Name - Saurabh Kesharwani
Branch - CSE
Rollno - 11911045

DS Assignment

OS Assignment

Ques-1:
Insertion sort algorithum.

```
InsertSort (A)
{
  for ( j = 1 to A length )
    {
      key = A[j]      ;
      i = j - 1       ;
      while ( i >= 0 && A[i] > key )
        {
          A[i+1] = A[i]    ;
            i = i - 1 ;
        }
      A[i+1] = key      ;
    }
}
```

Time complexity of Insertion sort algorithm in best case.

In best case data is arreged in assenting order or desendinging order (i.e. data is sorted).

Example.
List are < 1, 2, 3, 4, 5 >

< 1, 2, 3, 4, 5 >

Date: 4 Page:

	0	1	2	3	
I	ϕ 0	x x x	x x	x	
List data		1	2	3	5
J			x	x	4
Key			2	3	4
Comparison			1	1	1
Movement			1	1	1

∴ Time Complexity are.

$$1 + 1 + 1 + 1 \ldots \ldots \ldots n-1$$

$$n-1 \approx O(n)$$

In best case time complexity is $O(n)$

Ques - 2

(*) Bubble Sort :-

Algorithm :-

```
bubble sort ( int arr[], int n )
{
    int I, J, temp ;
for ( I = 0 ; I < n ; I++ ) → loop 1
    {
        for ( J = 0; J < n-I-1 ; J++ ) loop 2
        {
```

```
        If (arr[I] > arr[I+1])
        {
            temp = arr[I];
            arr[I] = arr[I+1]
            arr[I+1] = temp;
        }
    }
}
```

⊗ Time complexity.

I	loop 1	Inner loop: loop 2	Total
0	n	n-1	(n)(n-1)
1	n-1	n-2	(n-1)(n-2)
2		n-3	
⋮			
n-1		1	

Outer loop 1 operate for (n-1) times & O(n-1)

So, Total time complexity are

$T(n) = (1 + 2 + \dots + n-1)$

$= \frac{n(n-1)}{2}$

$\therefore T(n) \approx O(n^2)$

Space complexity: is Constant i.e O(1)

This is example of in place sorting.

Date: Page:

Quick Sort:

Algorithm :-

```
Quick Sort ( low, high )
{
  if ( low < high )
  {
     j = partition ( low, high ) ;

     Quick Sort ( low , j ) ;
     Quick Sort ( j+1, high ) ;
  }
}

Partition ( low, high )
{
   Pivot = A[low] ;
  i = low,      j = high ;
   while ( i < j )
    {
      do {
           i++ ;
         } while ( A[i] ≤ Pivot ) ;

      do {
           j-- ;
         } while ( A[j] > Pivot ) ;
       if ( i < j )
         Swap ( A[i], A[j] ) ;
    }
   Swap ( A[low], A[j] ) ;
   return j ;
}
```

Date: _____ Page: _____

```
Swap (a, b)
{
    temp = b ;
    b = a ;
    a = temp ;
}
```

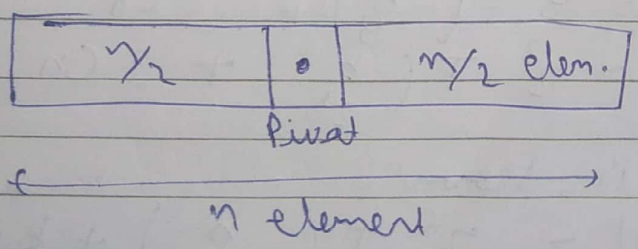
In quick sort
Pivot declaration is important.
Pivot may be any value from list.

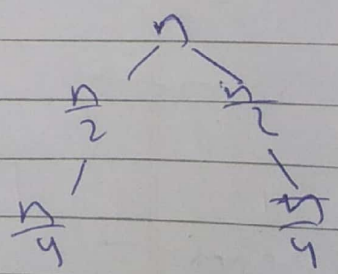
It help to by part arry into two sub-array in which.
a) first contains the elements less than. pivot.
b) Second contains the elements greater. that pivot.

Time Complexity.

a) Best case.
It occurs when list is divided into two equal part.

i.e. Pivot is at middle.





