**mid = low + (high - low) / 2;** → we do this instead of “(low+high)/2” is for Low Language like “C/C++” doesn’t get OVERFLOW in Integer if “low” & “high” is pretty big

**when to use low<high vs low<=high** →

<https://youtu.be/7jci-yQhGho?si=1ED2cRK6mw5gD20G>

<https://stackoverflow.com/questions/35613574/when-to-use-in-binary-search-condition>

**Binary Search Intuition**

* **First Occurrence** → **Left Side** ee **Pointer move** korate takhbo **First Occur** → So, Equal to “target” huile → “High” reh soraiya “Mid-1” ee niye ashbo
* **Last Occurrence** → **Right Side** ee **Pointer move** korate takhbo **Last Occur** → So, Equal to “target” huile → “Low” reh soraiya “Mid+1” ee niye ashbo
* One Element **How many times** in sorted array → Find First & Last Occur, then “**Last Occur Index - First Occur Index + 1**”
* **Rotated Sorted Array** index **where it was “rotated”** → we find the “**Minimum**” Element & “**Minimum” Element**’s “**index**” is where the Sorted Array was “Rotated”...

The **“index”** of the **“Minimum” element ==** where the **Array was “rotated”**

**Rotated Sorted Array**

if(nums[low]**<=**nums[mid]){...} → **Left Side** is **SORTED** Portion (don’t forget to Put the “equal” sign)

else{...} → **Right Side** is **UnSORTED** Portion

**Trick:** Always Try to find which Portion from the “**middle**” (**Left Side Sorted or Right Side UnSorted**) is Sorted… And based on that Condition we can approach from there.

// "mid" is In Left Sorted Portion

// don’t forget the “**equal**” sign for finding the “left” portion

if(nums[low]<=nums[mid]){

...

}

// "mid" is In Right UnSorted Portion

else{

...

}

* **Minimum element** in Rotated Sorted Array → find where **middle** is? is it “**Sorted** (**nums[low]<=nums[mid]**)” or “(else) **Unsorted**” section? Then pick the “**minimum**” from the “sorted/unsorted” section in which “mid” is present & Store it to a “ans” variable & remove the Rest of the values of that Particular section and continue the while loop → Explanation “**Striver/Take You Forward**”
* **Rotated Sorted Array** index **where it was “rotated”** → we find the “**Minimum**” Element & “Minimum” Element’s “**index**” in “Rotated Sorted Array”... The **“index”** of the **“Minimum” element ==** where the **Array was “rotated”**

**int findMin(vector<int>& nums) {**

**int low=0;**

**int high=nums.size()-1;**

**int ans=INT\_MAX;**

**//find where middle is? is it “Sorted (nums[low]<=nums[mid])” or “(else) Unsorted” section? Then pick the “minimum” from the “sorted/unsorted” section & Store it to a “ans” variable & remove the Rest of the values of that section and continue the while loop**

**while(low<=high){**

**int mid=low+((high-low)/2);**

**// "mid" is in Left Sorted Portion**

**if(nums[low]<=nums[mid]){**

**ans=min(ans, nums[low]); // in this Portion is lowest value is "low"**

**low=mid+1;**

**}**

**// "mid" is in Right UnSorted Portion**

**else{**

**ans=min(ans, nums[mid]); // in this Portion is lowest value is "mid"**

**high=mid-1;**

**}**

**}**

**return ans;**

**}**

**First & Last Occurrence**

**First Occurrence** → **Left Side** ee **Pointer move** korate takhbo **First Occur** → So, Equal to “target” huile → “High” reh soraiya “Mid-1” ee niye ashbo

**Last Occurrence** → **Right Side** ee **Pointer move** korate takhbo **Last Occur** → So, Equal to “target” huile → “Low” reh soraiya “Mid+1” ee niye ashbo

vector<int> searchRange(vector<int>& nums, int target) {

int low = 0;

int high = nums.size()-1;

int firstOccr = -1, lastOccr=-1;

vector<int> ans;

//First Occurrence

while(low<=high){

int mid = low + ((high-low)/2);

if(nums[mid]==target){

**firstOccr = mid;**

**high = mid-1;**

}

if(nums[mid]<target){

low = mid+1;

}

if(nums[mid]>target){

high = mid-1;

}

}

//Second Occurrence

//Re-initialize the "low" & "high" variable

low = 0;

high = nums.size()-1;

while(low<=high){

int mid = low + ((high-low)/2);

if(nums[mid]==target){

**lastOccr = mid;**

**low = mid+1;**

}

if(nums[mid]<target){

low = mid+1;

}

if(nums[mid]>target){

high = mid-1;

}

}

ans.push\_back(firstOccr);

ans.push\_back(lastOccr);

return ans;

}

**Lower & Upper Bound**

**Lower Bound:** First Index which fulfills the Condition of “**nums[i]>=target**”

**Upper Bound:** First Index which fulfills the Condition of “**nums[i]>target**” → NO “**EQUAL**” Condition

N.B.: “**ans**” should be Initialized with “**len**” (Not “**len-1**” OR, “**0**”)

## **Lower Bound**

class Solution {

public:

int lowerBound(vector<int>& nums, int target) {

int len = nums.size();

int low = 0;

int high = len-1;

int ans = len;

// in Lower Bound we find the First Index from the Left that Satisfies the condition --> nums[idx] >= target

while(low<=high){

int mid = low + ((high-low)/2);

if(nums[mid] >= target){

ans = mid;

high = mid-1; // search on the Left

}

else{

low = mid+1; // search on the Right

}

}

return ans;

}

};

## **Upper Bound**

class Solution {

public:

int upperBound(vector<int>& nums, int target) {

// in Upper Bound we find the First Index from the Left that Satisfies the condition --> nums[idx] > target

// Difference Between Lower & Upper Bound Condition is that there is No EQUAL Sign ("=") in the Condition

int len = nums.size();

int low = 0;

int high = len-1;

int ans = len;

while(low<=high){

int mid = low + ((high-low)/2);

if(nums[mid]>target){

ans = mid;

high = mid-1; // search on the Left

}

else{

low = mid+1; // search on the Right

}

}

return ans;

}

};

**Search & Insert Position**

Searching for 1st Insert Position → Left Side ee takbe Insert position → So, “target” Mid theke Bigger huile → “Low” reh soraiya “Mid+1” ee niye ashbo → (“**firstPos**” = New Value “**Low**”) assign korbo… That means “Lower Bound” ber korte hobe…

Lower Bound: First Index which fulfills the Condition of “nums[i]>=target”

“**Lower Bound**” == **Insert Position**

class Solution {

public int searchInsert(int[] nums, int target) {

int len = nums.length;

int low = 0;

int high = len-1;

int ans = len;

// search Insert Position == Lower Bound

// in Lower Bound we find the First Index from the Left that Satisfies the condition --> nums[idx] >= target

while(low<=high){

int mid = low + ((high-low)/2);

if(nums[mid]>=target){

ans = mid;

high = mid-1; // search on the Left

}

else{

low = mid+1; // search on the Right

}

}

return ans;

}

}