"श्री कृष्ण गोविंद हरे मुरारी, हे नाथ नारायण वासुदेवा"

Data Structure and Algorithm

By

Malay Tripathi

BINARY SEARCH.

Search in a Rotated Sorted Array ?

↳ Rotated Sorted Array.

Sorted Array = [1 2 3 4 5]

Rotated Sorted Array = [4 5 1 2 3] ↷ rotated.

arr[] = [1 2 3 4 5 6 7 8 9]

arr[] = [7 8 9 | 1 2 3 4 5 6]

This array is rotated at 7.

* our task is to find out the target. → means the element.

	0	1	2	3	4	5	6	7	8
arr[] =	7	8	9	1	2	3	4	5	6

So target = 1 and find the index, where 1 is present.

APPROACH-1: LINEAR SEARCH.

→ Search element by element, as soon as you find 1 return its index.
The worst case is if target = 6, than the complexity is O(n).

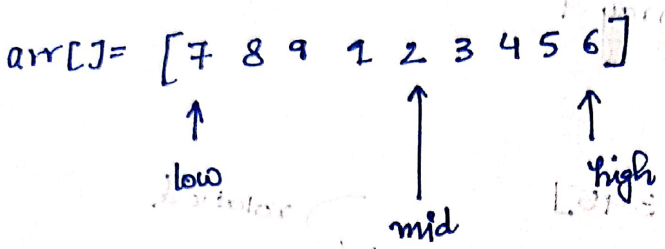
Whenever you find Search and Sorted → than "BINARY SEARCH".

"श्री कृष्ण गोविंद हरे मुरारी, हे नाथ नारायण वासुदेवा"

Data Structure and Algorithm

By

Malay Tripathi



Normally → let suppose target=1 → our mid=2 so as per the Binary Search rule, it is present at the left of 2 which is true.

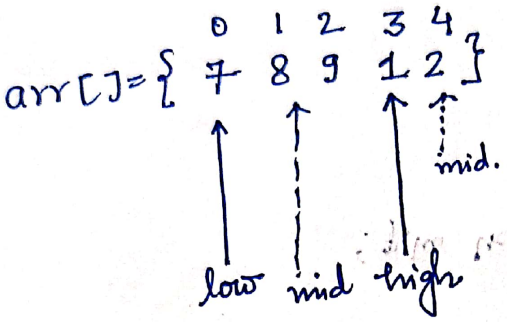
But → now, suppose target is 8, it has to be present at the right side of the 2, but in reality it is present at the ~~right~~ left side of the 2.

But, after Observation it is known that, the left side is not sorted but the right side is Sorted.

So key point here is that you have to find the Sorted half. It is the left half or right half.

* Left hand is NOT Sorted, So what you have to do, How To FIND, low = 7 (value) and mid = 2 (value) and low is not Smaller than mid, thus left half is not sorted.

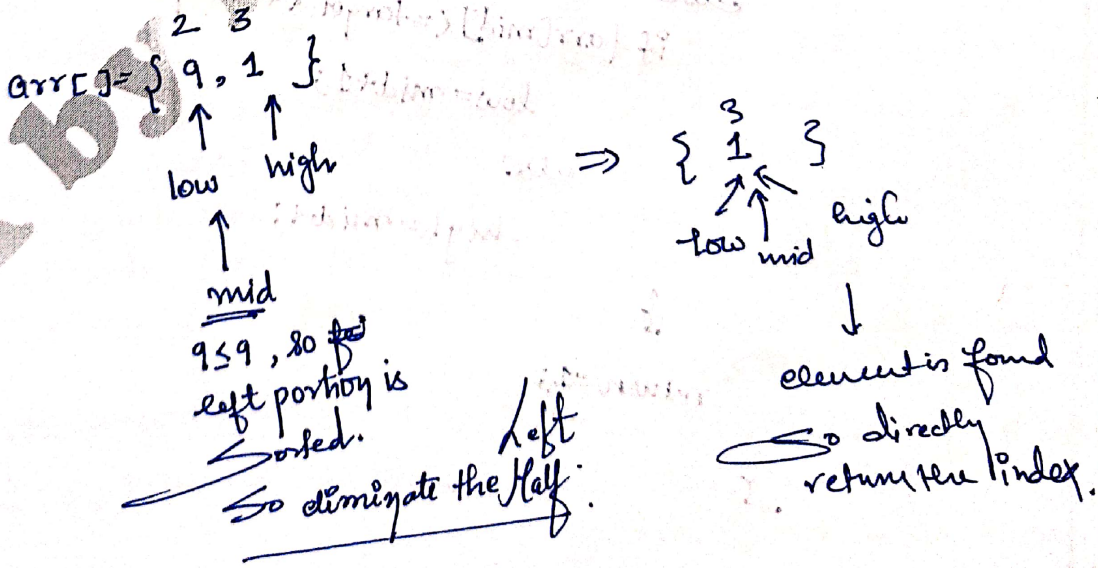
→ So, right is sorted, → only one thing we have to check that whether 1 is present b/w 2 and 6 or not.
1 is not present, so we eliminate the Right half.



Step: Identify the sorted half. LHS is sorted because element at low is smaller than element at mid.

Identify RHS as it is not following the property value of mid > value of high.

So eliminate left half.



"श्री कृष्ण गोविंद हरे मुरारी, हे नाथ नारायण वासुदेवा"

Data Structure and Algorithm

By

Malay Tripathi

Pseudo

```
f (arr, n, target)
  {
  low=0
  high=n-1
  while (low<= high)
     {
     mid= low+high
          2
     if (arr[mid]==target) return mid;

  // left half sorted
     if (arr[low] <= arr[mid])
        if (arr[low]<=target && target<=arr[mid])
              high= mid-1;
        else
              low= mid+1;

  // right sorted
     else
        if (arr[mid]<=target && target<=arr[high])
              low= mid+1;
        else
              high= mid-1;

     }

  return -1;

  }
```