Name → Nitik Singhal

Topic → DAA Assignment -2

Sec/Roll no → M4/38

Q1) → bool linearsearch (int arr, int n, int key)

```
{
    for(int i=0; i<n; i++)
    {
        if (arr[i] == key)
        {
            return true;
        }
        return false;
    }
```

Q2 → Iterative insertion sort

```
for(int i=1; i<n; i++)
{
    int t = a[i];
    int j = i-1;
    while(j>0 && a[j]>t)
    {
        a[j+1] = a[j];
        j--;
    }
    a[j+1] = t;
}
```

Recursive insertion sort

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

* It can sort elements while receiving new ones that's why it is called online sorting

* Other sorting techniques like merge, quick, selection can't do this

Q3→

Sorting technique	Best	Complexity avg	Worst
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Count	$O(n+k)$	$O(n+k)$	$O(n+k)$
Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Radix	$O(d*(n+k))$	$O(d*(n+k))$	$O(d*(n+k))$

Q4→

	Inplace	Stable	Outline sorting
Bubble	✓		
Selection	✓		
Insertion	✓	✓	✓ ☆
Count		✓	
Quick	✓		
Merge		✓	
Heap	✓		
Radix		✓	

05 → Recursive binary search

T(n) → int Search (arr, target, low, high)

```
{
    if (low > high)
        return -1;

    mid = (low + high)/2;
    if (arr[mid] == target)
      { return mid; }
    else if (arr[mid] < target)
      { return Search (arr, target, mid+1, high); }
    else
      { return Search (arr, target, low, mid-1); }.
```

T(n/2) ←

T(n/2) ←

Iterative binary Search

```
int Search (arr, target)
{
    low = 0;
    high = length(arr) - 1;
    while (low <= high)
    {
        mid = (low + high)/2;
        if (arr[mid] == target)
            return mid;
        else if (arr[mid] < target)
            low = mid + 1;
        else
            high = mid - 1;
    } return -1; }
```

	Recursive		Iterative	
Linear	O(n)	O(n)	O(n)	O(1)
Binary	O(log n)	O(log n)	O(log n)	O(1)

Q6→ Refer Q5

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

Q7→
```
int findPairWithSumK(arr, K)
{
    sort(arr);
    int left = 0;
    int right = length(arr) - 1;
    while (left < right)
    {
        if (arr[left] + arr[right] == K)
            { return 1("found"); }
        else if (arr[left] + arr[right] < K)
            { left = left + 1; }
        else
            right = right - 1;
    }
    return -1;
                ↳ ("not found");
}
```

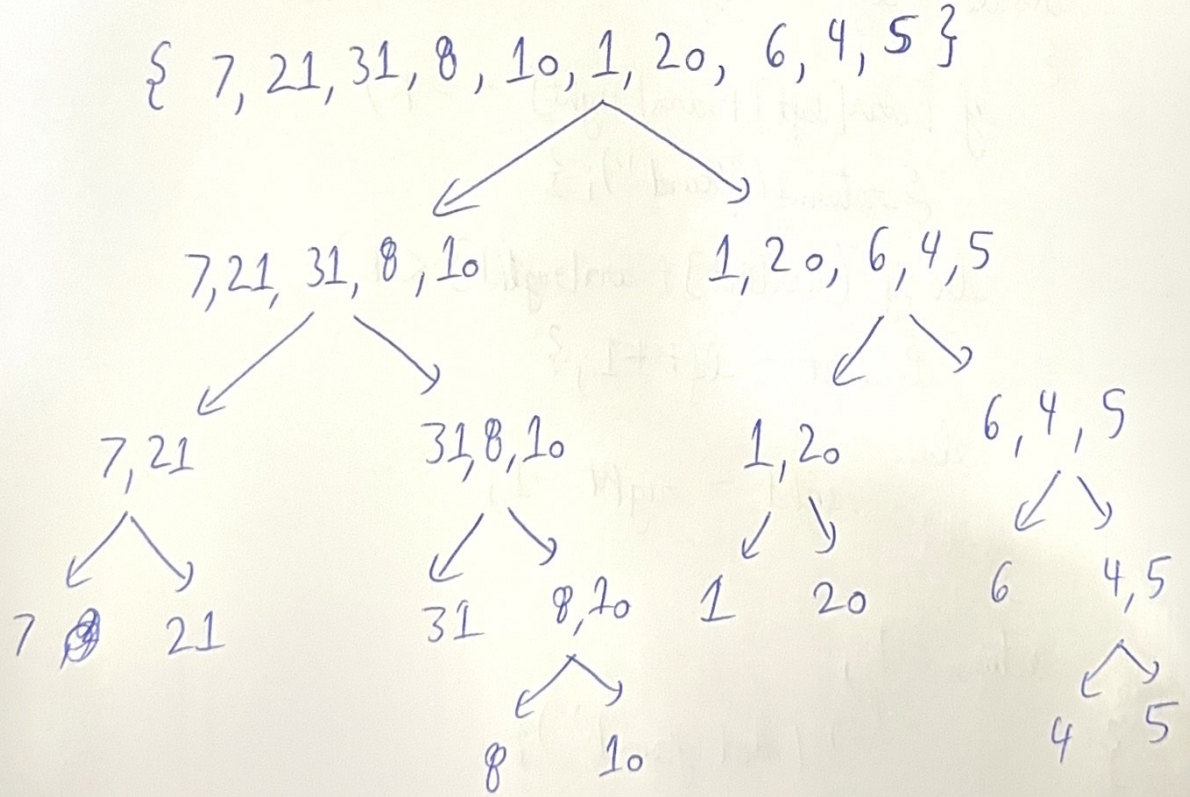
Q8→ Quick sort → Fastest sorting algorithm especially for large dataset

Merge sort → TC remains O(n logn) in all cases
Divide Conquer nature

Heap Sort → TC remains O(n logn)
doesn't require extra space

Insertion sort → Inplace, stable, online sorting

Q9→ Inversion → When smaller element is after larger element or vice versa



Q10 → Best case → When first divides array in equal halves

Worst case → When first divides array in very unbalanced way

Q11 → Merge Sort

★ → Best case : $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

★ → Worst case : $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

Quick Sort

★ → Best case : $T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + O(n)$

★ → Worst case : $T(n) = T(n-1) + O(n)$

Similarities

★ → Both have divide & conquer

★ → $TC = O(n \log n)$

Difference

A → In quick sort TC varies

Q12 →

```
void SelectionSort(arr, n)
{
  for(i=0; i<n; i++)
  {
    int minIndex = i;
    for(j=i+1; j<n; j++)
    {
      if arr[j] < arr[minIndex])
      {
        minIndex = j;
      }
    }
    minValue = arr[minIndex];
    while(minIndex > i)
    {
      arr[minIndex] = arr[minIndex - 1];
      minIndex = minIndex - 1;
    }
    arr[i] = minValue;
  }
}
```

Q13 →

```
void Sort (arr, n)
{
    bool swapped;
    for (int i=0; i< n-1; ++i)
    {
        swapped = false;
        for (int j=0 ; j< n-1; ++j)
        {
            if (arr[j] > arr [j+1])
            {
                swap (arr[j], arr [j+1]);
                swapped = true;
            }
        }
        if (!swapped)
            break;
    }
}
```

Q14 → Merge Sort → Optimised for external sorting

* → Divide and conquer approach
* → Requires small portion of ~~memory~~ data to fit in memory

→

External sorting	Internal sorting
☆→ On virtual memory	☆→ ON RAM
☆→ ex→ Quick and Merge sort	☆→ Bubble, selection and insertion sort