*** Binary Search ***

* **Binary Search:**
  - ⇒ It is the most Important algorithm.
  - ⇒ It is used for sorted array (either in ascending or descending order).

* **Algorithm (when elements are in Ascending Order):**

Step 1. Find the Middle element ($mid = \frac{start + end}{2}$)

Step 2. Check
if target > middle element ⇒ search in Right side

$start = mid + 1$

else ⇒ Search in Left

$end = mid - 1$

Step 3. if target == middle ⇒ Element found.

* **EXAMPLE:**

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

Target = 36

mid

↳ ascending order

S-1: Find middle element ⟶

$mid = \frac{start + end}{2} = \frac{0+9}{2} = 4$ ⇒ element at index 4 is mid element.

S-2: Check ⟶

Here, Target > mid (36 > 11) ⇒ check in Right side

Now,

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

Start = mid+1 = 4+1 = 5

mid

S-3: mid = $\frac{5+9}{2}$ = 7 ⇒ element at index 7 is mid element

S→4: Check:

Here, Target > mid (36 > 20) ⇒ check in Right side

Now,

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

Start = mid+1 = 7+1 = 8

S-5: mid = $\frac{8+9}{2}$ = 8 ⇒ element at index 8 is mid element

S-6: Check:

Here, Target == mid (36 = 36) ⇒ element found at index 8.

* Algorithm (when elements are in Descending Order):

Step 1. Find the Middle element.

Step 2. Check

if target > mid ⇒ Search in Left side

end = mid - 1

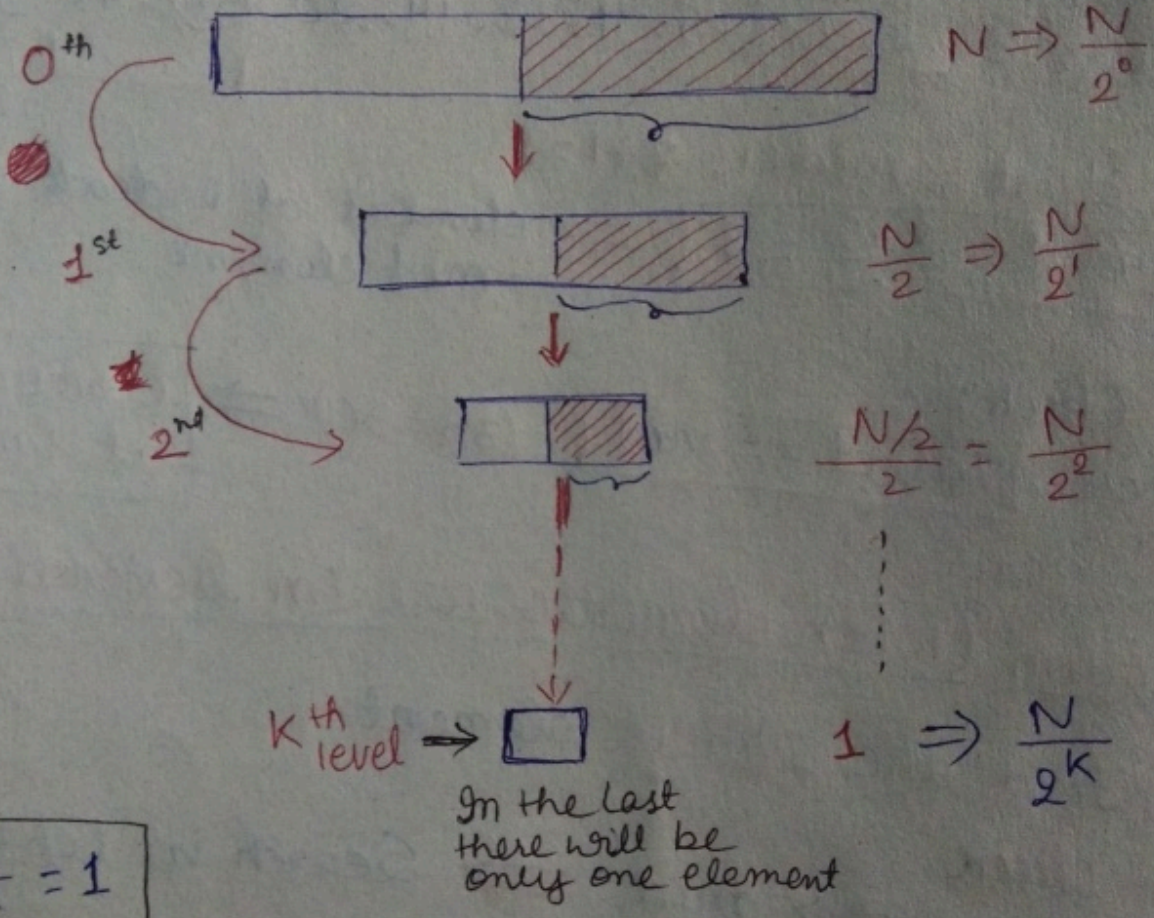
else ⇒ search in Right side

Start = mid + 1

Step 3. if target == middle ⇒ Element found.

Time Complexity

* **Best Case** : O(1) // constant
  [when first middle element = target value]
* **Worst Case** : O(log n) // n → size of array
* Find the Maximum number of comparisons in the worst case.....



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

$$\Rightarrow N = 2^K$$

$$\Rightarrow \log N = \log(2^K) \quad \text{// Taking log}$$

$$\Rightarrow \log N = K \log 2$$

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

$$\boxed{K = \log_2 N} \leftarrow \text{size of array}$$

↑ Total no. of comparisons in worst case

* Why Binary Search ?

(Q.) Search an element in an array of size 1,000,000 (1 million).

⇒ In worst case, Linear Search will make 1 million comparisons.

⇒ In worst case for same array, Binary Search will make log (1,000,000) i.e, 20 comparisons only

* Order-Agnostic Binary Search :

We know, for Binary search we need sorted array But, Let's say, if we don't know that the array is sorted in ascending or descending order.

⇒ If start > end ⇒ Descending Order

If start < end ⇒ Ascending Order