Kunal Kushwaha DSA Bootcamp 3

SEARCHING -

Basic binary search -

Note - for large mid that exceed integer range , mid formula =

mid = low + (high-low)/2

```
public class Searching{
public static void main(String[] args) {

int[] nums= {2,4,6,9,11,25,34,57};
int target  = 6 ;

int low = 0;
int high = nums.length-1 ;


  while(low<=high){
  int mid  = (low+high)/2;
  if(target==nums[mid]){
    System.out.println("target is found at "+ mid );
    break ;
  }
  else if(target>nums[mid]){
  low = mid+1;
  }

  else if(target<nums[mid]){
    high = mid-1 ;
if(low>high){
  System.out.println("no such element");
}
}
}
```

using methods -

```
public class Searching{
public static void main(String[] args) {
int[] arr= {-12, -9, 2,4,6,9,11,25,34,57};
int target  = 258;
int index = binarysearch(arr,target);
System.out.println(index);
}
```

```
static int binarysearch(int nums[], int target){
  int low = 0;
  int high = nums.length-1 ;

  while(low<=high){
  int mid  = low+(high-low)/2;
  if(target==nums[mid]){
   return mid;

  }
  else if(target>nums[mid]){
  low = mid+1;
  }

  else if(target<nums[mid]){
    high = mid-1 ;
  }

}
return -1;
}
}
```

KK code for binary search -

```
public class BinarySearch {

    public static void main(String[] args) {
        int[] arr = {-18, -12, -4, 0, 2, 3, 4, 15, 16, 18, 22, 45, 89};
        int target = 22;
        int ans = binarySearch(arr, target);
        System.out.println(ans);
    }

    // return the index
    // return -1 if it does not exist
    static int binarySearch(int[] arr, int target) {
        int start = 0;
        int end = arr.length - 1;

        while(start <= end) {
            // find the middle element
//            int mid = (start + end) / 2; // might be possible that (start + end) exceeds the range of int in java
            int mid = start + (end - start) / 2;

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

Order agnostic binary search -

```
public class OrderAgnosticBS {
    public static void main(String[] args) {
//        int[] arr = {-18, -12, -4, 0, 2, 3, 4, 15, 16, 18, 22, 45, 89};
        int[] arr = {99, 80, 75, 22, 11, 10, 5, 2, -3};
        int target = 22;
        int ans = orderAgnosticBS(arr, target);
        System.out.println(ans);
    }

    static int orderAgnosticBS(int[] arr, int target) {
```

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

        // find whether the array is sorted in ascending or descending
        boolean isAsc = arr[start] < arr[end];

        while(start <= end) {
            // find the middle element
//            int mid = (start + end) / 2; // might be possible that (start + end) exceeds the range of int in java
            int mid = start + (end - start) / 2;

            if (arr[mid] == target) {
                return mid;
            }

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

Ceiling of a number - (KK code)

```
public class Ceiling {

    public static void main(String[] args) {
        int[] arr = {2, 3, 5, 9, 14, 16, 18};
        int target = 15;
        int ans = ceiling(arr, target);
        System.out.println(ans);
    }

    // return the index of smallest no >= target
    static int ceiling(int[] arr, int target) {

        // but what if the target is greater than the greatest number in the array
        if (target > arr[arr.length - 1]) {
            return -1;
        }
        int start = 0;
        int end = arr.length - 1;

        while(start <= end) {
            // find the middle element
//            int mid = (start + end) / 2; // might be possible that (start + end) exceeds the range of int in java
            int mid = start + (end - start) / 2;

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

OR

(my code{ I am returning the number instead of the index})

```
public class Searching{
public static void main(String[] args) {
int[] arr= {-12, -9, 2,4,6,9,11,25,34,57};
int target  = -12;

int low = 0;
int high = arr.length-1;
while(low<=high){
int mid = low+(high-low)/2  ;
if(target>=arr[arr.length-1]){
  System.out.println("no ans");
  break;
}
if( target==arr[mid]){
  System.out.println(arr[mid]+" is the ans");
  break;
}
else if(target>arr[mid]){
  low  =mid+1 ;
  }
else if(target<arr[mid]){
  high = mid-1 ;
  }
if(low>high){
  System.out.println(arr[low]+ " is the ans");
}
}
}
}
```

