Binary Search Algorithm :

↳ Search a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeatedly check until the value is found or the interval is empty.

Ex:

0	1	2	3	4	5	6	7	8	9
2	5	8	12	16	23	38	56	72	91

Search 23,

L=0				M=4					H=9
2	5	8	12	16	23	38	56	72	91

$23 > 16$

take 2nd half

0	1	2	3	4	L=5	6	M=7	8	H=9
~~2~~	~~5~~	~~8~~	~~12~~	~~16~~	23	38	56	72	91

23 < 56

take 1st half

0	1	2	3	4	(L=5, M=5	H=6)	7	8	9
~~2~~	~~5~~	~~8~~	~~12~~	~~16~~	23	38	~~56~~	~~72~~	~~91~~

Found 23, Return 5

↳ We basically ignore half of the elements just after one comparison.

1. Compare x with middle element.
2. If x matches with the middle element, we return mid index.
3. Else if x is greater than the middle element, then x can only lie in the right half subarray after the mid element. So, we recur for the right half.
4. Else (x is smaller) recur for the left half.

Binary Search:

1. Find the middle element.
2. target > middle ⟹ search in the right. else search in the left.
3. If middle element == target element // [illegible] Ans → Middle
4. If start > end // element not found

Ex: Search 12 in [2, 4, 6, 9, 11, 12, 14, 20, 36, 48]

$$\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2, & 4, & 6, & 9, & \textcircled{11} & \boxed{12, & 14, & 20, & 36, & 48} \end{array}$$

1st

$$\begin{array}{ccccc} s & & m & & e \\ \boxed{12, & 14,} & \textcircled{20} & 36, & 48 \end{array}$$

$$\begin{array}{cc} s, m & e \\ \textcircled{12}, & 14 \end{array}$$

Ans = index 5

Imp

Time Complexity:

Best Case : O(1).

Worst Case : O(log n)

Ex: // Do binary search, & return the index of the element.

Imp

```
=> import java.util.Scanner;
public class BinarySearch {
    public static void main (String[] args) {
     Scanner in = new Scanner (System.in);
     System.out.println ("Enter the size of array :");
     int n = in.nextInt();
     int[] arr = new int [n];
     System.out.println ("Enter the array:");
     for (int i=0; i<n; i++)
          arr[i] = in.nextInt();
     System.out.println ("Enter target element:");
     int target = in.nextInt();
     System.out.println (Binarysearch (arr, target));
    }
    static int Binarysearch (int[] arr, int K) {
     int start = 0;
     int end = arr.length - 1;
     while (start <= end) {
          int mid = start + (end - start)/2;
          // or int mid = (start + end)/2;
          if (K < arr[mid]) {
               end = mid - 1;
          }
          else if (K > arr[mid]) {
               start = mid + 1;
          }
```

```
        else{
            return mid;     // Ans found
        }
    }
    return -1;     // In case not found
  }
}
```

Order-Agnostic Binary Search:

↳ It is a modified version of Binary Search Algo.
↳ In this modified Binary search, we check one more condition to find if the array is in ascending order or descending.

Imp

↳ Check ~~if the~~ / compare the first & last element.

- If the first element is smaller than the last element - then if the search key value X is less than the middle of the interval then the end pointer will be changed to middle - 1, otherwise start will be changed to middle + 1.
- If the first element is greater than the last element - then if the search key value X is less than the middle of the interval then the start pointer will move to the next element of the middle element otherwise the end pointer will move previous to the middle element.

Imp

Ex:

```
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 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 - 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;      }}
```