Binary Search

Applicable only on sorted arrays either ascending or descending order

Sorted → (Monotonic, Monotonous)

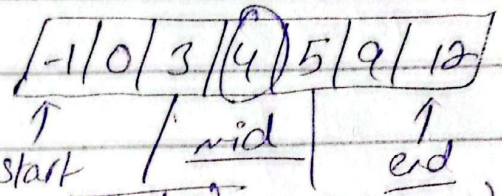
Linear Search gives $O(n)$ T.C

Binary Search gives $O(\log n)$ T.C

Steps

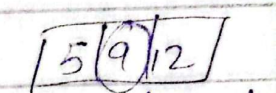
① Find mid

$$mid = \left\lfloor \frac{start + end}{2} \right\rfloor$$

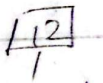


(i) tar > arr[mid] → 2nd Half (st = mid+1) (mid-1) (mid+1) (n-1)

(ii) tar < arr[mid] → 1st Half (end = mid-1)



(iii) tar = arr[mid] → ans mid



1st Half → (st to mid-1)

2nd Half → (mid+1 to end)

dry Run :-

$mid = \frac{0+6}{2} = 3$

st = 4 end = 6

$mid = \frac{4+6}{2} = 5$

st = 6, end = 6

$\frac{6+6}{2} = 6$

Code

st=0, end=n-1

```
while(st<=end){
    mid=(st+end)/2
    if(tar>arr[mid]){
        st=mid+1
    }
    else if(tar<arr[mid]){
        end=mid-1
    }
    else{
        return mid;
    }
}
return -1;
```

$\longrightarrow \left[st + \frac{(end-st)}{2}\right]$

Optimizing the mid formula

$mid = \frac{st+end}{2}$ INT_MAXES [Two Int_max values gives bigger version of int_max that makes the int mid overflow

So, the solution is ↓

$mid = \left[st + \frac{(end-st)}{2}\right] \rightarrow st + \frac{(end-st)}{2}$

$\frac{2st+end-st}{2}$ $\left(\frac{st+end}{2}\right)$

Binary Search with Recursion

```
int BS(int arr, tar, st, end){
    if(st<=end){
        mid=st+(end-st)/2
        if(tar>arr[mid]){
            return BS(arr, tar, mid+1, end)
        }
        else if(tar<arr[mid]){
            return BS(arr, tar, st, mid-1)
        }
        else
            return mid
    }
    return -1;
```

Better = ?

Recursion

T.C = O(log n)

S.C = O(log n)

Iteration

T.C = O(log n)

S.C = O(1)

↓

Optimized in terms of space