374. Guess Number Higher or Lower

```
public class Solution extends GuessGame {
    public int guessNumber(int n) {
    int low=1,high=n,mid=n;
      while(guess(mid)!=0){
          mid=low+(high-low)/2;
          if(guess(mid)==1){
             low=mid+1;
          }
          else{
              high=mid-1;
          }
      }
      return mid;
    }
}
```

744. Find Smallest Letter Greater Than Target

```
class Solution {
    public char nextGreatestLetter(char[] letters, char target) {
        int low = 0;
        int high = letters.length -1;

        while(low<=high){
```

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

             if(target>=letters[mid])
                low= mid+1;
            else
                high = mid-1;
        }

            return letters[low%letters.length];   // modulud is used for the last element
                                                   // to print the first element
    }

}
```

OR (without using modulus)

```
class Solution {
    public char nextGreatestLetter(char[] letters, char target) {
        int low = 0;
        int high = letters.length -1;

        if(target>=letters[letters.length-1])
            return letters[0];

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

             if(target>=letters[mid])
                low= mid+1;
            else
                high = mid-1;
        }
            return letters[low];
    }
}
```

34. Find First and Last Position of Element in Sorted Array

```
int[] result={-1,-1};
  int low = 0;
  int high = nums.length-1 ;

//binary search for the first index

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

            if(target==nums[mid]){
                result[0] = mid ;
                high = mid-1 ; // when found an occurence, check for the left side of the remaining array by running binary search again, f
            }
            else if(target>nums[mid]){
                low=mid+1;
            }

            else{
                high = mid-1 ;
            }
        }
      //binary search for the last index
     low = 0;
  high = nums.length-1 ;
          while(low<=high){
            int mid = low + (high-low)/2 ;
```

```
            if(target==nums[mid]){
                result[1] = mid ;
                low= mid+1 ;   // when found an occurence, check for the right side of the remaining array by running binary search again ,
            }
            else if(target>nums[mid]){
                low=mid+1;
            }

            else{
                high = mid-1 ;
            }
        }
        return result  ;
```

infinite array - GFG

Find position of an element in a sorted array of infinite numbers - GeeksforGeeks

Suppose you have a sorted array of infinite numbers, how would you search an element in the array?Source: Amazon Interview Experience. Since array is sorted, the first thing clicks into mind is binary search, but the problem here is that we don't know size of array.

https://www.geeksforgeeks.org/find-position-element-sorted-array-infinite-numbers/

KK code -

```
public class InfiniteArray {
    public static void main(String[] args) {
        int[] arr = {3, 5, 7, 9, 10, 90,
                100, 130, 140, 160, 170};
        int target = 10;
        System.out.println(ans(arr, target));
    }
    static int ans(int[] arr, int target) {
        // first find the range
        // first start with a box of size 2
        int start = 0;
        int end = 1;

        // condition for the target to lie in the range
        while (target > arr[end]) {  // doesn't need to check for the element at lower index
            int newStart = end + 1; // this is my new start
            // double the box value
            // end = previous end + sizeofbox*2
            end = end + (end - start + 1) * 2;
            start = newStart;
        }
        return binarySearch(arr, target, start, end);

    }
    static int binarySearch(int[] arr, int target, int start, int end) {
        while(start <= end) {
            // find the middle element
//            int mid = (start + end) / 2; // might be possible that (start + end) exceeds the range of int in java
            int mid = start + (end - start) / 2;

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

852. Peak Index in a Mountain Array ; 162. Find Peak Element

kk-sol

```
public int peakIndexInMountainArray(int[] arr) {
        int start = 0;
        int end = arr.length - 1;

        while (start < end) {
            int mid = start + (end - start) / 2;
            if (arr[mid] > arr[mid+1]) {
                // you are in dec part of array
                // this may be the ans, but look at left
                // this is why end != mid - 1
                end = mid;
            } else {
                // you are in asc part of array
                start = mid + 1; // because we know that mid+1 element > mid element
            }
        }
        // in the end, start == end and pointing to the largest number because of the 2 checks above
        // start and end are always trying to find max element in the above 2 checks
        // hence, when they are pointing to just one element, that is the max one because that is what the checks say
        // more elaboration: at every point of time for start and end, they have the best possible answer till that time
        // and if we are saying that only one item is remaining, hence cuz of above line that is the best possible ans
        return start; // or return end as both are =
    }
```

