

Tutorial - 3

①

Name :- Gourang Sinha

Section :- F

Roll No :- 57.

Q 1 for (i = 0 to n)
 {
 if (arr[i] == value)
 // element found
 }
 }

Q 2. Iterative :-

```
void insertion-sort (int A[], int n)
{
    for (int i = 1; i < n; i++)
    {
        j = i - 1;
        x = A[i];
        while (j > -1 && A[j] > x)
        {
            A[j+1] = A[j];
            j--;
        }
        A[j+1] = x;
    }
}
```

Recursive :-

```
void insertion-sort (int arr[], int n)
{
    if (n <= 1)
        return;
    insertion-sort (arr, n-1);
    int last = arr[n-1];
    int j = n-2;
```

while (j >= 0 && arr[j] > last)

{
arr[j+1] = arr[j];
j--;

}

arr[j+1] = last;

}

Insertion sort is called online sort because it does not need to know anything about what values it will sort and the information is requested while the algorithm is running.

Other Sorting algorithm:-

- Bubble Sort
- Quick Sort
- Merge Sort
- Selection Sort
- Heap Sort

Q 3

	Best	Worst	Average
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Q 4.

Inplace Sorting	Stable Sorting	Online Sorting
<ul style="list-style-type: none"> • Bubble • Selection • Insertion • Quick Sort • Merge Heap Sort 	<ul style="list-style-type: none"> • Merge Sort • Bubble • Insertion • Count 	<ul style="list-style-type: none"> • Insertion

Q5. Iterative :-

(3)

```
int binarysearch (int arr[], int l, int r, int key).  
{  
    while (l <= r).  
    {  
        int m = ((l+r)/2);  
        if (arr[m] == key).  
            return m;  
        else if (key < arr[m])  
            r = m-1;  
        else  
            l = m+1;  
    }  
    return -1  
}
```

Recursive :-

```
int binarysearch (int arr[], int l, int r, int key).  
{  
    while (l <= r)  
    {  
        int m = (l+r)/2;  
        if (key == arr[m])  
            return m;  
        else if (key < arr[m])  
            return binarysearch(arr, l, mid-1, key);  
        else  
            return binarysearch(arr, mid+1, r, key);  
    }  
    return -1;  
}
```

Time Complexity :-

- Linear Search - $O(n)$
- Binary Search - $O(\log n)$

Q6. $T(n) = T(n/2) + 1$ — (1)

$T(n/2) = T(n/4) + 1$ — (2)

$T(n/4) = T(n/8) + 1$ — (3)

$T(n) = T(n/2) + 1$

$= T(n/4) + 1 + 1$ from eq (2)

$= T(n/8) + 1 + 1 + 1$ from eq (3)

⋮

$= T(n/2^k) + 1 (k \text{ times})$

Let $2^k = n$

$T(n) = T(n/n) + \log n$

$k = \log n$

$T(n) = T(1) + \log n$

$T(n) = O(\log n)$

Q7.

```
for (int i = 0; i < n; i++)
```

{

```
    for (int j = 0; j < n; j++)
```

{

```
        if (a[i] + a[j] == K)
```

```
            printf("%d %d", i, j);
```

}

}

Q8. Quick Sort is the fastest general-purpose sort. In most practical situations quicksort is the method of choice. If stability is important & space is available, merge sort might be best.

Q9. A pair $(A[i], A[j])$ is said to be inversion is

$A[i] > A[j]$

$i < j$

• Total no. of inversion in given array are 31 using merge sort.

Q10. Worst Case ($O(n^2)$) :- The worst case occurs when the picked ⁽⁵⁾ pivot is always an extreme (small or large) element. This happens when input array is sorted or reverse sorted and either first or last element is picked as pivot.

Best Case ($O(n \log n)$) :- The best case occurs when we will select pivot element as a mean element.

Q11. Merge Sort :- Best Case :- $T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$.
Worst Case :- $T(n) = 2T(n/2) + O(n)$.

Quick Sort :- Best Case :- $T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$.
Worst Case :- $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$

Quick Sort :- The array of elements is divided into parts & repeatedly ~~continue~~ ^{until} it is not possible to divide it further. It is not necessary to divide half.

Merge Sort :- The elements are split into two sub-array ($n/2$) again & again until one element ~~to~~ is left.

Q12

```
for (int i = 0; i < n-1; i++)
{
    int min = i;
    for (int j = i+1; j < n; j++)
    {
        if (a[min] > a[j])
            min = j;
    }
    int key = a[min];
    while (min > i)
    {
        a[min] = a[min-1];
        min--;
    }
    a[i] = key;
}
```

Q13 A better version of bubble sort, known as ^⑥ optimized bubble sort, includes a flag that is set if an exchange is made after an entire pass over the array. If no exchange is made then it should be called the array is already ordered because no two elements need to be switched.

```
void bubble (int a[], int n)
{
    for (int i = 0; i < n; i++)
    {
        int swaps = 0;
        for (int j = 0; j < n - i - 1; j++)
        {
            if (a[j] > a[j+1])
            {
                int t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
                swaps++;
            }
        }
        if (swaps == 0)
            break;
    }
}
```