Name → Adhyan Dhyani
Class → 4th Sem, Sec-H
Roll no → 20

Date.
Page No.

DAA- Tutorial - 3

Q1
Soln

```
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;
```

Q2
Soln Iterative insertion sort:

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

Date.
Page No.

```
{
    A[j+1] = A[j];
  }  j--;
   A[j+1] = n;
}
```

Recursive insertion sort :-

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

Q3
Solⁿ

Bubble Sort → $O(n^2)$
Insertion Sort → $O(n^2)$
Selection Sort → $O(n^2)$
Merge sort → $O(n * \log n)$
Quick sort → $O(n \log n)$
Count sort → $O(n)$
Bucket sort → $O(n)$

Q4
Soln

Online sorting → Insertion Sort
Stable sorting → Merge sort, Insertion sort, Bubble sort.
Inplace sorting → Bubble sort, Insertion sort, Selection sort.

Q5
Soln Iterative Binary Search : $O(\log n)$

```
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;
}
```

Recursive Binary Search : $O(\log n)$

```
        if (low <= 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;
```

Q6
Sol^n

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

Date.
Page No.

Q7
Solⁿ

```
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<<" "<<mp[arr[i]];
  }
}
```

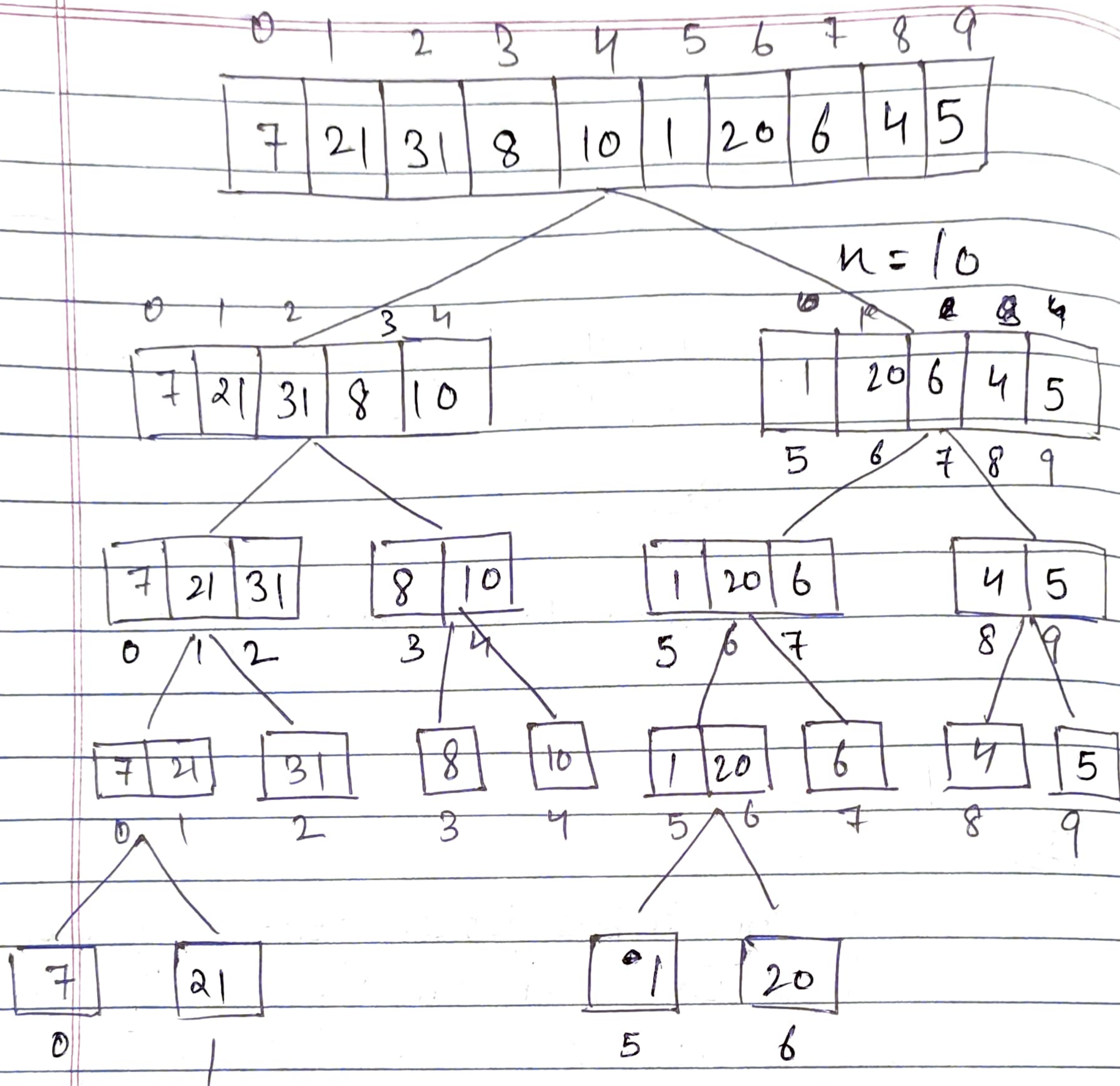
Q8
Solⁿ

Quick sort is the fastest general purpose sort. In most practical solⁿ, quick sort is the method of choice. If stability is important and space is available, merge sort is the best sort

Q9
Solⁿ

Inversion indicators → how far as close the array is from being sorted.

Date.
Page No.



Inversions ⇒ 31

Q10
Sol^n

Worst Case : The worst case occurs when the picked pivot is always an extreme element. This happen when input array is sorted as 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 or near to the middle element.

$$O(n \log n)$$

Q11
Soln

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

Quick Sort : $T(n) = 2T(n/2) + n + 1$

Basis	Quick Sort	Merge sort
• Partition	splitting is done in any ratio	array is halved into 2 equal part.
• Works well on	smaller array	fine on any size of array.

Date.
Page No.

Addition of space	less (in-place)	More (Not-inplace)
efficient	Works faster on small data set	Has consistent speed for any size array
Stability	Not stable	Stable
Method	Internal sorting method	External sorting method
→	good locality of reference	bad locality of reference

Q12
Sol^n

```
void stableselection sort (int a, int n)
{
    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;
```

Date.
Page No.

```
int key = a[min];
while (min > i)
{
    a[min] = a[min-1];
    min--;
}
a[i] = key;
}
}
```

Q13
Sol^n

```
void bubblesort (int a, int n)
{
    int i, j;
    for (i=0; i<n-1; i++)
     for (j=0; j<n-i-1; j++)
      if (arr[j] > arr[j+1])
      {
          int temp;
          temp = arr[j];
          arr[j] = arr[j+1];
          arr[j+1] = temp;
      }
}
```

Date.
Page No.

Q14
Soln We will be using merge sort algorithm because we can ~~divide~~ ~~do~~ divide the 4gb data into 4 pockets of 1gb and sort them seperately 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.