1095. Find in Mountain Array

My code (works fine in VScode but not in leetcode) -

```
int nums[] = {1,2,3,4,5,3,1} ;

int target = 3;
int peak = peakindex(nums);
int ans = ascbinarysearch(nums,peak,target) ; // search before the peak element
if(ans==-1) // if not found , searching after the peak element
 ans = decbinarysearch(nums,peak,target);
System.out.println(ans);
}//output =

static int peakindex(int[] nums ){

int low = 0;
int high = nums.length-1;
while( low<high){
    int mid = low+ ( high-low)/2 ;

      if(nums[mid]>nums[mid+1]){
        high=mid;
    }
       else
         low=mid+1;

    }

return low;
 }

 static int ascbinarysearch(int[] arr, int index , int target){

     int low = 0;
     int high = index;
     while(low<=high){
        int mid = low + ( high-low)/2;
       if(target==arr[mid]){
         return mid;
```

```
      }

      else if(target< arr[mid]){
        high  =mid-1;
      }
      else{
        low=mid+1 ;
      }
    }
    return -1;
}

    static int decbinarysearch(int[] arr, int index , int target){

       int low = index;
       int  high = arr.length-1;

        while(low<=high){
            int mid = low + ( high-low)/2;
          if(target==arr[mid]){
          return mid ;

          }

          else if(target< arr[mid]){
            low=mid+1 ;
          }
          else{
            high=mid-1 ;
          }
     }
  return -1;
 }

}
```

KK code (using order agnostic binary search) -

```
class Solution {
    public int findInMountainArray(int target, MountainArray mountainArr) {
  int peak = findPeakElement(mountainArr);
        int firstTry = OrderAgnosticBinarySearch(mountainArr, target, 0, peak);  // search before the peak element
        if(firstTry == -1){  // search after the peak element if not found
            return OrderAgnosticBinarySearch(mountainArr, target, peak+1, mountainArr.length()-1);
        }
        return firstTry;
    }

    static int findPeakElement(MountainArray nums) {
        int start = 0;
        int end = nums.length() - 1;

        while(start < end){
            int mid = start + (end - start) / 2;
            if(nums.get(mid) > nums.get(mid+1)){
                end = mid;
            }else{
                start = mid + 1;
            }
        }
        return start;
    }

    static int OrderAgnosticBinarySearch(MountainArray arr, int target, int start, int end){

        boolean isAsc = arr.get(start) < arr.get(end);

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

            if(arr.get(mid) == target)
                 return mid;

            if(isAsc){
```

```
                if(arr.get(mid) < target)
                    start = mid + 1;
                else
                    end = mid - 1;
            }
             else{
                if(arr.get(mid) > target)
                     start = mid + 1;
                else
                      end = mid - 1;
            }
        }
        return -1;
    }
}
```

33. Search in Rotated Sorted Array

```
public class RBS {
    public static void main(String[] args) {
        int[] arr = {1,2,3,4,5,5,6};
        System.out.println(findPivotWithDuplicates(arr));
    }

    static int search(int[] nums, int target) {
        int pivot = findPivot(nums);

        // if you did not find a pivot, it means the array is not rotated
        if (pivot == -1) {
            // just do normal binary search
            return binarySearch(nums, target, 0 , nums.length - 1);
        }

        // if pivot is found, you have found 2 asc sorted arrays
        if (nums[pivot] == target) {
            return pivot;
        }

        if (target >= nums[0]) {
            return binarySearch(nums, target, 0, pivot - 1);
        }

        return binarySearch(nums, target, pivot + 1, nums.length - 1);
    }

    static int binarySearch(int[] arr, int target, int start, int end) {
        while(start <= end) {
            // find the middle element
//            int mid = (start + end) / 2; // might be possible that (start + end) exceeds the range of int in java
            int mid = start + (end - start) / 2;

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

    // this will not work in duplicate values
    static int findPivot(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
```

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

    static int findPivotWithDuplicates(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
            }
            if (mid > start && arr[mid] < arr[mid - 1]) {
                return mid-1;
            }

