Binary Search -> Deepak Manglani

-> When the array/matrix is sorted, always think how you can apply Binary Search.

-> There is also one High level concept which is "Binary Search on Answer" which is also used when array is not sorted

-> Concept of Binary Search :-

Given an Array :-

I/P : 1 2 3 4 5 6 7 8 9 10

Ele : 2

O/P : 1 (Index of 2 in array)

First, we can do linear search, picking element by element in order.

Time Complexity :- O(N)

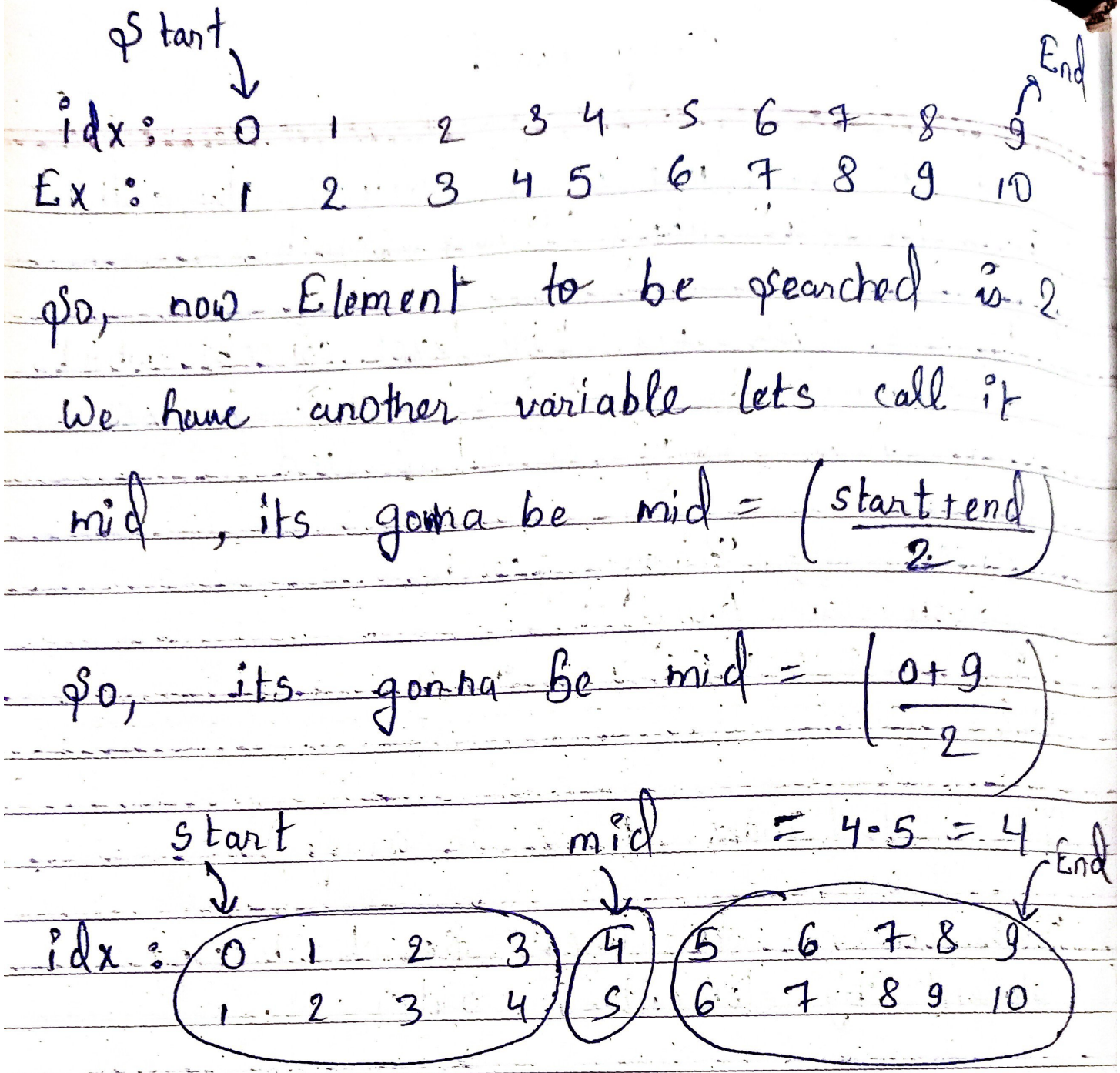
idx : 0 1 2 3 4 5 6 7 8 9

Ex :- 1 2 3 4 5 6 7 8 9 10

(Linear Search) (ans) ele : 4

Since, the array is sorted we can think of an optimised approach.