Date: _____ Page: _____

∴ Time complexity expression ar.

from Quick Sort alagritum. we can write time complexity as:

Let $T(n)$ be total amount for Sorting of n element

for $n/2$ element time complexity are $T(\frac{n}{2})$

$$\therefore T(n) = \boxed{cn} + T(n/2) + T(\tfrac{n}{2}) \quad \text{—(I)}$$

↓ Time taken is partition.

Solving abave equation using substitution method.

$$\therefore T(n) = 2T(\tfrac{n}{2}) + cn$$

$$T(\tfrac{n}{2}) = 2T(\tfrac{n}{4}) + \tfrac{cn}{2} \quad \text{—(II)}$$

$$T(\tfrac{n}{4}) = 2T(\tfrac{n}{8}) + \tfrac{cn}{4} \quad \text{—(III)}$$

$$\therefore T(n) = 2\left[2T(\tfrac{n}{4}) + \tfrac{cn}{2}\right] + cn$$

2nd term ← $T(n) = 4T(\tfrac{n}{4}) + 2cn$

$$T(n) = 4\left[2T(\tfrac{n}{8}) + \tfrac{cn}{4}\right] + 2cn$$

$$T(n) = 8T(\tfrac{n}{8}) + 3cn$$

3 term ← $T(n) = 2^3 T(\tfrac{n}{2^3}) + 3cn$

: kth term!

k term ← $T(n) = 2^k T(\tfrac{n}{2^k}) + k.cn$

Let $n = 2^k$

$\log_2 n = k \log_2 2 = k$

$\therefore T(n) = n\,T\left(\frac{n}{n}\right) + Cn \times \log_2 n$

$T(n) = n\,T(1) + Cn \log_2 n$

for $n = 1$ constant amount of time is required $\therefore T(1) = C$

$\therefore T(n) = Cn + Cn \log_2 n$

$\therefore T(n) \approx Cn \log_2 n$

In best case time complexity of quick sort are $O(n \log_2 n)$

b) Worst Case :-

It occurs when the list is already sorted or all number is same in list.

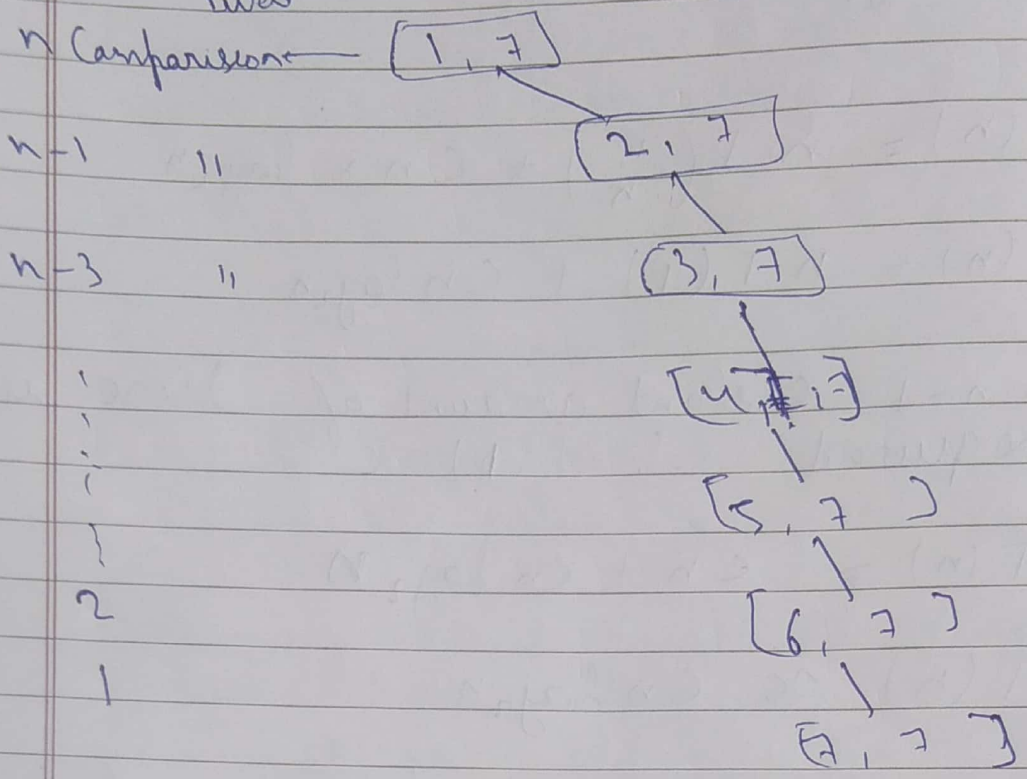
Ex. 1, 2, 3, 4, 5

1, 1, 1, 1, 1

In this partition will occur at begeng of list.

