(without duplicate)

* 8] Search in Rotated Array

arr = [3, 4, 5, 6, 7, 0, 1, 2]
(0 1 2 3 4 5 6 7)
asc des asc
Pivot

- Find pivot in Rotated array
- search in first half => single BS (0, pivot)
- otherwise, search in second half => (pivot+1, length-1)

Finiding pivot

Pivot
[3, 4, 5, 6, 7, 0, 1, 2]
(0 1 2 3 4 5 6 7)
Des

*

=> Case-1

```
if(arr[mid] > arr[mid+1]){
    // we found pivot (ans = mid)
    return mid;
}
```

=> Case-2

```
if(arr[mid] < arr[mid-1]){
    // we found pivot (ans = mid-1)
    return mid-1;
}
```

note:

below cases (3 & 4) are for without duplicates

=) case-3

```
if (arr[start] ≥ arr[mid]){
    // in this case all elements from middle
    // will be < start.

    // Hense, we can ignore all these
    // elements and look for peak in left
    // side of arr

        end = mid-1;
}
```

=) case-4

```
if (arr[start] < arr[mid]){
    start = mid+1; // ignore all elements of
                   left side from middle
                   and look pivot in right
}                  side of arr.
```

=) // Now we have found a peak element so we can apply binary search in rotated array

For Binary Search there are three cases:

=) (case 1 =) arr[pivot]

```
if (pivot element == target){
    // ans = pivot
}
```

t > s ↙ lies in here
[s ////////p ——— e]

=> case-2
if (target > arr[start]) {
    search (s, pivot - 1)
            start
}

t < s ↙ lies in here

=> Case-3   [s   p ///////e]
if (target < arr[start]) {
    search (pivot + 1, end)
}

* **with duplicates find ~~peak~~ pivot**

we can use case 1 & case 2 of finding pivot

=> Case => 3

[2, 2, 2, 2, 9]
    ⇓ rotate twice

s↓ 0, 1, 2↙m, 3, 4↙e
[2, 9, 2, 2, 2]

Here
start, middle and end all are ecual. so we can not uses past cases(3, 4) for finding pivot

```
if (arr[mid] == arr[start] && arr[mid] == arr[end])
{
    // if all are equal we skip the start
    and end and update by 1.

=) // but before updating check if start
    is pivot

    if (start < end && arr[start] > arr[start+1])
    {
        return start; //ans
    }
    // if start is not pivot increment it by 1.
        start++;

=) // check if end is pivot

    if (end > start && arr[end] < arr[end-1])
    {
        return end; //ans
    }

    // if end is not pivot decrement by 1.

        end--;

}
```

=) Case 4:

```
if(arr[start] < arr[mid] ||
    (arr[start] == arr[mid] &&
        arr[mid] > arr[end])) {
    start = mid+1;
}
else {
    end = mid-1;
}
```

* Let's code:

```
//without duplicates

findPivot(int[] arr){

    int start = 0;
    int end = arr.length -1;

    while(start <= end){

        int mid = start + (end - start) /2;

        if(mid < end && arr[mid] > arr[mid+1])
        {
            return mid;
        }

        if(mid > start && arr[mid] < arr[mid-1])
        {
            return mid-1;
        }
```

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

  return -1;

}

// With duplicates

FindPivotWithDuplicates (int[] arr) {

    int start = 0;
    int end = arr.length - 1;

    while (start <= end) {

        int mid = start + (end - start) / 2;

        if (mid < end && arr[mid] > arr[mid+1])
        {
          return mid;
        }

        if (mid > start && arr[mid] < arr[mid-1])
        {
          return mid-1;
        }
```

```
// if start, end and mid are equal
if (arr[mid] == arr[start] &&
        arr[mid] == arr[end])
{

  //check if start is pivot

  if (start < end && arr[start] > arr[start+1])
  {
    return start;
  }

  // if start is not pivot
  start++;

  //check if end is Pivot

  if (end > start && arr[end] < arr[end-1])
  {
    return end-1;
  }

  // if end is not Pivot
  end--;
}

else if (arr[start] < arr[end] ||
    (arr[start] == arr[mid] &&
      arr[mid] > arr[end])) {
  start = mid+1;
} else {
    end = mid-1;
}
return -1;
}
```

Now, after finding pivot apply binary search

```
search(int[] arr, int target){
    // if without duplicates
    int pivot = ~~find~~ ~~pivot~~ findPivot(arr);

                or
    // if duplicates are available

    int pivot = findPivotWithDuplicates(arr);

    if(pivot == -1){
        // if you did not find pivot means,
           arr is not rotated.
        // so apply simple normal binary search

        return binarySearch(arr, target, 0,
                                arr.length-1);
    }

    if(arr[pivot] == target){
        return pivot;
    }
    // target lie in left from pivot
    if(target >= arr[0]){
        return binarySearch(arr, target, 0, pivot-1);
    } // target lie in right from pivot
    return binarySearch(arr, target, pivot+1,
                            arr.length-1);
}
```

```
binarySearch(int[] arr, int target, int start,
                                    int end)
{
    while( start <= end)
    {
      int mid = start +(end - start) / 2;

      if (target > arr[mid]){
          start = mid + 1;
      }
      else if (target < arr[mid]){
          end = mid - 1;
      }else{
          return mid;
      }
    }

  return -1;
}
```