Lets, suppose we have two variables start and end pointing to the 0th and size-1 index of the array.



So, now Element to be searched is 2

We have another variable lets call it mid, its gonna be mid = $\left(\frac{start+end}{2}\right)$

So, its gonna be mid = $\left(\frac{0+9}{2}\right)$ = 4.5 = 4

Now, the search space / array is divided into three parts.

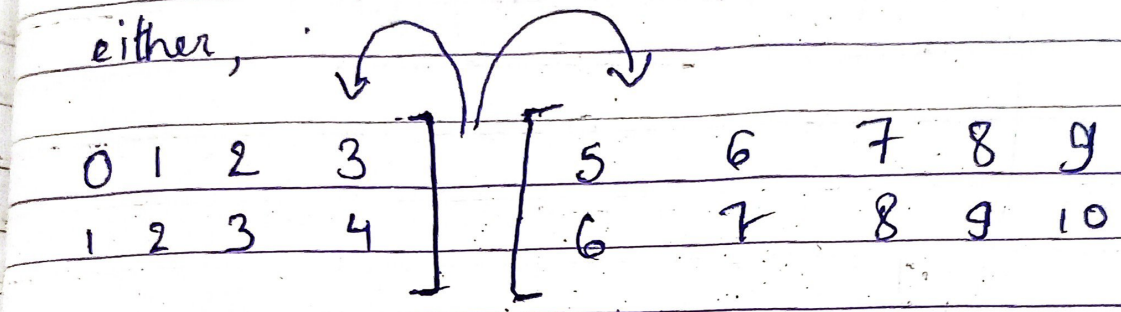
1) One which is greater than mid
2) one which is smaller than mid
3) Mid itself

So, now we compare

is ann[mid] == ele (2) ? 5 == 2
(no)