Index 1 2 3 4 5 6 7 Date: Page:

Ex. (2) 4, 8, 10, 16, 18, 20
Pivot



∴ Time complexity are

$$T(n) = 1 + 2 + \cdots \cdots n-1 + n$$
$$= \frac{n(n+1)}{2}$$
$$\approx O(n^2)$$

In worst case.
Time complexity is of $O(n^2)$
In best case.
Time complexity is of $O(n \log n)$

Quick sort is example of in place sorting algorithms.

⊛ Merge sort

Algorithm:

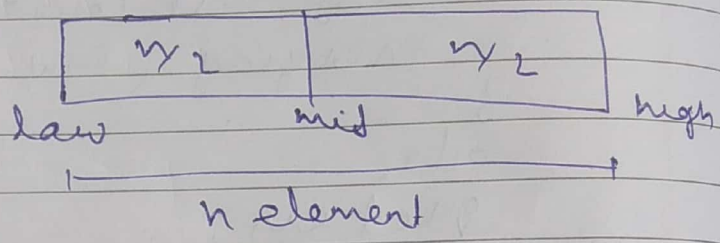
```
void mergesort (int low, int high)
{
    int mid;
    if (low < high)
    {
        mid = (low + high) / 2;
        mergesort (low, mid);          → T(n/2)
        mergesort (mid+1, high);       → T(n/2)
        merging (low, mid high);       → (n
    }
    else
    return;
}
void merging (int low, int mid, int high)
{
    int L1, L2, i;
    for (L1 = low, L2 = mid + 1, i = low; L1 <= mid
        && L2 <= high; i++)
    {
        If (a[L1] <= a[L2])
            b[i] = a[L1++];
        else
            b[i] = a[L2++];
    }
while [L1 <= mid)
    b[i++] = a[L1++];
```

```
while (l2 <= high)
    b[i++] = a[l2++];

for (i = low; i <= high; i++)
    a[i] = b[i]
}
```

(*) **Time complexity :-**

Let the time complexity of merge sort be T(n) for n elements.



by algorithm we can write. T(n) as

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + cn$$

Time taken in merging process.

$$\therefore T(n) = 2T\left(\frac{n}{2}\right) + cn \quad \text{---(I)}$$

Solving this equation with back substitution method.

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + c\frac{n}{2} \quad \text{---(II)}$$

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{cn}{4} \quad \text{---(III)}$$

Date: _____ Page: _____

$\therefore T(n) = 2\left[2T\left(\frac{n}{4}\right) + c\frac{n}{2}\right] + cn$

2nd term $T(n) = 4T\left(\frac{n}{4}\right) + 2cn$

$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{cn}{4}\right] + 2cn$

3 term $T(n) = 8T\left(\frac{n}{8}\right) + 3cn$

⋮

k^{th} term are

k term $T(n) = 2^k T\left(\frac{n}{2^k}\right) + kcn$

let $n = 2^k$

$\log_2 n = k$

$T(n) = nT\left(\frac{n}{n}\right) + \log_2 n \, cn$

$T(n) = cn\log_2 n + n(T(1))$

$\therefore T(1) \approx c$ (for one element constant time will taken)

$\therefore T(n) = cn\log_2 n + cn$

$\approx cn\log_2 n$

So, Time complexity of merge sort are

$$\boxed{T(n) \approx O(n \log n)}$$

and Merge sort space complexity $O(n)$.

Date: Page:

(*) Insertion Sort:

Algorithms:

```
Insertion Sort(A)
{
  for (j = 2 to A length)
  {
    key = A[j] ;
    i = j-1 ;
    while (i >= 0 && A[i] > key)
    {
      A[i+1] = A[i]
      i = i-1
    }
    A[i+1] = key
  }
```

∴ # Time Complexity of Insertion Sort

At worst case ≈ $O(n^2)$
At best case ≈ $O(n)$

and Space complexity of . are ≈ $O(n)$