            // if elements at middle, start, end are equal then just skip the duplicates
            if (arr[mid] == arr[start] && arr[mid] == arr[end]) {
                // skip the duplicates
                // NOTE: what if these elements at start and end were the pivot??
                // check if start is pivot
                if (start < end && arr[start] > arr[start + 1]) {
                    return start;
                }
                start++;

                // check whether end is pivot
                if (end > start && arr[end] < arr[end - 1]) {
                    return end - 1;
                }
                end--;
            }
            // left side is sorted, so pivot should be in right
            else if(arr[start] < arr[mid] || (arr[start] == arr[mid] && arr[mid] > arr[end])) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        return -1;
```

or

```
class Solution {
    public int search(int[] nums, int target) {

        int low  =0 ;
        int high  =nums.length -1  ;
        int mid = 0 ;

        while(low<=high){
            mid = low+ (high -low)/2 ;

            if(nums[mid]==target){
                return mid ;
            }

            // left side is sorted
            if(nums[low]<=nums[mid]){
         if( nums[low]<= target && target <= nums[mid]){
               high = mid-1 ;
         }
             else
                 low  =mid+1 ;
```

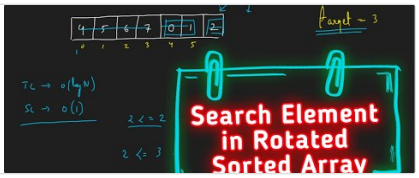
```
            }


            else{
                if(nums[mid]<=target && target<= nums[high]){
                    low  =mid+1 ;
                }
                else
                    high = mid -1  ;
            }
    }
        return -1;
}
}
```

Search Element In a Rotated Sorted Array | Leetcode

Check our Website: https://www.takeuforward.org/ In case you are thinking to buy courses, please check below: Link to get 20% additional Discount at Coding Ninjas: https://bit.ly/3wE5aHx Code "takeuforward" for 15% off at GFG: https://practice.geeksforgeeks.org/courses Code "takeuforward" for 20% off on sys-design: https://get.interviewready.io?

https://youtu.be/r3pMQ8-Ad5s



410. Split Array Largest Sum

Split Array Largest Sum - LeetCode

Given an integer array nums and an integer k, split nums into k non-empty subarrays such that the largest sum of any subarray is minimized. Return the minimized largest sum of the split. A subarray is a contiguous part of the array.

https://leetcode.com/problems/split-array-largest-sum/

kk code-

```
package com.kunal;

public class SplitArray {
    public static void main(String[] args) {

    }

    public int splitArray(int[] nums, int m) {
        int start = 0;
        int end = 0;

        for (int i = 0; i < nums.length; i++) {
            start = Math.max(start, nums[i]); // in the end of the loop this will contain the max item of the array
            end += nums[i];
        }

        // binary search
        while (start < end) {
            // try for the middle as potential ans
            int mid = start + (end - start) / 2;

            // calculate how many pieces you can divide this in with this max sum
            int sum = 0;
            int pieces = 1;
            for(int num : nums) {
                if (sum + num > mid) {
                    // you cannot add this in this subarray, make new one
                    // say you add this num in new subarray, then sum = num
                    sum = num;
```

```
                    pieces++;
                } else {
                    sum += num;
                }
            }

            if (pieces > m) {
                start = mid + 1;
            } else {
                end = mid;
            }

        }
        return end; // here start == end
    }
}
```

Rotation count -

kk code -

```
package com.kunal;

public class RotationCount {
    public static void main(String[] args) {
        int[] arr = {4,5,6,7,0,1,2};
        System.out.println(countRotations(arr));
    }

    private static int countRotations(int[] arr) {
        int pivot = findPivot(arr);
        return pivot + 1;
    }

    // use this for non duplicates
    static int findPivot(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
            }
            if (mid > start && arr[mid] < arr[mid - 1]) {
                return mid-1;
            }
            if (arr[mid] <= arr[start]) {
                end = mid - 1;
            } else {
                start = mid + 1;
            }
        }
        return -1;
    }

    // use this when arr contains duplicates
    static int findPivotWithDuplicates(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
            }
            if (mid > start && arr[mid] < arr[mid - 1]) {
                return mid-1;
            }