So, now we have two options

either,



Also, take note that array is sorted

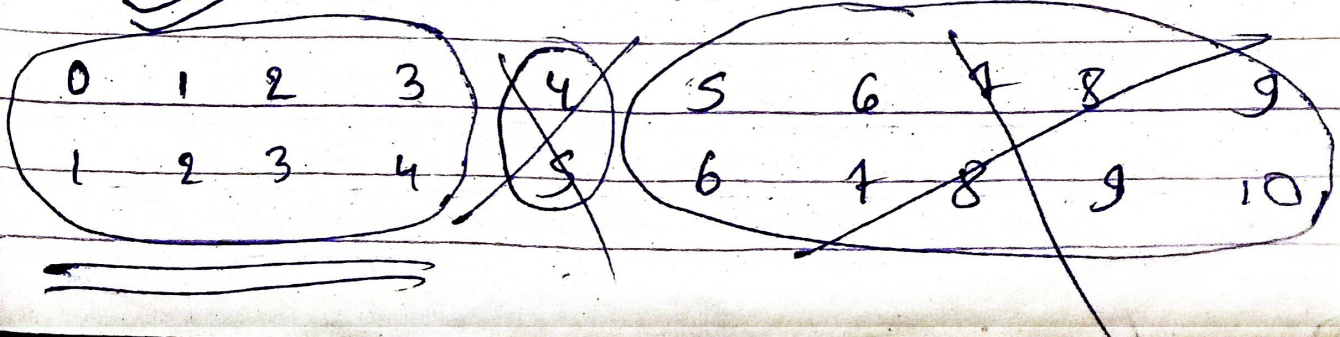
So, if ann[mid] > ele

the elements

between, (mid+1) to end

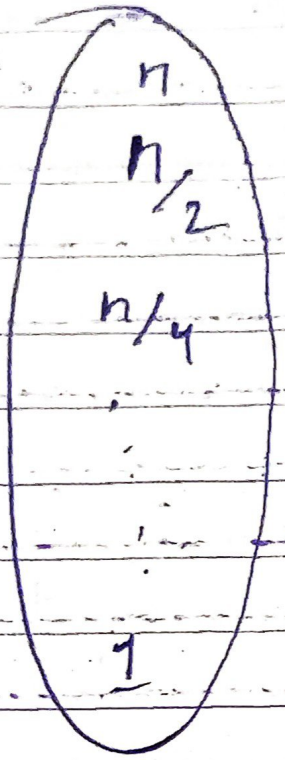
are useless to us since, they all will be greater than ele.

So, our search space would be



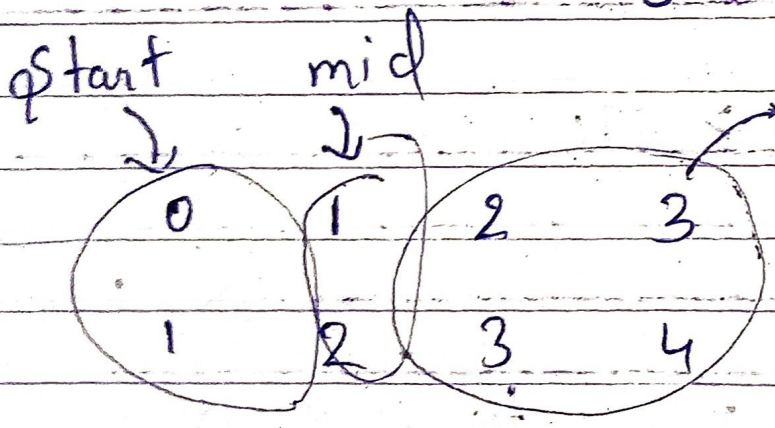
So, here everytime we are going to make our search space/array half.

(In Worst Case)



$\rightarrow \log_2(n)$

So, time Complexity will be $O(\log_2(n))$



Since, arr[mid] == ele (2 == 2)

we will return our answer.

Also, take note that binary search runs until start ≤ end

start → end

0	1	2	3
1	2	3	4

These are just like first and last index of array so, if end < start there is no element

start → ← end

0	1	2	3	4	5
5	6	8	12	14	15

Also, $mid = \frac{start + end}{2}$

$mid = start + \frac{(end - start)}{2}$

This is better than above because lets say maximum value of int is 5

So,

Lets say start = end = 5,

mid $\frac{5+5}{2}$ = 10 (overflow) = (-5)

(10)

1 2 3 4 5 -1 2 3 4 5

Instead

$$mid = 5 + \frac{(5-5)}{2} = 5 + \frac{0}{2}$$

This would not cause overflow.

Code :

```
int binarySearch (arr, int ele) {
  int start = 0
  int end = size - 1

  while (start <= end) {
    int mid = start + (end - start)/2
    if (arr[mid] == ele) return mid
```

```
else if (arr[mid] > ek)   end = mid - 1
else   start = mid + 1
}
return -1
}
```

→ Concept of Descending sorted Array :-

idx :- 0	1	2	3	4	5	6
20	18	16	13	11	10	8

So, if I need to search element 8.

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

Since, 13 > 8.

start = mid + 1 (not end = mid - 1)

otherwise,

end = mid - 1,

Code Variation :

```
int start = 0
int end = size - 1
while (start <= end) {
int mid = start + (end - start / 2)
if (arr[mid] == ele) return mid
else if (arr[mid] > ele) start = mid + 1
else end = mid - 1
}
return -1;
```

Order Not Known / Order Agnostic Seach

In this problem we actually don't known whether the array is sorted in ascending order or descending order

Array will always be sorted.

