① Binary Search :- It is an algorithm that is generally used to find the position of a specific element within a sorted array. Here, the array is divided into left and right half on the basis of middle element and then depending on the value of element that is searched, we need to search in left or right half of the array.

eg:- 11, 21, 31, 40, 58, 68, 79, 90

0 1 2 3 4 5 6 7

n = 8 , target = 20

left = mid + 1 = 4

right = 7

$$mid = left + \left(\frac{right - left}{2}\right)$$

= 3

mid = 7

right = 6

$$= \frac{2\,left + right - left}{2}$$

$$= \frac{left + right}{2}$$

To avoid Integer overflow

- Each time you will compare target element with middle element; if matched then good otherwise each time you divide the array by half.

step 0 :- Size of array n

step 1 :- " " $\frac{n}{2}$

step 2 :- " " $\frac{n}{4}$

step 3 :- " " $\frac{n}{8}$

⋮

step k :- " " $\frac{n}{2^k}$ } (1 element)

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

$$\Rightarrow 2^k = n$$

$$\Rightarrow \boxed{k = \log_2 n}$$

Max^m no. of steps is $\log_2 n$.

Whenever in an Interview, you have provided the solution of O(n) but your interviewer still asks you for a better T.C, then in that situation THINK OF Binary Search as T.C of Binary Search is $\log_2 n < n$.

- Binary Search is Implemented via 2 ways :-

1. Iterative Ways

2. Recursive Ways.

⊙ Tricks to keep in Mind while using Recursion :-

1. Identify the parameters which you will need to use to solve the problem.

2. Think of Base Condition.

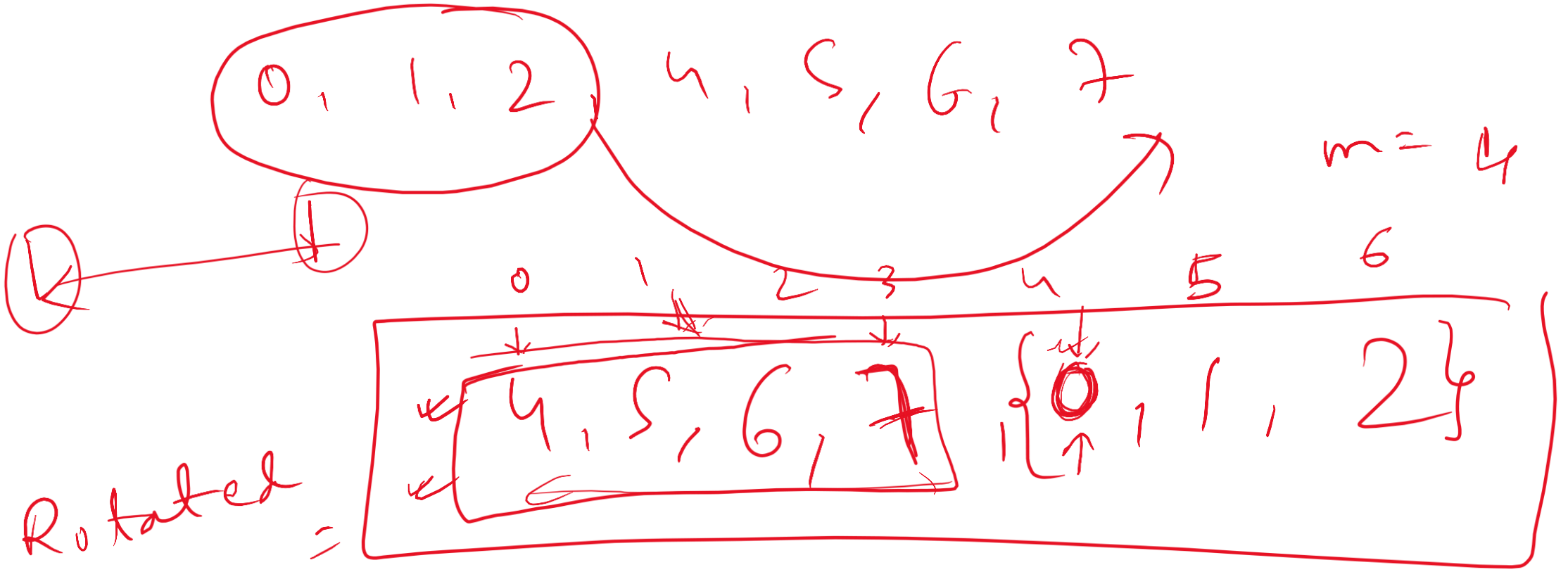
3. Make Recursive Fn call by changing parameter values at recursion.

4. Return appropriate value back to it's parent f^n call depending on the problem.

```
int func( int arr[], int s, int e, int t )
{   if ( s > e) return -1;
    int mid = ( s + (e - s) / 2 );
    if ( arr[mid] == t)
        return mid;
```

index of target element

```
if( target > arr[mid] )
{
    int res = (func(arr, mid+1, e, t));
    return res;
}
else
{
    int res = func(arr, s, mid-1, t);
    return res;
}
```



mid = 3

target = 0

s = 4

e = 4

4 → if(target == a(mid)) → 7

if(a[s] <= a(mid])

{ if (target < a(mid] && target >= a[s])

{ e = mid - 1; } s = mid + 1;

① 0 1 2 3 4 5 6 7

6, 7, 0, 1, 2, 3, 4, 5 , target = 0

m = 3 , a(m) = 7

s = 0
e = 2 mid = 1

a(s) = 6 , a(m) = 1

if(a[mid] <= a[end]) { s = 2 (mid = 2)
e = 2

{ if(target > a[mid] && target <= a[end])

s = mid + 1; mid = 2+2 / 2 = 4/2 = 2

else

e = mid - 1;

}

0 1 2 3 4 5 6 7 8

① 5, 8, 13, 18, 19, 25, 25, 25, 30

target = 25

s = 5, e = 8, m = 6

a(m) = 25

res = m = 5

while (s <= e)

e = m - 1 = 4

s = 5

m = 5