            // if elements at middle, start, end are equal then just skip the duplicates
            if (arr[mid] == arr[start] && arr[mid] == arr[end]) {
```

```
                // skip the duplicates
                // NOTE: what if these elements at start and end were the pivot??
                // check if start is pivot
                if (arr[start] > arr[start + 1]) {
                    return start;
                }
                start++;

                // check whether end is pivot
                if (arr[end] < arr[end - 1]) {
                    return end - 1;
                }
                end--;
            }
            // left side is sorted, so pivot should be in right
            else if(arr[start] < arr[mid] || (arr[start] == arr[mid] && arr[mid] > arr[end])) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
        return -1;
    }
}
```

2D matrix Searching-

My code -

```
public class Searching{
public static void main(String[] args) {
int[][] matrix = {{30,40,60},{32,42,62},{35,45,65},{37,47,67}} ;
int target =67 ;
int[] ans = twoDsearching(matrix , target);
System.out.println(Arrays.toString(ans));
}

static int[] twoDsearching (int[][] matrix , int target){
 int row = 0;
int col  =matrix[0].length-1 ;

while(col>=0 && row<matrix.length){

  if(matrix[row][col]==target){
    return new int[]{row,col} ;
  }

  else if(matrix[row][col] > target){
    col -- ;
  }
  else{
    row++ ;
  }
}
return new int[] {-1,-1};
}

}
```

Sorted matrix binary search -

kk code -

```
import java.util.Arrays;

public class SortedMatrix {
```

```
    public static void main(String[] args) {
        int[][] arr = {
                {1, 2, 3},
                {4, 5, 6},
                {7, 8, 9}
        };
        System.out.println(Arrays.toString(search(arr, 9)));
    }

    // search in the row provided between the cols provided
    static int[] binarySearch(int[][] matrix, int row, int cStart, int cEnd, int target) {
        while (cStart <= cEnd) {
            int mid = cStart + (cEnd - cStart) / 2;
            if (matrix[row][mid] == target) {
                return new int[]{row, mid};
            }
            if (matrix[row][mid] < target) {
                cStart = mid + 1;
            } else {
                cEnd = mid - 1;
            }
        }
        return new int[]{-1, -1};
    }

    static int[] search(int[][] matrix, int target) {
        int rows = matrix.length;
        int cols = matrix[0].length; // be cautious, matrix may be empty
        if (cols == 0){
            return new int[] {-1,-1};
        }
        if (rows == 1) {
            return binarySearch(matrix,0, 0, cols-1, target);
        }

        int rStart = 0;
        int rEnd = rows - 1;
        int cMid = cols / 2;

        // run the loop till 2 rows are remaining
        while (rStart < (rEnd - 1)) { // while this is true it will have more than 2 rows
            int mid = rStart + (rEnd - rStart) / 2;
            if (matrix[mid][cMid] == target) {
                return new int[]{mid, cMid};
            }
            if (matrix[mid][cMid] < target) {
                rStart = mid;
            } else {
                rEnd = mid;
            }
        }

        // now we have two rows
        // check whether the target is in the col of 2 rows
        if (matrix[rStart][cMid] == target) {
            return new int[]{rStart, cMid};
        }
        if (matrix[rStart + 1][cMid] == target) {
            return new int[]{rStart + 1, cMid};
        }

        // search in 1st half
        if (target <= matrix[rStart][cMid - 1]) {
            return binarySearch(matrix, rStart, 0, cMid-1, target);
        }
        // search in 2nd half
        if (target >= matrix[rStart][cMid + 1] && target <= matrix[rStart][cols - 1]) {
            return binarySearch(matrix, rStart, cMid + 1, cols - 1, target);
        }
        // search in 3rd half
        if (target <= matrix[rStart + 1][cMid - 1]) {
            return binarySearch(matrix, rStart + 1, 0, cMid-1, target);
        } else {
            return binarySearch(matrix, rStart + 1, cMid + 1, cols - 1, target);
        }
    }
}
```

69. Sqrt(x)

