★ Sorted Array apply Binary search

BINARY SEARCH

Binary search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half.

Condition -

- The data structure must be sorted.
- Access to any element of data structure takes constant time.

Example -

Find the index of element '36' in array num = [2, 4, 6, 9, 11, 12, 14, 20, 36, 48]

Procedure : Loop until target = middle

1. Find the middle element (Between S-E)
2. Check:

   if target > middle → Search in right S = M+1

   else if target < middle → search in left E = M-1

   else target = middle → we return middle value.

Start →	0	1	2	3	4	5	6	7	8	9	← End
	2	4	6	9	11	12	14	20	36	48	target = 36

1. Find middle element:

$$mid = \frac{start + end}{2} = \frac{0 + 9}{2} = 4$$

middle element = 4th index value of array

i.e 11

2. Check:

target > middle val. 36 > 11 ✓ Check in Right

condition True : → Value update

Start = Mid + 1

start = 4 + 1

start = 5th index

New Start → 5 6 7 8 9 → End

			12	14	20	36	48

1. Find middle:

$mid = \frac{start + end}{2} = \frac{5+9}{2} = \frac{14}{2} = 7$

mid = 7th index of array i.e 20

2. check:

target > middle value 36 > 20 True → Check in Right

start = mid + 1

start = 7 + 1

start = 8th index

New Start ← 8 9 → End

		36	48

1. Find middle:

$mid = \frac{start + end}{2} = \frac{8+9}{2} = \frac{17}{2} = 8.5 = 8^{th}$ index ie 36

2. check:

target = middle value 36 = 36

∴ Element found at index 8 (mid).

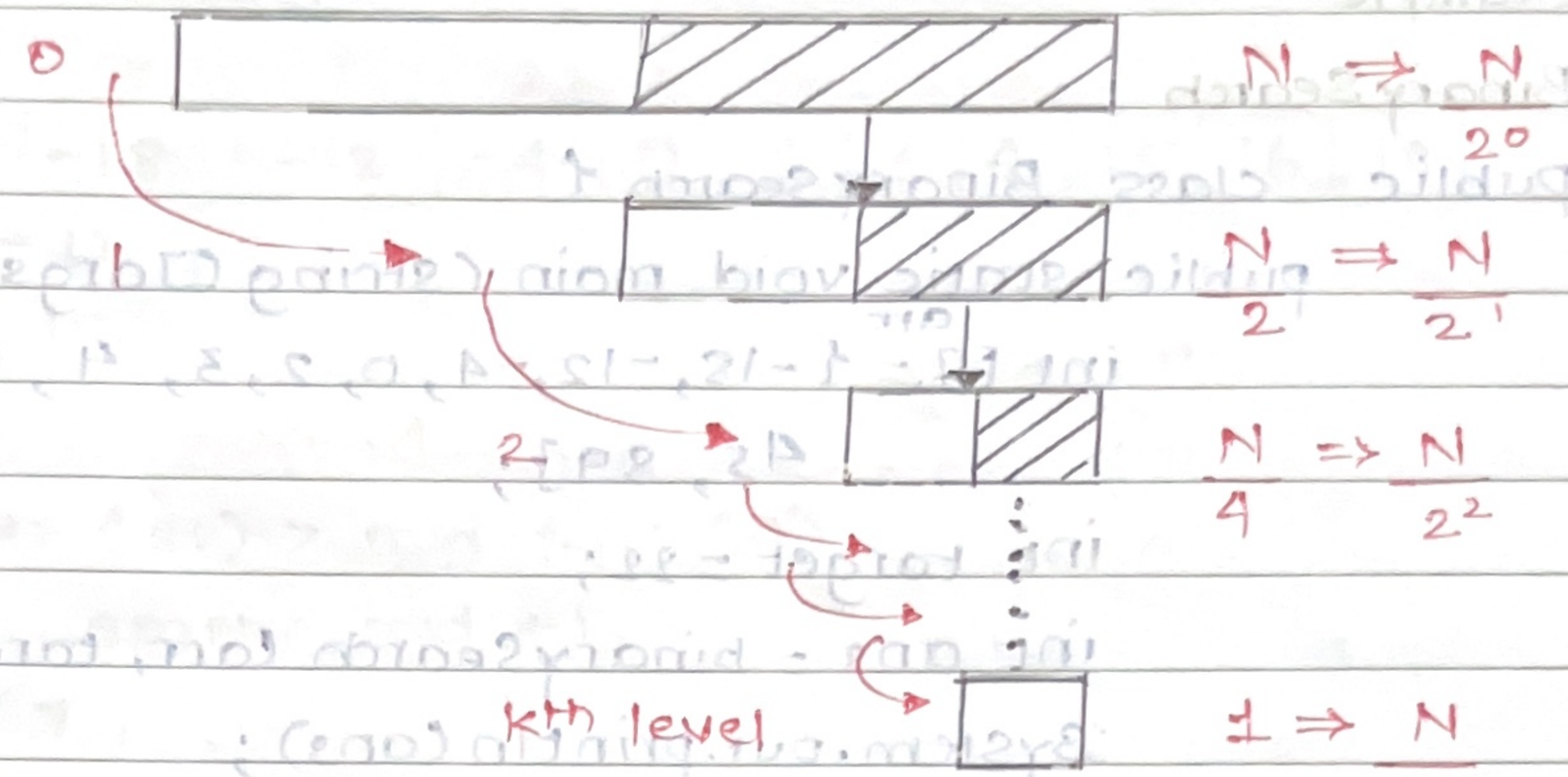
Time complexity -

- Best Case → O(1)

Best case is when the element is at the middle index of array. It only takes one comparison to find the target element, so the best case complexity is O(1)

- Worst Case → O(log n)

The worst case will be when the element is present in the first position/last position.



