Manan Khandelwal

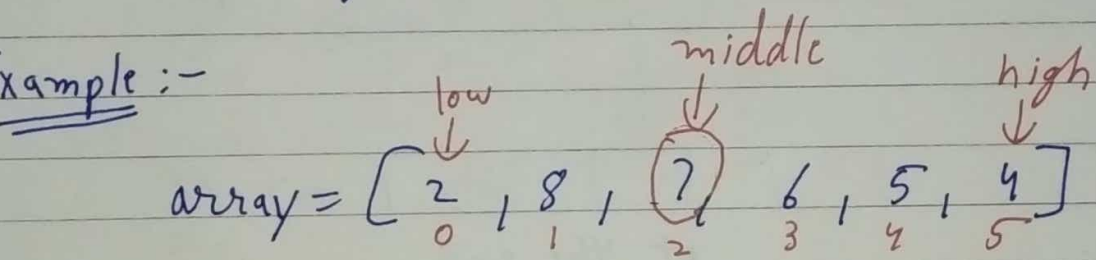
Date .....

Binary Search

~~Let~~ Let x be the target that we need to find

1. Compare x with the middle element of our array
2. If x == middle element
   return middle index
   else if x > middle element
   it means x lies in right half subarray (search in right half)
   else if x < middle element
   it means x lies in left half subarray so, search in left half

Example :-

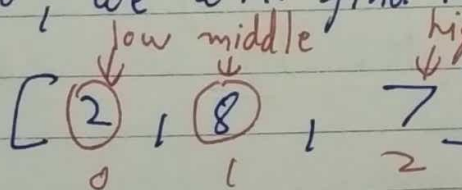


target = 8

$$middle = \frac{low + high}{2} = \frac{0+5}{2} = 2$$

The element at index 2 is our middle element

our target is 8 and middle element is 7
that means target < middle
So, we will find in left half



$$middle = \frac{low + high}{2} = \frac{0+2}{2} = 1$$

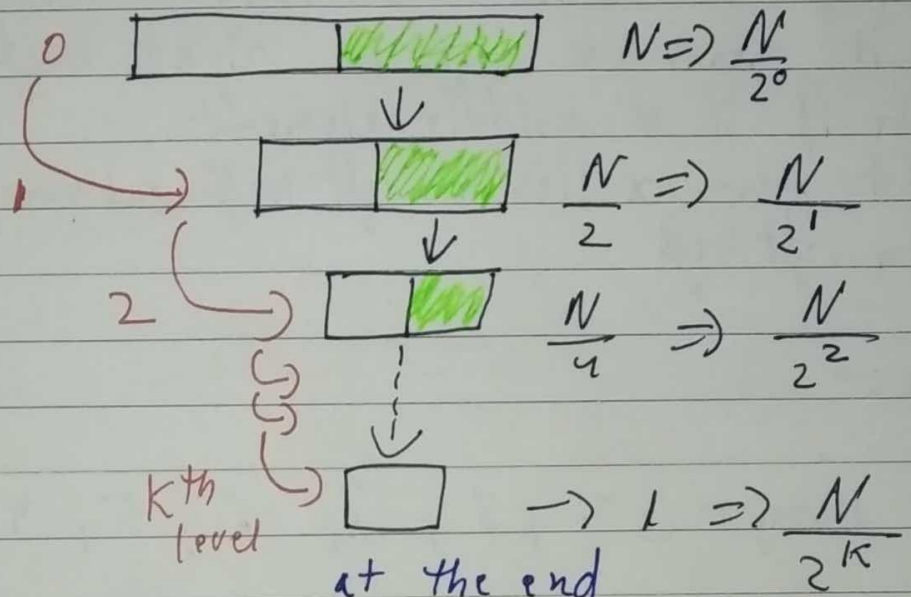
Now, target == middle element
we found element at index 1.

Date .....

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

⇒ Time Complexity:

Best case : $O(1)$
worst case : $O(\log n)$



at the end only one element we will there

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

$$N = 2^K \Rightarrow \log(N) = \log(2^K)$$
$$\Rightarrow \log(N) = K \log(2)$$
$$K = \frac{\log(N)}{\log(2)}$$

$$K = \log_2(N)$$

total no. of comparisons

size of array

Date .....

// Iterative

```
int Binary( int A[], int n, int data)
{
    int low = 0;
    int high = n-1;
    while (low <= high)
    {
        mid = low +(high -low)/2; // To avoid
        if ( A[mid] == data)          // overflow
            return mid;
        else if ( A[mid] < data)
            low = mid +1;
        else high = mid -1;
    }
    return -1;
}
```

// Recursive

```
int Binary( int A[], int low, int high, int data)
{
    int mid = low + (high -low)/2;
    if ( A[mid] == data)
        return mid;
    else if ( A[mid] < data)
        return Binary(A, mid+1, high, data);
    else
        return Binary(A, low, mid-1, data);
    return -1;
}
```