So, we need to check

If the array is of size 1, we simply return the result (based on the problem)

if size of array ≥ 2

We checks :

if arr[0] ≤ arr[size-1]

sorted in ascending order

else :

sorted in descending order

First and Last Occurrence of an Element

In this problem, lets say we have an array.

	0	1	2	3	4	5	6
Ex :- I/P :	2	8	10	10	10	14	20

Ele : -10

O/P :- First Occurence will be at index 2

O/P : Last Occurrence will be at index 4.

So, there will be just a slight variation to the binary search of an element

Whenever we find the element, we just don't stop there and move our pointer according to our problem i.e., first / last occurrence.

Here,

0	1	2	3	4	5	6
2	8	10	10	10	14	20

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

Since, We need to find first occurrence

We do, end = mid - 1

Otherwise for last occurrence,

start = mid + 1

Code :-

```
int start = 0
int end = size - 1
int res = -1
while start <= end {
    int mid = (start + ((end - start) / 2))
    if (arr[mid] == ele) {
        res = mid
    }   end = mid - 1 (first) // start = mid + 1 //(Last)
else if (arr[mid] > ele) {
        end = mid - 1
}
```

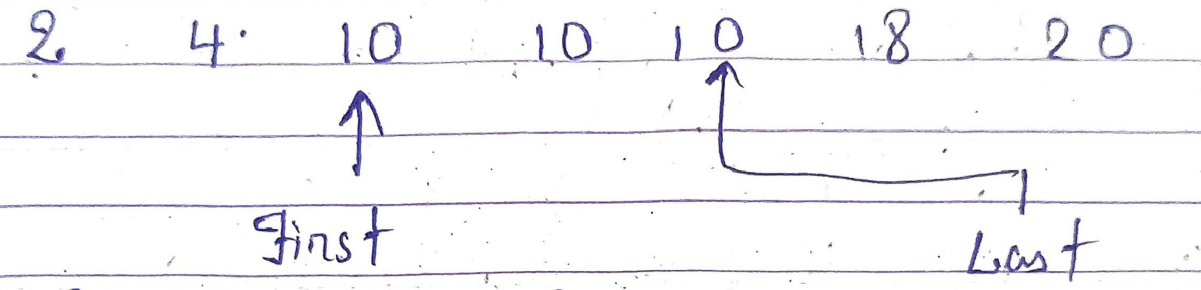
else:

start = mid + 1

Count of an element in a Sorted Array

Its Basically,

given an array.



$$\text{Count} = \left(\begin{matrix} \text{Last} \\ \text{Occurrence} \end{matrix} - \begin{matrix} \text{First} \\ \text{Occurrence} \end{matrix} + 1 \right)$$

Keep in mind, the edge case when the element is not present

i.e., $-1 - (-1) + 1 = +1$

It should be 0.

Number of times Array is Rotated

Really very important problem.

Problem Statement

An array of distinct numbers sorted in increasing order

The array has been rotated K number of times. Given such an array, find the value of K

```
        0   1   2   3   4   5
I/P :- 15, 18, 2, 3, 6, 12
```

O/P :- 2

Explanation :- {2, 3, 6, 12, 15, 18} initial array

So, array after two cyclic rotation will be

{15, 18, 2, 3, 6, 12}

```
          0  1  2  3  4
I/P :- {7, 9, 11, 12, 5}
```

O/P :- 4

Observation :-

The number of time the array is rotated is equal to the index of the minimum element

Number of Rotation	= =	Index of Minimum Element in the Array

We can find this index in linear time with ease.

