Date: / / Page no: 7

• Binary Search :-

The binary search algorithms use the divide and conquer approach, it does not scan all the elements in the array, it only search half of the list instead of going through each elements. Hence it is considered the best searching algorithm.

Ex:- Array of 10,00,000 elements.

worst case for linear search ⇒ 10,00,000 -

worst case for Binary search ⇒ 20 ($\log_2 10,00,000$)

worst case ⇒ $O(\log N)$.

• Logic and algorithm :-

For Ascending order

i) Find the middle index.
ii) Compare with target.
iii) if target > middle ⟶ search in the right side
iv) else in the left side
v) if target == middle ⟶ Return middle

arr = [2, 4, 6, 8, 10, (12), 14, 16, 18, 20]

↓ middle

target = 14

i) target > middle
ii) Search in the right part

Date: / / Page no:

arr = [2, 4, 8, 10, (12), [14, 16, (18), 20]]

new middle

now target < middle

So search in the left side.

arr = [2, 4, 8, 10, 12, [(14), 16, 18, 20]]

new middle

target == middle

↳ return middle.

- Better way to find middle element

$$\left\{ \text{Middle} = \text{Start} + \frac{[\text{end} - \text{start}]}{2} \right\}$$

because in int there is some range which might be exceed and we'll be getting errors.

* The one prequisite for binary search is that array should be sorted.

- Order agnostic binary search :-

In this we don't know whether the array is in ascending or descending order.

Date: / / Page no:

o Code for order agnostic binary search :-

⇒
```
public class Main
{
    public static void main (String[] args)
    {
        int [] arr = {2, 4, 6, 8, 10, 12, 14, 27, 31, 100};
        int target = 27;
        int ans = binary_search (arr, target);
        Sout (ans);
    }
    Static int binary_search (int[] arr, int target)
    {
        int start = 0;
        int end = arr.length - 1;
        while (start <= end)
        {
            int middle = start + (end - start) / 2;
            if (arr[middle] == target)
            {
                return target;
            }
            else if (arr[start] < arr[end])
            {
                if (target < arr[middle])
                {
                    end = middle - 1;
                }
                else
                {
```

Date: / / Page no:

```
        start = middle + 1;
    }
}
    else if (arr[start] > arr[end])
{
    if (arr[middle] > target)
    {
        start = middle + 1;
    }
    else
    {
        end = middle - 1;
    }
}
    return -1;
}
}
```

Ques 2 ⇒ Write a code for ceiling number.

⇒ Smallest element in the array which is greater than or equal to target.

```
public class Main
{
    public static void main (String[] args)
    {
        int[] arr = {1, 2, 3, 4, 5, 6, 7, 9, 10};
        int target = 8;
```

```
    System.out.println(ceiling(arr, target));
}
static int ceiling(int[] arr, int target)
{
    int start = 0;
    int end = arr.length - 1;
    while (start <= end)
    {
        int mid = start + (end - start) / 2;
        if (target > arr[mid])
        {
            start = mid + 1;
        }
        else if (target < arr[mid])
        {
            end = mid - 1;
        }
    }
    return start;
}
}
```

⇒ Logic for the above question :-

1st middle → 5 target > middle

second middle → 9 since target < middle

↳ next array [6,7]

start	middle	end
7	7	7

target > middle now -

end = 7 start = 9 . (mid+1)

Since start > end ⇒ while loop break

print start