```
class Solution {
    public int mySqrt(int x) {
        long low  =0;
        long high = x ;
        int ans = 0;
    while(low<=high){
        long mid = low + (high -low)/2 ;
        if(mid*mid==x){
           ans = (int)mid ;
            break ;
        }

        else if(mid*mid<x){
            low =mid+1;
            ans =(int)mid  ;
        }
        else {
            high =mid-1;
        }
    }

     return ans ;
    }
}
```

374. Guess Number Higher or Lower

```
public class Solution extends GuessGame {
    public int guessNumber(int n) {
    int low=1,high=n , mid = n ;
      while(guess(mid)!=0){
          mid=low+(high-low)/2;
          if(guess(mid)==1){
             low=mid+1;
          }
          else{
              high=mid-1;
          }
      }
      return mid;


    }
}
```

278. First Bad Version

```
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {

        int low =1;
        int high = n;
        int mid = 0;
        int ans=0 ;
        while(low<=high){
            mid= low + (high-low)/2 ;
            if(isBadVersion(mid)==true){
                ans = mid ;  //store the ans
                high = mid-1; //search in thhe lower indices for accurate answers
```

```
            }
            else
               low = mid+1 ;
        }

        return ans ;
    }
}
```

167. Two Sum II - Input Array Is Sorted

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {

        for(int i =0 ;i<numbers.length ; i++){
         int low =i+1; // search for indices in between one more than itself and the last

        int high = numbers.length-1 ;
        int mid= 0;

        while(low<=high){
            mid = low+(high-low)/2 ;
            if(numbers[i]+ numbers[mid]==target){
            return new int[]{i+1, mid+1} ;
            }
            if(numbers[i]+ numbers[mid]<target)
                low=mid+1;

            else
                high = mid-1 ;
        }

    }
     return new int[]{ -1,-1} ;

    }
}
```

367. Valid Perfect Square

```
class Solution {
    public boolean isPerfectSquare(int num) {

       long low = 1;
        long high = num ;
        long mid =0;

       while(low<=high){
           mid = low+(high-low)/2 ;
           if(mid*mid==num){
              return true;
           }

           else if(mid*mid>num){
               high=mid-1;
           }
           else{
               low=mid+1;
           }

       }

        return false;
    }
}
```

441. Arranging Coins

```
class Solution {
    public int arrangeCoins(int n) {
        long low = 0;
        long high= n;
        long mid = 0 ;
        long count =0 ;
        while(low<=high){
            mid = low+(high-low)/2 ;
            if(mid*(mid+1)/2==n){  // sum of n numbers
                count =mid;
                break ;
            }

             if(mid*(mid+1)/2>n)
                 high =mid-1;
            else{
                count = mid;
                low=mid+1 ;
            }
        }
        return (int)count ;
    }
}
```

1539. Kth Missing Positive Number

```
public class Searching{
public static void main(String[] args) {
int[] arr = {2,4,9,14,17} ;
int  num = 5 ;
int ans = findKthPositive( arr,num);
System.out.println(ans);
}


public static int findKthPositive(int[] arr, int k) {
  int start = 0, end = arr.length;
  while(start < end){
      int mid = start + ((end - start)/2);
      if((arr[mid] - (mid  + 1)) >= k){
          end = mid;
      }else{
         start = mid + 1;
      }
  }
  return start + k ;
}
}
```

35. Search Insert Position

```
class Solution {
    public int searchInsert(int[] nums, int target) {
        int low = 0;
        int high = nums.length-1 ;
          int mid=0 ;
        while(low<=high){
             mid = low +(high-low)/2 ;
            if(nums[mid]==target){
                return mid ;
            }
```

```
            else if(nums[mid]> target){
                high = mid-1 ;

            }

            else
                low =mid+1 ;
        }
     if(target>nums[mid])
         return mid+1;

        else
         return mid ;
    }
}
```

1351. Count Negative Numbers in a Sorted Matrix

```
class Solution {
    public int countNegatives(int[][] grid) {
    int count = 0;
        int row = 0, col = grid[0].length - 1;

        while (row < grid.length && col >= 0) {
            if (grid[row][col] < 0) {
                count += (grid.length - row); // if any element in a column is a negative , the entire col is going to be negative,
                                              // hence subtract no of columns to the now number
                col--;
            }
            else row++;
        }

        return count;
}

}
```

