15/8/21

Binary Search

Algorithm:

sorted array

arr = [2, 4, 6, 9, 11, 12, 14, 20, 36, 48]

ascending order

target (36)

1. Find the middle element
2. Check :
   if target > middle ⇒ search in right
   else ⇒ search in left
3. if target == middle ⇒ we found element.

Example:

In the above array,

target = 36

arr = [2(0), 4(1), 6(2), 9(3), 11(4), 12(5), 14(6), 20(7), 36(8), 48(9)]

1st → find middle element

mid = $\frac{start + end}{2}$ = $\frac{0+9}{2}$ = 4 [the element at index 4 is the middle element]

2nd → let's check:

Is target > middle ⇒ 36 > 11 ⇒ yes! ⇒ Check in right side

Now, arr = [2, 4, 6, 9, 11, 12(5), 14(6), 20(7), 36(8), 48(9)]

mid = $\frac{5+9}{2}$ = 7 [the element at index 7 is the middle element]

Let's check:

Is target > middle ⇒ 36 > 20 ⇒ yes! ⇒ check in right side

i.e., arr = [2, 4, 6, 9, 11, 12, 14, 20, 36(8), 48(9)]

mid = $\frac{8+9}{2}$ = 8 [the element at index 8 is the middle element]

Here, target == middle ⇒ 36 == 36 ⇒ we found Element at index 8

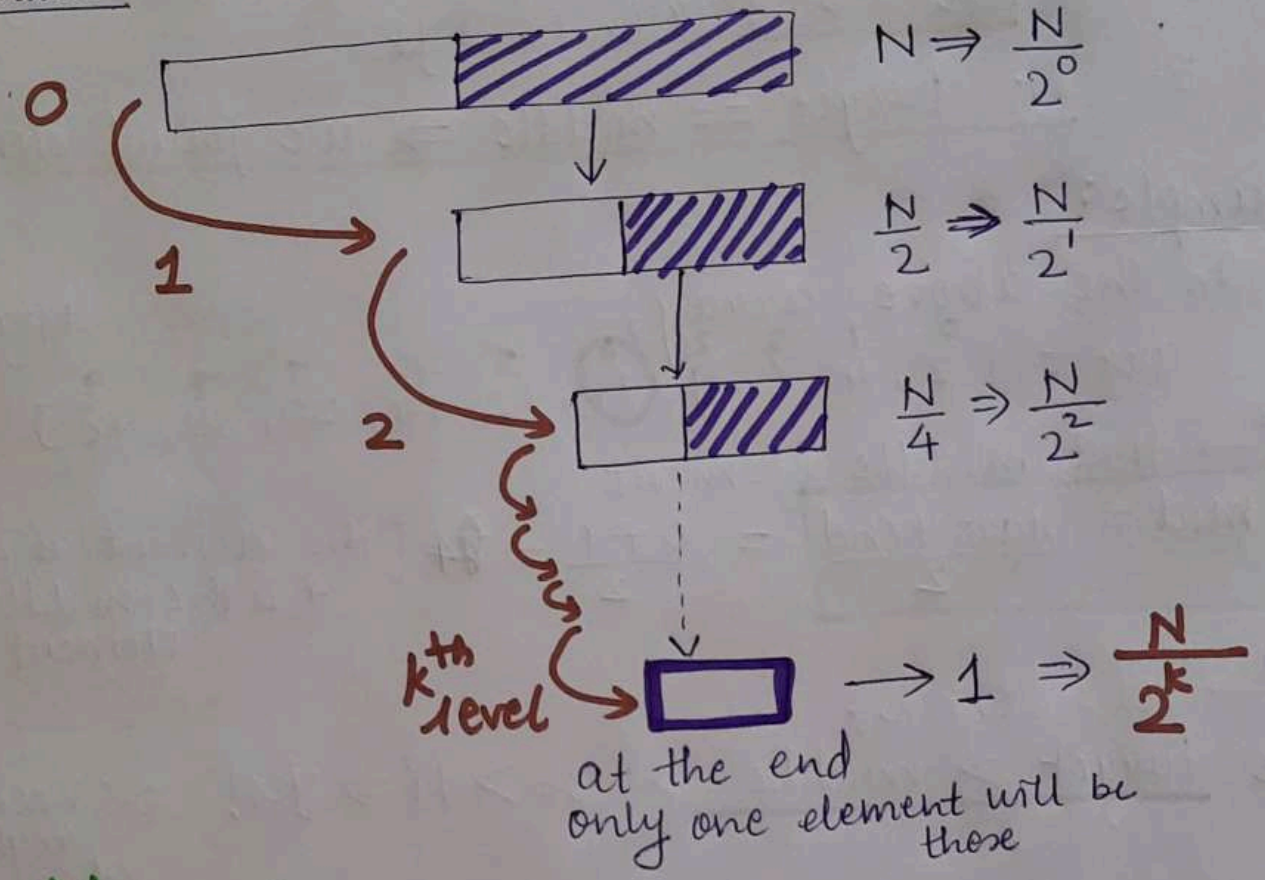
Here, the space in which we are searching is getting divided into two spaces.

⇒ Time Complexity:

Best case: $O(1)$

Worst Case: $O(\log n)$

Explanation: Find the maximum number of comparisons



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

$$N = 2^k$$

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

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

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

$$\boxed{k = \log_2 N}$$

k → total number of comparison in worst case

N → size of array

⇒ Order agnostic Binary Search

Let's say if we don't know that the array is sorted in ascending or descending order.

arr = [90, 75, 18, 12, 6, 4, 3, 1]

target = 75

Here, target > middle ⇒ search in left

~~for descending order~~

Here, start > end] → descending order.

90 > 1

when start < end → Ascending order