

TUTORIAL -3

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

2) Iterative
`void insertionSort (int arr[], int n)`
 {
 `for (int i=1; i<n; i++)`
 {
 `j = i-1;`
 `x = arr[i]`
 `while (j > -1 && arr[j] > x)`
 {
 `arr[j+1] = arr[j]`
 `j--;`
 }
 `arr[j+1] = x;`
 }
 }

Recursive

`void insertionSort (int arr[], int n)`
 {
 `if (n <= 1)`
 `return;`
 `insertionSort (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 information is requested while algorithm is running.

Other sorting algorithms are:-

- 1) Bubble sort
- 2) Selection sort
- 3) Quick Sort
- 4) Merge sort
- 5) Heap sort

3) <u>Sorting Algorithm</u>	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	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

<u>Inplace Sorting</u>	<u>Stable Sorting</u>	<u>Online Sorting</u>
Bubble Sort	Merge Sort	Insertion Sort
Selection Sort	Bubble Sort	
Insertion Sort	Insertion Sort	
Quick Sort	Count Sort	
Heap Sort		

5) Iterative :-

```

int binSearch (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 binSearch(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 binSearch(arr, l, mid-1, key);
        else
            return binSearch(arr, mid+1, r, key);
    }
    return -1;
}
```

Time complexity of :-

- (i) Linear search - $O(n)$
- (ii) Binary search - $O(\log n)$

$$6) T(n) = T(n/2) + 1 \quad - (i)$$

$$T(n) = T(n/4) + 1 \quad - (ii)$$

$$T(n) = T(n/8) + 1 \quad - (iii)$$

$$\begin{aligned} T(n) &= T(n/2) + 1 \\ &= T(n/4) + 1 + 1 \\ &= T(n/8) + 1 + 1 + 1 \\ &= T(n/2^k) + 1(k) \end{aligned}$$

Let $j^k = n$
 $K = \log n$

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

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

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

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

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

}

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

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

}

]

8) Quick sort is fastest general purpose sort. In most practical situations quicksort is the method of choice as stability is important and space is available, mergesort might be best.

9) A pair $(a[i], a[j])$ is said to be inversion if:-

(i) $a[i] > a[j]$

(ii) $i < j$

Total number of inversions in given array are 3 using merge sort.

10) Worst case $O(n^2)$:- The worst case occurs when the pivot element is an extreme (smallest / largest) element. This happens when input array is sorted or reversed sorted and either first or the last element is selected as pivot element.

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

11) Merge Sort :-

$$\begin{aligned} \text{Best case}:- T(n) &= 2T(n/2) + O(n) \\ \text{Worst case}:- T(n) &= 2T(n/2) + O(n) \end{aligned} \quad \left. \begin{array}{l} \\ \end{array} \right\} O(n \log n)$$

Quick Sort :-

$$\text{Best case}:- T(n) = 2T(n/2) + O(n) \Rightarrow O(n \log n)$$

$$\text{Worst case}:- T(n) = T(n-1) + O(n) \Rightarrow O(n^2)$$

In quick sort, array of element is divided into 2 parts repeatedly until it is not possible to divide it further.

In merge sort the elements are split into 2 subarray ($n/2$) again and again until only one element is left.

```
12) 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 -= 1;  
            }  
        a[i] = key;  
    }
```

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

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