Binary Search

→ Binary Search.
→ Order agonostic Binary Search.
→ 1st and last Occerences of an Element.
→ Count of element in Sorted Array.
→ # of times → Array is rotated
→ Find an element in Rotated Sorted Array
→ Searching in nearly sorted Array
→ Floor | ceil of an Element
→ Next Letter
→ Index of Last 1 in Sorted Array
→ Find the position of an element in an sorted Array.
→ Min diff element in a Sorted Array
→ Bitonic Array max Element.
→ Search in a Bitonic Array.
→ search in Row wise + Col wise Sorted matrix
→ Find Element in sorted Array that appears only once.
→ Allocate Min ~~#~~ # of Pages.

→ If Question have sorted keyword then there will be more chance to apply Binary search to reduce time complexity

Binary Search.

	0	1	2	3	4	5	6	7	8	9
arr[]:	1	2	3	4	5	6	7	8	9	10
	Start				Mid					end

Key = Search 2

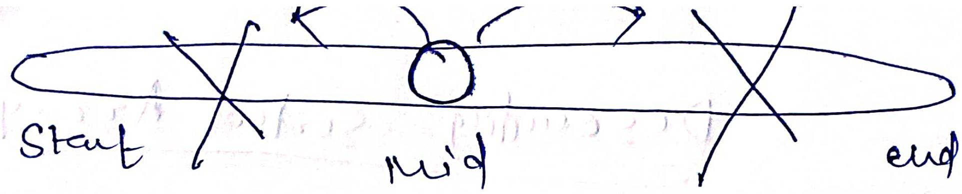
←

Return index of 2 (i.e 1)

- One method is of Linear search in O(n)
- But here we have sorted array so we can apply Binary search.

- int Binary_search(arr[], key)

```
{   int start = 0;
    int end = arr.size();
// if key element is less than mid then keep
    end to mid - 1.
// If key element is greater than mid so
    left-hand-side is useless so move
    start to mid + 1.
```



$\rightarrow n$

$n/2$

$n/4$

$O(\log_2 n)$

Code.

```
while (start <= end)
{
    int mid = (start + end) / 2;
    int mid = start + (end - start) / 2;
                          ↳ optimized
                            ↳ int overflow
    if (key == arr[mid])
        return mid.
    else if (key < arr[mid])
        end = mid-1;
    else
        start = mid + 1;
}
```

↳ start + (end - start) / 2;

// to avoide integer overflow

Descending Sorted Array.

	0	1	2	3	4	5	6	7	8	9
arr[ ]:	20	17	15	14	13	12	10	9	8	4
	↑ start				Mid					↑ end

key : 4.

- Now If key is less then mid then set mid = start + 1
- and If key is greater then mid then set mid = end - 1.

Code

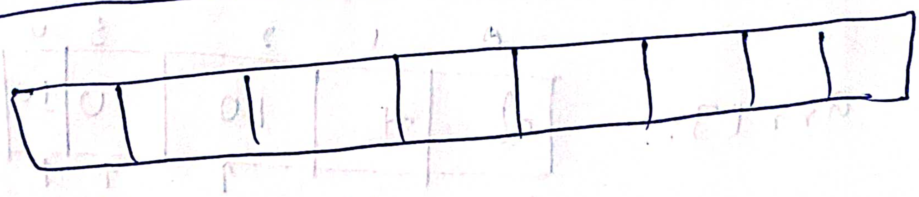
```
If (key < arr[mid])
    Start = mid + 1
else
    end = mid - 1;
```

always keep in mid as

$$mid = start + (end - start) / 2$$

for better passes of test cases

Order Agnostic Search:

arr[]: 

key : 5

Sorted array, we don't know either of is ascending or descending

$$\begin{cases} arr[0] < arr[1] \Rightarrow \text{ascending} \\ arr[0] > arr[1] \Rightarrow \text{descending} \end{cases}$$

- We will have a sorted array and an key element, search an element in this but we don't know, either of is ascending and descending.

So check:

Compare any two element:
- $arr[0] < arr[1] \Rightarrow$ ascending.
- $arr[0] > arr[1] \Rightarrow$ descending.

- and check for size as well, if size is one the return the same key element.