# Binary Search

Page No.
Date :

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 the element.

Example :- In the above array,

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

target = 36

1st → find middle element.

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

2nd → Let's check :

Is target > middle => 36 > 11 => yes => check in right

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

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

Let's check :

Is target > middle => 36 > 20 => yes => check in right side.

Page No.
Date :

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

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

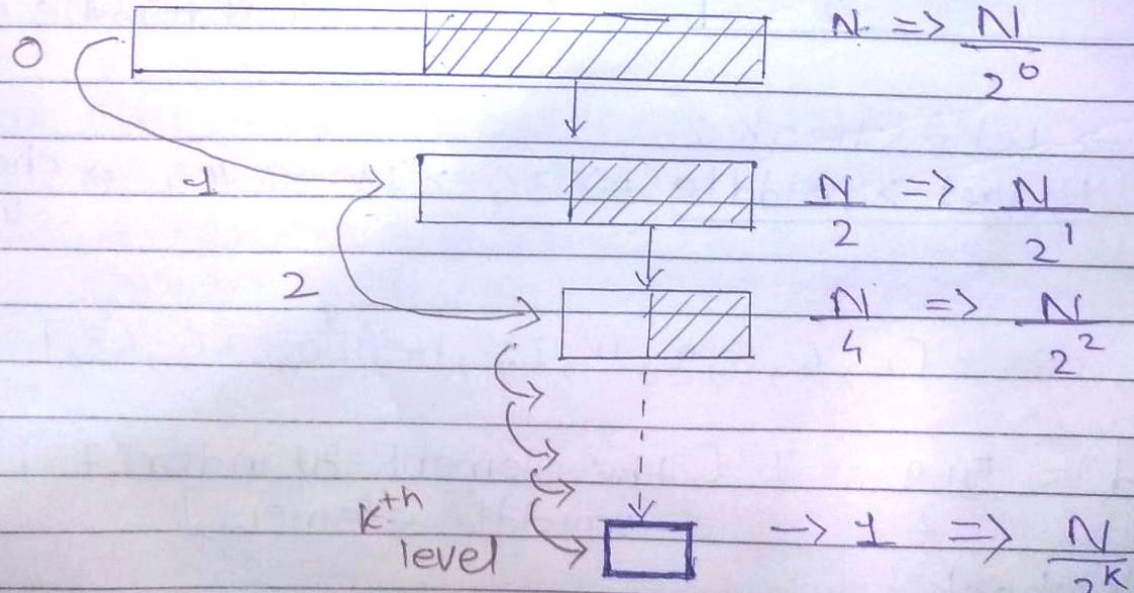
here,
target == middle ⇒ 36 == 36 ⇒ we found the element at index 8

* 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 maximum Number of comparison.




at the end only one element will be there.

Page No.
Date :

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

$$N = 2^k$$

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

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

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

$$k = \log_2 N$$

total number of comparison in worst case.

size of array

Example :-

```
Public Static Void Main () {
    int [] arr = { 2, 4, 6, 8, 10, 12 };
    int target = 10;
    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) {
    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 -1;
}
```

## * Order agnostic Binary Search

Let's say if we don't ~~kmo~~ 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

target < middle = —11— right

Here,

start > end. → Descending order

when start < end → Ascending order.