Binary Search Algorithm

Binary search is a searching algorithm which is used to search an element from a sorted array. It cannot be used to search from an unsorted array. Binary search is an efficient algorithm and is better than linear search in terms of time complexity.

The time complexity of linear search is O(n). Whereas the time complexity of binary search is O(log n). Hence, binary search is efficient and faster-searching algorithm but can be used only for searching from a sorted array.

How does Binary Search work?

The basic idea behind binary search is that instead of comparing the required element with all the elements of the array, we will compare the required element with the middle element of the array. If this turns out to be the element we are looking for, we are done with the search successfully. Else, if the element we are looking for is less than the middle element, it is sure that the element lies in the first or left half of the array, since the array is sorted. Similarly, if the element we are looking for is greater than the middle element, it is sure that the element lies in the second half of the array.

Thus, Binary search continuously reduces the array into half. The above process is recursively applied on the selected half of the array until we find the element we are looking for.

We will start searching with the left index 0 and right index equal to the last index of the array. The middle element index (mid)is calculated which is the sum of the left and right index divided by 2. If the required element is less than the middle element, then the right index is changed to mid-1, which means we will now be looking at the first half of the array only. Likewise, if the required element is greater than the middle element, then the left index is changed to mid+1, which means we will now be looking at the second half of the array only. We will repeat the above process for the selected array half.

How do we know if element is not present in the array?

We need to have some condition to stop searching further which will indicate that the element is not present in the array. We will iteratively search for the element in the array as long as the left index is less than or equal to the right index. Once this condition turns false and we haven’t found the element yet, this means that the element is not present in the array.

Example

Let us take the following sorted array and we need to search element 6.

2	5	6	8	10	11	13	15	16

L=0 H=8 Mid=4

2	5	6	8	10	11	13	15	16

6<10, therefore take the first half.

H=Mid-1

L=0 H=3 Mid=1

2	5	6	8	10	11	13	15	16

6>5, therefore choose the second half.

L=Mid+1

L=2 H=3 Mid=2

2	5	6	8	10	11	13	15	16

6==6, an element found

Hence the element 6 is found at index 2.

Implementation

From a given sorted array, search for a required element and print its index if the element is present in the array. If the element is not present, print -1.

The code for the implementation of binary search is given below.

Example

```
def binary_search(arr,x):
  l=0
  r=len(arr)-1
  while(l<=r):
    mid=(l+r)//2
    if(arr[mid]==x):
      return mid
    elif(x<arr[mid]):
      r=mid-1
    elif(x>arr[mid]):
      l=mid+1
  return -1
array=[1,2,3,4,5,6,7,8,9,10]
a=7
print(binary_search(array,a))
b=15
print(binary_search(array,b))
```

Output

```
6
-1
```

Element 7 is present at index 6.

Element 15 is not present in the array, hence -1 is printed.