

Tutorial - 3

Ques 1 While (low <= high)

```
{
    mid = (low + high) / 2;
    if (arr[mid] == key)
        return true;
    else if (arr[mid] > key)
        high = mid - 1;
    else low = mid + 1;
}
return false;
```

Ques 2 Iterative insertion sort: 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 insertion sort: void insertionSort (int arr[], int n)

Insertion sort is online sorting because whenever a new element come, insertion sort define its right place

```
{
    if (n <= 1)
        return;
    insertionSort (arr, n-1);
    int last = arr[n-1];
    j = n-2;
    while (j >= 0 && arr[j] > last)
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = last;
}
```


ques 3 Bubble sort — $O(n^2)$
 Insertion sort — $O(n^2)$
 Selection sort — $O(n^2)$
 Merge sort — $O(n \cdot \log n)$
 Quick sort — $O(n \log n)$
 Count sort — $O(n \log n)$
 Bucket sort — $O(n)$

ques 4 Online sorting — Insertion sort
 Stable sorting — Merge sort, Insertion sort, Bubble sort
 Inplace sorting — Bubble sort, Insertion sort, Selection sort

ques 5 Iterative Binary Search: While ($low \leq high$)
 { int mid = ($low + high$) / 2;
 if ($arr[mid] == key$)
 return true;
 else if ($arr[mid] > key$)
 high = mid - 1;
 else
 low = mid + 1;
 }

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

Recursive Binary Search: while ($low \leq high$)
 { int mid = ($low + high$) / 2
 if ($arr[mid] == key$)
 return true;
 else if ($arr[mid] > key$)
 Binary_Search(arr, low, mid - 1);
 else
 Binary_Search(arr, mid + 1, high);
 } return false;

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

Ques 6. $T(n) = T(n/2) + T(n/2) + C$

Ques 7

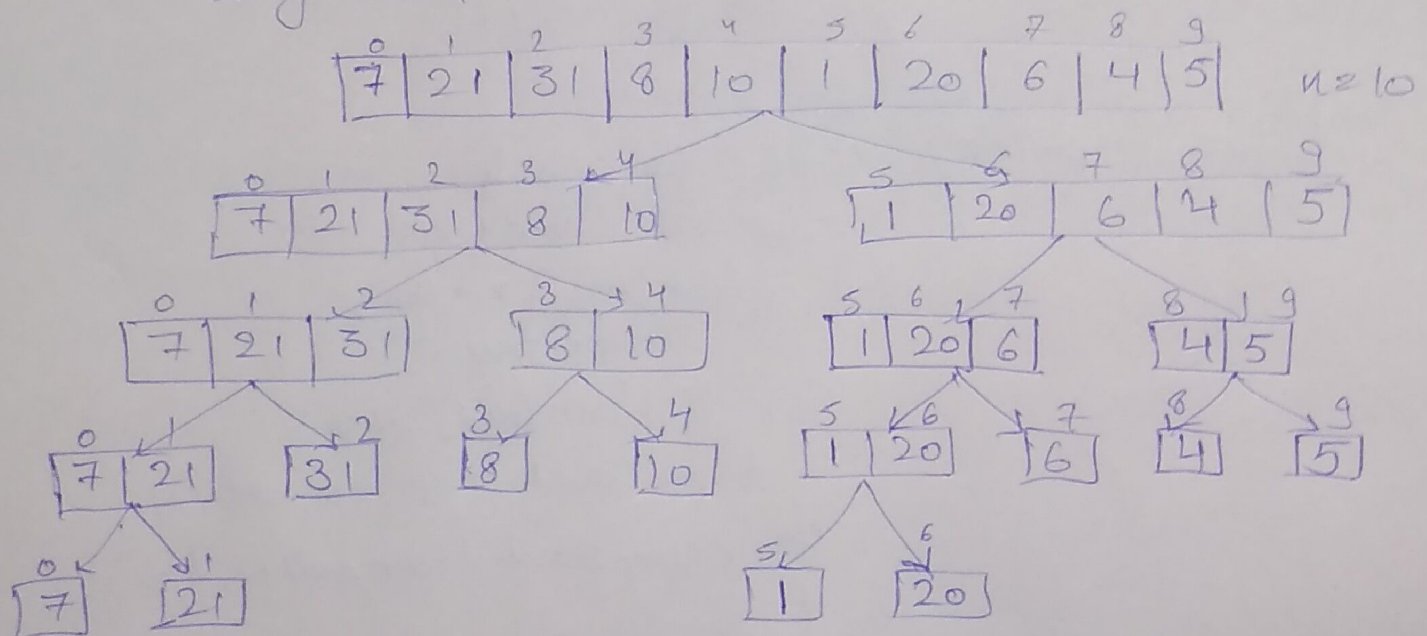
```

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

```

Ques 8. Quicksort is the fastest general purpose sort. In most practical situation, quicksort is the method of choice. If stability is important and space is available, merge sort might be best.

Ques 9. Inversion indicates - how far as close the array is from being sorted.



Inversion = 31

Ans 10 Worst case: The worst case occurs when the Picked Pivot is always an extreme (smallest ~~or~~ largest) element. This happens when input array is sorted ~~or~~ reverse sorted and either first or last element is Picked as Pivot.

$$O(n^2)$$

Best case: Best case occurs when Pivot element is the middle element as near to the middle element

$$O(n \log n)$$

Ans 11 Merge sort: $T(n) = 2T(\frac{n}{2}) + n$

Quick sort: $T(n) = 2T(\frac{n}{2}) + n + 1$

Basic	Quick sort	Merge sort
• Partition	Splitting is done in any ratio	array is Partitioned into Just 2 halves
• Works well on	Smaller array	fine on any size of array
• Addition of space	Less (in-place)	More (Not in-place)
• Efficient	Inefficient for larger array	More efficient
• Sorting Method	Internal	External
• Stability	Not stable	Stable

Aug 14 We will use Merge sort because we can divide the 4 GB data 4 packets of 1 GB and sort them separately and combine them later.

- Internal Sorting: all the data to sort is stored in memory at all times while sorting is in progress.
- External Sorting: all the data is stored outside memory and only loaded into memory in small chunks.