Date:

Ternary Search, Comparision Sort / Selection, Bubble Sort

⇒ Ternary Search

0	1	2	3	4	5	6	7	8
20,	25,	47,	56,	59,	63,	65,	79,	82
		m1				m2		

mid1 = i + (j - i)/3
mid2 = j - (j - i)/3
} Getting 2 mids

i → Left Index
j → right Index

i.e mid1 = 0 + 8/3 = 2
mid2 = 8 - 8/3 = 6

key = 79

1st Path → i, mid1 - 1
2nd Path → mid1 + 1 → mid2 - 1
3rd Path → mid2 + 1, j

↳ 3 paths to access 3 parts of array.

Date:

```
ternarySearch(arr, i, j, key):
    while i<=j: → Covers corner cases
        if arr[mid1] == key:
            return mid1 → Return mid1
        if arr[mid2] == key:
            return mid2
    if key < arr[mid1]:
        return ternarySearch(arr, i, mid1-1,
                             key)
```

↳ Navigating to 1st path since key is less than mid1

```
    elif key > arr[mid2]:
        return ternarySearch(arr, mid2+1, j,
                             key)
```

↳ Since it's > mid2 navigate to 3rd path.

```
    else:
        return ternarySearch(arr, mid1+1,
                             mid2-1, key)
```

↳ Navigating to the 2nd path if key is present

```
        return -1
```

Date:

Recurrence Relation

$T(n) = T(n/3) + C$

$= T(n/3^2) + C + C = T(n/3^2) + 2C$

$= T(n/3^3) + C + C + C = T(n/3^3) + 3C$

↓ k times

$n/3^k = 1$

$n = 3^k \Rightarrow k = \log_3 n$

↳ According to log properties

$T(n/3^k) + C.k$

$\Rightarrow T\left(\frac{n}{3^{\log_3 n}}\right) + C \cdot \log_3 n$

$\Rightarrow T\left(\frac{n}{n^{\log_3 3}}\right) + C \cdot \log_3 n$

$\Rightarrow O(\log_3 n)$ → Time complexity

Date:

Binary Search Ternary Search

$O(\log_2 n)$ > $O(\log_3 n)$

Adobe Interview Question:

Assignment Problem:-

↳ Binary Search is more preferable than Ternery Search why ??

Lower the Base higher the value in log properties

Difference b/w Binary & Ternary Search

↳ Interview Question

(Technical Round Adobe)

Date:

Sorting

Comparision-based Sorting

Compare the elements inside the Array

1) Bubble Sort
2) Selection Sort
3) Insertion Sort
4) Quick Sort } Divide & conquer
5) Merge Sort } module
6) Heap Sort
7) Shell Sort

Non-comparision based sorting

No comparision within the elements.

1) Count Sort
2) Radix Sort
3) Bucket Sort

Stable & Unstable Sort

$20, 10^a, 5, 15, 10^b, 25, 45$

↓ Sorting Algo

$20, 10^a, 10^b, 15, 20, 25, 45$ → Merge Sort.

→ Stable sort

→ $20, 10^b, 10^a, 15, 20, 25, 45$

↳ Not a stable sorting Algo.

Date:

QuickSort, HeapSort
↳ Not stable sorting Algo.

↳ Does stable or unstable sort matter while sorting?

Index	Name	Sort
→23	Suraj	89
47	Suraj	45
150	Suraj	100
171	Suraj	69

↳ Always Stable sort.

Inplace & Outplace Sorting

arr = [_ _ _ _ _ _ _ _ _ _]
↳ extra space → new Array
↳ Outspace sorting => Mergesort
↳ Taking extra space

In place sorting is sorting an array without any extra space.

Date:

Bubble Sort

0	1	2	3	4	5	6	
70	20	50	30	90	5	15	n=7

Pass 1: [20, 50, 30, 70, 5, 15], 90
Largest element ← Last element

Pass 2: [20, 30, 50, 5, 15], 70, 90

Pass 3: [20, 30, 5, 15], 50, 70, 90

Pass 4: [20, 5, 15], 30, 50, 70, 90

Pass 5: 5, 15, 20, 30, 50, 70, 90

Sorted Array

n element → worst case ⇒ (n-1) passes or iterations

# of comparisions :-

$(n-1) + (n-2) + (n-3) \cdots + 1$

$\Rightarrow \frac{(n-1)*n}{2} = O(n^2)$ → All cases

Sum of n natural numbers $\Rightarrow \frac{n(n+1)}{2}$

Swaps:

Best case → 0

worst case → $(n-1)+(n-2)+\cdots+1$

$\Rightarrow O(n^2)$

10, 20, 30, 40 => Best case for swaps

40, 30, 20, 10 => worst case for swaps

Time complexity →

no. of comparisions + no. of swaps

$\downarrow$

$n^2 + n^2$

$\Rightarrow \underline{O(n^2)}$

Why $(n-1)$?

We get one largest number at each iteration & hence we get length-1, length-2 till last element is reached.

Selection Sort

0 1 2 3 4 5 6
70, 20, 50, 30, 90, 5, 15

i=0 → start index
min = 0 & it compares with next index
↳ used to store the index of minimum element.

Pass 1:-
0 1 2 3 4 5 6
5, (20, 50, 30, 90, 70, 15) i=1, min=1
↳ Only one swap happens which reduces the costs.
↳ Smallest number will be at first pass.

Pass 2: 0 1 2 3 4 5 6
5, 15, (50, 30, 90, 70, 20)

Pass 3: 0 1 2 3 4 5 6
5, 15, 20, (30, 90, 70, 50)
i=3 min=3

Pass 4: 0 1 2 3 4 5 6
5, 15, 20, 30, (90, 70, 50)
i=4, min=4 & 6

Pass 5: 5, 15, 20, 30, 50, 70, 90

⇓
Sorted Array

# of comparisions

$\Rightarrow (n-1)+(n-2)+(n-3)+\ldots+1$

$\Rightarrow \frac{(n-1)n}{2} \Rightarrow O(n^2)$

# of swaps $\Rightarrow 1+1+1+1+\ldots+(n-1)$

$\Rightarrow O(n)$

Time complexity $\Rightarrow$ # comparision + # swaps

$\Rightarrow n^2+n$

$\Rightarrow \underline{O(n^2)}$

Here we reduce number of swaps required so selection sort is better than Bubble sort than for lesser cost operations