Tutorial - 3

DAA

Name - Ayushman Bhatt
Section - AI & DS
Class Roll No - 21
University Roll No - 2017640

Answer 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;
```

Answer 2

Iterative Insertion Sort:

```
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 Insertion Sort:

```
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--;
  } array[j+1] = last;
}
```

Insertion sort is an online sorting algorithm because whenever a new element comes, Insertion sort defines its right place.

Ans 3

Bubble Sort ⟶ $O(n^2)$
Insertion Sort ⟶ $O(n^2)$
Selection Sort ⟶ $O(n^2)$
Merge Sort ⟶ $O(n * \log n)$ (or just $O(n \log n)$)
Quick Sort ⟶ $O(n * \log(n))$
Count Sort ⟶ $O(n+k)$
Bucket Sort ⟶ $O(n+k)$

Ans 4

Online Sorting → Insertion Sorting.
Stable Sorting → Insertion sort, Merge sort, Bubble sort.
Inplace Sorting → Insertion sort, Selection sort, Bubble sort.

Ans 5

Iterative Binary Search:

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

Time Complexity = $O(\log n)$
Space Complexity = $O(1)$

Recursive Binary Search:

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

Time Complexity = $O(\log n)$
Space Complexity = $O(\log n)$

Answer 6 $T(n) = T(n/2) + T(n/2) + c$ ③

Simplified form of Recurrence Relation for Recursive Binary Search is :-

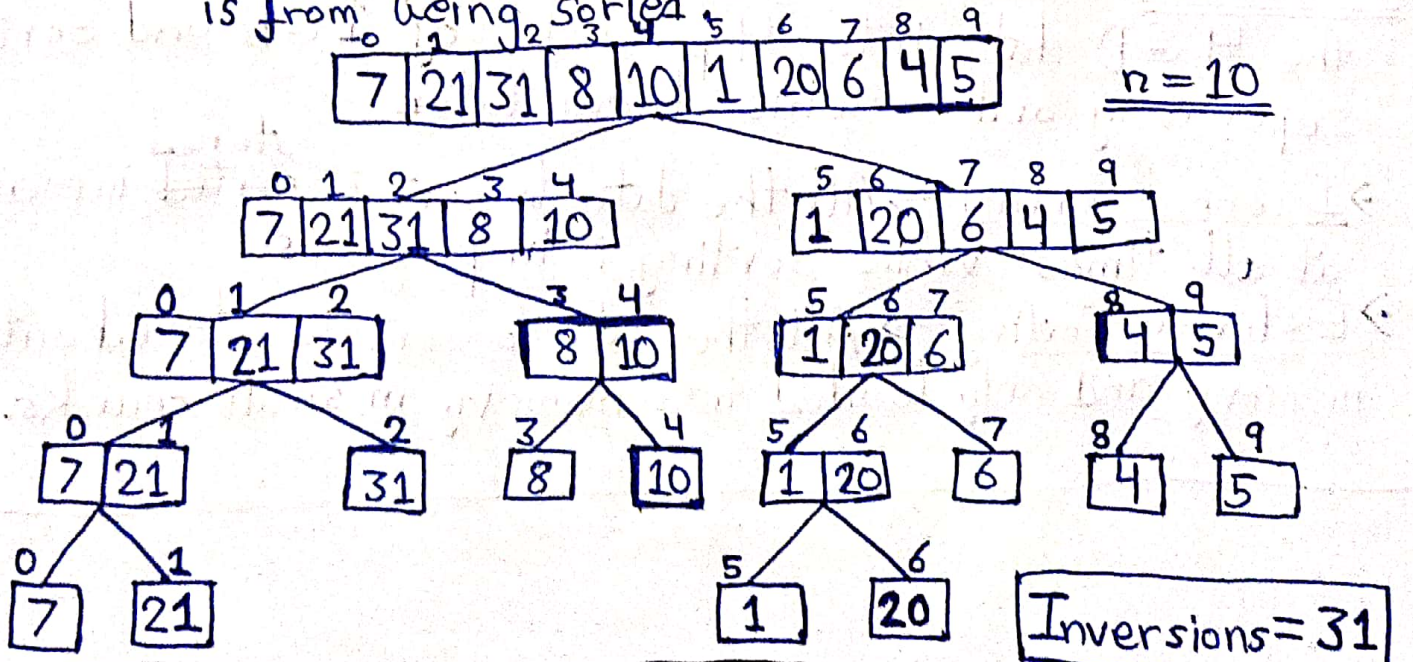
$$\boxed{T(n) = T(n/2) + 1}$$

Answer 7

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

Answer 8 Quick sort is the fastest general purpose sort. In most practical situation, Quick sort is the method of choice.
If stability is important and space is available, Merge sort might be the best.

Answer 9 Inversion indicates how far or close the array is from being sorted



Answer 10 **Worst Case** → The worst case occurs when the picked pivot is always on an extreme element (smallest or largest element). This happens when input array is sorted as reverse sorted array and either first or last element element is picked as pivot.

$$\text{Time Complexity} = O(n^2)$$

Best Case → Best case occurs when pivot element is the middle element or near to the middle element.

$$\text{Time Complexity} = O(n \log n)$$

Answer 11 **Merge Sort**: $T(n) = 2T\left(\frac{n}{2}\right) + n$

Quick Sort: $T(n) = 2T\left(\frac{n}{2}\right) + n + 1$

Basis	Quick Sort	Merge Sort
• Partition	Splitting is done in any ratio.	Array is divided into just 2 parts.
• Works well on	Smaller array.	Fine on any size of array.
• Additional space	Less (in-place).	More (not in-place).
• Efficient	Inefficient for larger array.	More efficient.
• Sorting Method	Internal.	External.
• Stability	Not stable.	Stable.

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

➢ **Internal Sorting** → All the data to sort is ~~sorted~~ stored in memory at all times while sorting is in progress.

➢ **External Sorting** → All the data to sort is stored outside memory and only loaded into memory in small chunks.