349. Intersection of Two Arrays

```
class Solution {
    public int[] intersection(int[] nums1, int[] nums2) {

        Arrays.sort(nums1);
        Arrays.sort(nums2); // sort nums2[] to avoid duplicates

        ArrayList<Integer> res = new ArrayList<>();
        for (int i = 0; i < nums2.length; i++) {
            if (i >= 1 && nums2[i] == nums2[i - 1]) {
                continue;
            }
            if (binarysearch(nums1, nums2[i]) == true) {
                res.add(nums2[i]);
            }
        }

        return res.stream().mapToInt(i -> i).toArray(); // arraylist of Integer to int[] array
    }
    private boolean binarysearch(int[] arr, int target) {
        int lo = 0;
        int hi = arr.length - 1;

        while (lo <= hi) {
            int mid = lo + (hi - lo) / 2;
```

```
            if (arr[mid] == target) {
                return true;
            } else if (arr[mid] < target) {
                lo = mid + 1;
            } else {
                hi = mid - 1;
            }
        }
        return false;
    }
}
```

*Let Sum of choclates by Alice Be Sa and Sum of choclates by Alice Be Sb

Let the choclates from Alice to Bob be A and from Bob to Alice be B

Sa -A + B = Sb -B+A
2(A-B) = (Sa-Sb)
A-B = (Sa-Sb)/2
A = B+(Sa-Sb)/2

So the goal is to search the elements of Alice such that Bob + (Diff of Sum) exists in the Alice
For optimisation Alice is sorted and Binary search is applied on Alice*

```
class Solution {
    private int binarySearch(int[] arr,int target){

        int s = 0, e = arr.length-1;

        while(s <= e)
        {
            int mid = s + (e-s)/2;
            if(arr[mid] == target) return mid;
            else if(arr[mid] < target) s=mid+1;
            else e = mid - 1;
        }
        return -1;
    }

    public int[] fairCandySwap(int[] a, int[] b) { //a = Alice , b=Bob

        int suma=0,sumb=0;
        for(int i = 0; i < a.length; i++)
            suma += a[i];

        for(int i = 0; i < b.length; i++)
            sumb +=b [i];

        int diff = (suma - sumb)/2;

        Arrays.sort(a);
        for(int num : b)
        {
            if (binarySearch(a, num + diff) !=-1)
                return new int[] { num + diff, num };
        }

        return null;
    }
}
```

350. Intersection of Two Arrays II

```
using binary search, search for the elements of the smallest array (nums1) in the largest array (nums2)

sort the largest array o that binary search is feasible
sort the smallest array so that we can seach sequentially

if element is found,
    keep searching to the left until we find the first occurrence of the element

    add element to the result

when element is found, keep track of the last index where element was found so that next binary search ignores previous used indexes
    ie. nums1 = 1,1   nums2 = 1,2,2 - output should be [1] - once we found first 1 at index 0 and next search is done as of index 1
```

```
public int[] intersect(int[] nums1, int[] nums2) {
  if(nums2.length < nums1.length){
    return intersect(nums2, nums1);
  }

  Arrays.sort(nums1);
  Arrays.sort(nums2);

  List<Integer> result = new ArrayList<>();
  int leftIndex = 0;
  for(int num: nums1){
    int index = binarySearch(nums2, num, leftIndex);

    if(index != -1){
      result.add(num);
      leftIndex = index + 1;
    }
  }

  return result.stream().mapToInt(Integer::intValue).toArray();
}

private int binarySearch(int[] nums, int target, int left){
  int right = nums.length - 1;
  int index = -1;

  while(left <= right){
    int middle = left + (right - left) / 2;

    if(nums[middle] == target){
      index = middle;

      right = middle - 1;
    } else if(nums[middle] > target){
      right = middle - 1;
    } else {
      left = middle + 1;
    }
  }

  return index;
}
```

