

Tutorial-3

Q1

```

int linear (int*arr, int n, int key)
    int i = 0 to n-1
        if (arr[i] == key)
            return arr[i]
    return -1
    
```

Q2
Iterative

```

void Insertion (int *arr, int n)
    int i, temp, j;
    for i = 1 to n
        temp = arr[i]
        j = i - 1
        while (j >= 0 AND arr[j] > temp)
            arr[j+1] = arr[j]
            j = j - 1
        arr[j+1] = temp
    
```

Recursive

```

void Insertion (int*arr, int n)
    if (n <= 1)
        return
    Insertion (arr, n-1)
    last = arr[n-1]
    j = n-2
    while (j >= 0 & arr[j] > last)
        arr[j+1] = arr[j]
        j = j - 1
    arr[j+1] = last
    
```

It is called online sorting bcz it does not need to know anything about what values it will sort & the information is requested while the algo is running.

Q3	Algo.	Best Case	Worst Case	Space comp
=	Bubble sort	$O(n^2)$	$O(n^2)$	$O(1)$
	Insertion	$O(n)$	$O(n^2)$	$O(1)$
	Selection	$O(n^2)$	$O(n^2)$	$O(1)$
	Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n)$
	Quick sort	$O(n \log n)$	$O(n^2)$	$O(n)$
	Heap sort	$O(n \log n)$	$O(n \log n)$	$O(1)$

Q4	Sort	Inplace	Stable	Online
=	Selection	✓		
	Insertion	✓	✓	✓
	Merge		✓	
	Quick	✓		
	Heap	✓		
	Bubble	✓	✓	

Q5
 iterative binary

```

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

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

B.C = $O(1)$

Avg = $O(\log n)$

Worst = $O(\log n)$

Recursive

```

int binary (int arr [], int l, int r, int x)
{
    if (l <= r)
    {
        int mid = l + (r-l)/2;
        if (arr[mid] == x)
            return mid;
        else if (arr[mid] > x)
            return binary(arr, l, mid-1, x);
        else
            return binary(arr, mid+1, r, x);
    }
}

```

Q6 $T(n) = T\left(\frac{n}{2}\right) + 1$

Q7

```

map<int, int> m;
for (int i = 0; i < arr.size(); i++)
{
    if (m.find(target = arr[i]) == m.end())
        m[arr[i]] = 1;
    else
        cout << i << " " << m[arr[i]];
}
}

```

Q8 Quick sort is fastest general purpose sort in most practical situation. It is method of choice if stability is important & space is available. Then merge sort is good.

Q9 Inversion for an array indicates how close or far the array is being sorted.

if array is already sorted then inversion count is 0 but if array is sorted in reverse order then inversion count is max
 $arr[] = \{7, 21, 31, 8, 10, 1, 20, 6, 4, 5\}$
 there are 28 inversions in above array

Q10 The worst case occur when picked pivot is an extreme that is when input array is sorted or reverse sorted or either first or last element is picked
 Best case of Quick sort is when we selected pivot as a mean element

Q11 Merge sort $= T(n) = 2T\left(\frac{n}{2}\right) + n$

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

Merge sort works faster than Quick sort in case of large array size
 worst case time complexity of QS is $O(n^2)$
 & merge sort is $O(n \log n)$