In the end only one element will be there.

$$\frac{N}{2^k} = 1$$

$$N = 2^k$$

$$\log(N) = \log(2^k)$$

$$\log(N) = k \log 2$$

$$k = \frac{\log N}{\log 2}$$

$k = \log_2 N$ ← total no. of comparisons in worst case

$\log_2 N$ → size of array

Order-Agnostic Binary Search

In this modified version of Binary search comes with one more condition checking. The intution behind this algorithm is what if the order of sorted array is not given.

Start > End ⟶ Descending order

Start < End ⟶ Ascending order

SORTED ARRAY ——→ APPLY BINARY SEARCH

Example -

- BinarySearch

```
public class BinarySearch {
    public static void main (String [] args) {
        int [] arr = {-18, -12, -4, 0, 2, 3, 4, 15, 16, 18, 22,
                      45, 89};
        int target = 22;
        int ans = binarySearch (arr, target);
        System.out.println (ans);
    }
}
static int binarySearch (int [] arr, int target) {
        int start = 0;
        int end = arr.length - 1;
        while (start <= end) {
// Find middle element
// int mid = start + end / 2; might exceed range of int
        int mid = start + (end - start) / 2;
        if (target > arr [mid]) {
            start = mid + 1;
        } else if (target < arr [mid]) {
            end = mid - 1;
        } else {
            return mid; //ans found
        }
    }
    return -1;
}
```

- O/P -

10

- Working - target = 22

	0	1	2	3	4	5	6	7	8	9	10	11	12
Array	-18	-12	-4	0	2	3	4	15	16	18	22	45	89

start - 0

End - 12

mid = 6 ⟹ 4

target (22) > mid term (4)

start = mid + 1

start = 7

End = 12

mid = 9 ⟹ 18

target (22) > mid (18)

start = mid + 1

Start = 10

End = 12

mid = 11 ⟹ 45

target (22) < mid (45)

start = mid - 1

Start = 10

End = 11

mid = 10 ⟹ 22

target (22) == mid (22)

return mid

- Order-Agnostic Binary Search

```
public class OrderAgnosticBS {
    public static void main(String[] args) {
        int[] arr = {99, 80, 75, 22, 10, 10, 5, 2, -34};
        int target = 22;
        int ans = orderAgnosticBS(arr, target);
        System.out.println(ans);
    }

    static int orderAgnosticBS(arr, length) {
        int start = 0;
        int end = arr.length - 1;
        boolean isAsc = arr[start] < arr[end];
        while (start <= end) {
            int mid = start + (end - start) / 2;
            if (arr[mid] == target)
                return mid;
            if (isAsc) {
                if (target < arr[mid]) {
                    end = mid - 1;
                } else {
                    start = mid + 1;
                }
            } else {
                if (target < arr[mid]) {
                    start = mid + 1;
                } else {
                    end = mid - 1;
                }
            }
        }
        return -1;
    }
}
```

- Working.

	0	1	2	3	4	5	6	7	8
Array	99	80	75	22	11	10	5	2	-3

Start - 0

End - 8

isAsc - arr[start] < arr[end] → 8th index < 8th index

(99) < (-3)

False

∴ Array is not in Ascending order.

Executing else block

first Mid = 4th index value = 11

target (22) > mid value (11)

end = mid - 1;

second End = 3

Mid = 1st index value = 80

target (22) < mid value (80)

start = mid + 1;

Third Start = 1

Mid = 2nd index value = 75

target (22) < mid value (75)

start = mid + 1;

fourth Start = 2 + 1 = 3

Mid = 3rd index value = 22

target == value

return mid;

- Ceiling = smallest element in array greater or equal to target element

```
public class Main {
    public static void main (String[] args) {
        int[] arr = {2, 3, 5, 9, 14, 16, 18};
        int target = 15;
        int ans = ceiling (arr, target);
        System.out.println (ans);
    }
    Static int ceiling (int[] arr, int target){
    // If target no is greater than greatest no in array
    if (target > arr[arr.length - 1]){
        return -1;
    }
    while (start <= end) {
        int mid = start + (end-start)/2;
        if (target < arr[mid]){
        end = mid -1;
        } else if (target > arr[mid]) {
            start = mid +1;
        } else {
            return mid;
        }
    }
    return start;
    }
}
```

O/P 5

- Working-

Start → 0 1 2 3 4 5 6 ← End

Array	2	3	5	9	14	16	18

target = 15

first — Loop start: (start <= end)

Start = 0 And End = 6

mid = 3 Array[mid] = 9

target (15) > Array[mid] (9)

start = mid + 1

Second — Start = 4 And End = 6

mid = 5 Start → 4 5 6 ← End

Array[mid] = 16

		14	16	18

target (15) < Array[mid] (16)

end = mid - 1

Third. — Start = 4 And End = 4

mid = 4 target (15) > Array[mid] (14)

start = mid + 1

Start → 4 ← End

14

– Start = 5 And End = 4

Start → 4 5 ← End

14	16

Loop End.

∵ Start > End

Loop Terminated ∴ return start

Explanation The Binary Search is used to find the target value but when it is not found it returns start. Because start will be the smallest element greater than target as it moves forward when target element not found so we get greater value than target and as the array is sorted the number after it will be more greater than it

★ Same can be done to find floor of a number. We just have to return 'end' value