1346. Check If N and Its Double Exist

```
class Solution {
    public boolean checkIfExist(int[] arr) {
        int target = 0;
        Arrays.sort(arr);
       for(int i =0; i<arr.length ;i++){
           target = arr[i]*2 ;
```

```
           if(checkdouble(arr , target, i)  == true)
               return true;
           }

        return false ;
    }

    public boolean checkdouble(int[] nums , int target ,int index){

        int low = 0 ;
        int high = nums.length -1 ;
        int mid =0;

        while(low<=high){
            mid  =low +(high-low)/2 ;

            if(mid!=index && target==nums[mid]){  // mid!=index so that it doesn't check itself
                return true;
            }

            else if(target> nums[mid]){
                low = mid+1 ;
            }

            else
                high =mid-1 ;
        }

        return false ;

    }
}
```

1608. Special Array With X Elements Greater Than or Equal X

```
public int specialArray(int[] nums) {
    int low = 0;
    int high = nums.length;
    int ans = -1;
     while(low <= high) {
        int mid = low + (high - low)/2;
        int count = 0;
        for(int i: nums) {
            if(i >= mid) {
                count++;
            }
        }
        if(count == mid) {
            ans = mid;
            break;
        }else if(count > mid) {
            low = mid + 1;
        }else {
            high = mid - 1;
        }
    }
    return ans;
}
```

UNSUCCESSFUL my code -

```
public class Searching{
public static void main(String[] args) {
int[] nums1 = {1,0,3,5,6};
int ans = specialarray(nums1);
System.out.println(ans);
```

```
}
Arrays.sort(nums1) ;
int result = -1 ;
int ans = 0 ;
for ( int i =0 ; i<nums1.length ;i++){
    ans = checkdouble(nums1 , i) ;
    if(ans == i)
    result=ans ;
    break;
}
System.out.println(result);
}

public static int checkdouble(int[] nums , int target){

  int low = 0 ;
  int high = nums.length -1 ;
  int mid =0;

  while(low<=high){
      mid  =low +(high-low)/2 ;

      if(target==nums[mid]){
       return (nums.length - mid) ;
      }
      else if(target> nums[mid]){
          low = mid+1 ;
      }
      else
          high =mid-1 ;
  }

  return (nums.length - low);

}
}
```

540. Single Element in a Sorted Array

```
class Solution {
    public int singleNonDuplicate(int[] nums) {
          int low = 0;
    int high = nums.length - 1;

    int n = nums.length;
    if (n == 1)
      return nums[0];

    if (nums[1] != nums[0])
      return nums[0];

    if (nums[n - 1] != nums[n - 2])
      return nums[n - 1];

    while (low <= high) {

      int mid = (low + high) / 2;

      if (nums[mid] != nums[mid - 1] && nums[mid] != nums[mid + 1])
        return nums[mid];

      if (nums[mid] == nums[mid - 1]) {

        int lc = mid - low + 1;

        if (lc % 2 == 0)
          low = mid + 1;

        else
          high = mid - 2;
      }
```

```
      if (nums[mid] == nums[mid + 1]) {

        int rc = high - mid + 1;

        if (rc % 2 == 0)
          high = mid - 1;

        else
          low = mid + 2;
      }

    }

    return -1;
    }
}
```

Or - (better code according to me)

idea -

we are checking if the length of one of the given half is even or odd , if even ,the answer lies on the other half because every element is paired or else if odd, answer resides in the current half .

```
class Solution {
    public int singleNonDuplicate(int[] nums) {
     public static int singleNonDuplicate(int[] nums) {
        if(nums.length==1)
        return nums[0];

    int low = 0;
    int high = nums.length-1;

    while(low<high){
        int mid = low+(high-low)/2;      // divide the array

        if(nums[mid]==nums[mid+1])
            mid = mid-1;      //two same elements should be in same half

        if((mid-low+1)%2!=0) high = mid;            // checking the length of left half. If its is odd then update ur right pointer to mid

        else low = mid+1;    // else your right half will be odd then update your left pointer to mid+1
    }

    return nums[low];
}
}
```

153. Find Minimum in Rotated Sorted Array

```
class Solution {
    public int findMin(int[] nums) {
      int pivot = findPivot(nums) ;
    int ans = nums[(pivot+1)%nums.length] ;
        return ans ;

    }

     public int findPivot(int[] arr) {
        int start = 0;
        int end = arr.length - 1;
        while (start <= end) {
            int mid = start + (end - start) / 2;
            // 4 cases over here
            if (mid < end && arr[mid] > arr[mid + 1]) {
                return mid;
            }
```

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