary list

Now display the list without duplicates.

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

The worst-case runtime complexity is: **nlogn.**

4. you are given 2 sorted arrays:

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

Initially, taking two sorted lists: listX and listY

Merged_list = listX+ listY

We need to sort the Merged_list:

Function to sort the Merged_list:

```
Def  bubbleSort(Merged_list):
n = len(Merged_list)
  for i in range(n-1):
     for  j  in  range(0,  n-i-1):
          if  Merged_list   [j]  >  Merged_list  [j+1]  :
          Merged_list  [j],  Merged_list  [j+1]  =  Merged_list  [j+1],
          Merged_lierged_list  =  [61,  23,  25,  16,  22,  11,  69]

          bubbleSort(Merged_list)
          print  ("Sorted  Merged  List  is:")

          for  i  in  range(len(Merged_list)): print  ("%d"  % Merged_list  [i])
              Finding the median :

              Median = length of Merged_list //2
              Med = Merged_list(median) //* we store the median element in
              'med' *// Display 'Med' which will display the median of the
              Merged_list.
```

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

The worst-case runtime complexity is: **n logn**

5. Use one of the sorting algorithms to sort the following array input = {4, 1, 2, 7, 10, 1, 2, 4, 4, 7, 1, 2, 1, 10, 1, 2, 4, 1, 2, 7, 10, 1, 2};

Show step by step what happens for the input (no pseudo code is required)

We'll assume k as an index.
We can sort the above array using Insertion sort.
The iterations are as follows :

{4,1,2,7,10,1,2,4,4,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=0 :
{1,4,2,7,10,1,2,4,4,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=1 :
{1,4,2,7,10,1,2,4,4,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=2 :
{1,2,4,7,10,1,2,4,4,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=3 :
{1,2,4,7,10,1,2,4,4,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=4 :
{1,2,4,7,10,1,2,4,4,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=5 :
{1,1,2,4,7,10,2,4,4,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=6 :
{1,1,2,2,4,7,10,4,4,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=7 :
{1,1,2,2,4,4,7,10,4,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=8 :
{1,1,2,2,4,4,4,7,10,7,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=9 :
{1,1,2,2,4,4,4,7,7,10,1,2,1,10,1,2,4,1,2,7,10,1,2}

For k=10 :
{1,1,1,2,2,4,4,4,7,7,10,2,1,10,1,2,4,1,2,7,10,1,2}

For k=11 :
{1,1,1,2,2,2,4,4,4,7,7,10,1,10,1,2,4,1,2,7,10,1,2}

For k=12 :
{1,1,1,1,2,2,2,4,4,4,7,7,10,10,1,2,4,1,2,7,10,1,2}

For k=13 :
{1,1,1,1,2,2,2,4,4,4,7,7,10,10,1,2,4,1,2,7,10,1,2}

For k=14 :
{1,1,1,1,1,2,2,2,4,4,4,7,7,10,10,2,4,1,2,7,10,1,2}

For k=15 :
{1,1,1,1,1,2,2,2,2,4,4,4,7,7,10,10,4,1,2,7,10,1,2}

For k=16 :
{1,1,1,1,1,2,2,2,2,4,4,4,4,7,7,10,10,1,2,7,10,1,2}

For k=17 :
{1,1,1,1,1,1,2,2,2,2,4,4,4,4,7,7,10,10,2,7,10,1,2}

For k=18 :
{1,1,1,1,1,1,2,2,2,2,2,4,4,4,4,7,7,10,10,7,10,1,2}

For k=19 :
{1,1,1,1,1,1,2,2,2,2,2,4,4,4,4,7,7,7,10,10,10,1,2}

For k=20 :
{1,1,1,1,1,1,2,2,2,2,2,4,4,4,4,7,7,7,10,10,10,1,2}

For k=21 :
{1,1,1,1,1,1,1,2,2,2,2,2,4,4,4,4,7,7,7,10,10,10,2}

For k=22 :
{1,1,1,1,1,1,1,2,2,2,2,2,2,4,4,4,4,7,7,7,10,10,10}

6. 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?

The worst case of Quick sort occurs when the Pivot element picked is either the smallest element in the list or the largest element in the list.

Usually, it happens when the list is sorted and in ascending order or it is reverse sorted and in descending order and first or last element is picked.

In terms of Big O, the worst-case run 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?

The Best-case scenario for Bubble sort is when the list is already sorted.

In terms of Big O, the best case run time complexity 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.

Below are runtime complexity for insertion sort :

1. Best Case - **Ω (n)**
2. Average Case - **θ (n^2)**
3. Worst Case - **O (n^2)**

1. The best case of Insertion sort occurs when the array is already sorted. In this case every element is greater than previous element hence insertion sort does not check the values before the previous element again.
Therefore, the best time complexity of the insertion sort is **Ω (n).**

2. In all the other cases except for the best case the algorithm will have to check the elements before the previous element in each iteration.

Therefore, the average case time complexity will be **θ (n^2).**

After considered an average of all the cases. i.e., even when we average the best and worst cases, $(\Omega(n)+O(n^2))/2 = \theta(n+n^2) =$ **θ (n^2)**

3.The worst case will be when the array is sorted in reverse order.

In order to sort this, we required almost n-1 comparisons and n-1 swaps..

Therefore, the worst case time complexity will be **O (n^2).**