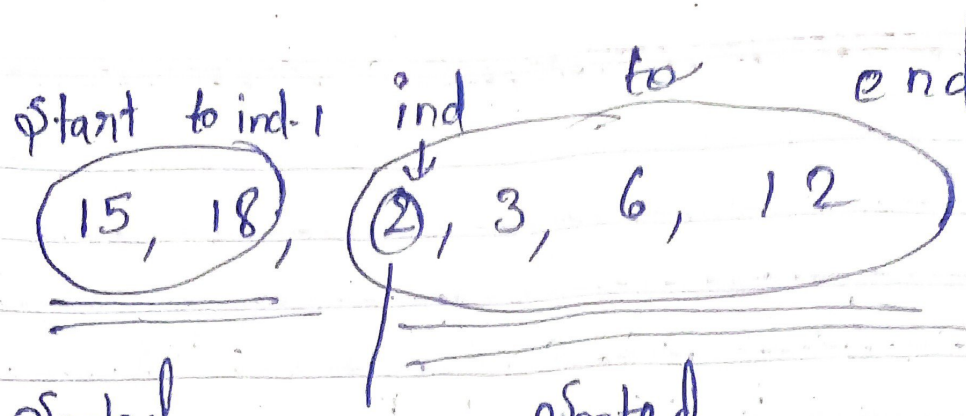
but, how to do this problem with the help of sorted array using binary search in $O(\log(n))$ time

more
⇒ Lets gather some observation :-

1) wherever the minimum element is given in the given array, it will be smaller than its adjacent left and right neighbours.

2) Also keep in mind that the position where the minimum element is present to the end is one sorted array

and, from start to minele ind − 1 is another sorted array.



So, we just need to find this minimum element index using binary search But, how can we do that?

How, Binary Search basically works?

We take two variables start and End and we find out the mid and decrease the search space every time by half depending on the condition to the problem

So, what would be the condition here?

Its basically going to be when,

arr[mid] ≤ arr[end]

↓

The elements between mid to end is sorted we go make end = mid - 1

otherwise if, (arr[start] ≤ arr[mid]) then, we do start = mid + 1

Basically, we can't find the minimum element in the search space where the array is sorted.

So, we left them off.

Since, in the sorted array the minimum element is not going to be present. the array are distinct & the min element lies between two greater elements.

Code :-

```
int start = 0
int end = size - 1

while(start ≤ end)

  int mid = start + ((end - start)/2)
  int next = (mid + 1) % n.
  int prev = (mid - 1 + n) % n.

  if (arr[mid] ≤ arr[prev] && arr[mid] ≤ arr[next])
      return mid.
  else if (arr[mid] ≤ arr[end]) end = mid - 1
```

else if (arr[mid] ≥ arr[start]) start = mid+1

Find an element in a rotated sorted array

We need to find any element present or not in the array of the (previous problem)

We just need to find the index of minimum element that we just did previously.

then, the element would be present either to start ind-1
or to ind ..... end.

and both the array are sorted.

So, again we do binary search & see if we can get the element or not,

Code:-

```
int ele_ind1 = bs(start, ind-1)
int ele_ind2 = bs(ind, end)

return max(ele_ind1, ele_ind2);
```

Searching in a Nearly Sorted Array.

Given an array which is almost sorted. that means element array at index i is either swapped with its adjacent elements or present at i.

The element arr[i] can only be swapped with arr[i+1] or arr[i-1].

I/P : {10, 3, 40, 20, 50, 80, 70}

Sorted Array : {3, 10, 20, 40, 50, 70, 80}

Nearly Sorted

Key : 40

O/P : 2 (index) (position of 40).

So, How can we do it?

Lets, compare it with binary search on sorted array.

Code:

```
int start = 0
int end = size-1
while(start ≤ end)
{
  int mid = (start + end / 2)

  if (arr[mid] == ele)
      return mid
  elseif (arr[mid] ≥ ele)
      end = mid - 1
  else
      start = mid + 1
}
```

Here, arr[mid] element can be present at either arr[mid-1], arr[mid] arr[mid+1] positions.

```
while (start ≤ end)
{
  int mid = start + (end-start)/2
  if (arr[mid] == ele)
      return mid
  else if (mid - 1 ≥ start &&
      arr[mid-1] == ele)
      return (mid-1)
  else if (mid+1 ≤ end &&
      arr[mid+1] == ele)
      return (mid+1)
  else if (arr[mid] ≥ ele)
      end = mid - 2
  else if (arr[mid] ≤ ele)
      start = mid + 2
```

So, the idea is to basically check middle of three elements arr[mid-1], arr[mid], arr[mid+1] & then shift either to end = mid-2 or start = mid+2