Binary Search (Recursive)

Problem: Given a sorted array **arr[]** of **n** elements, write a function to search a given element **x** in **arr[]** and return the index of x in the array.

Consider array is 0 base index.

Examples:

> ***Input:*** *arr[] = {10, 20, 30, 50, 60, 80, 110, 130, 140, 170}, x = 110*
> ***Output:*** *6*
> ***Explanation:*** *Element x is present at index 6.*
>
> ***Input:*** *arr[] = {10, 20, 30, 40, 60, 110, 120, 130, 170}, x = 175*
> ***Output:*** *-1*
> ***Explanation:*** *Element x is not present in arr[].*

Binary Search Approach:

> ***Binary Search*** *is a searching algorithm used in a sorted array by* ***repeatedly dividing the search interval in half****. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).*

Binary Search Algorithm: The basic steps to perform Binary Search are:

- Begin with the mid element of the whole array as a search key.
- If the value of the search key is equal to the item then return an index of the search key.
- Or 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 from the second point until the value is found or the interval is empty.

```
            high = mid - 1
```

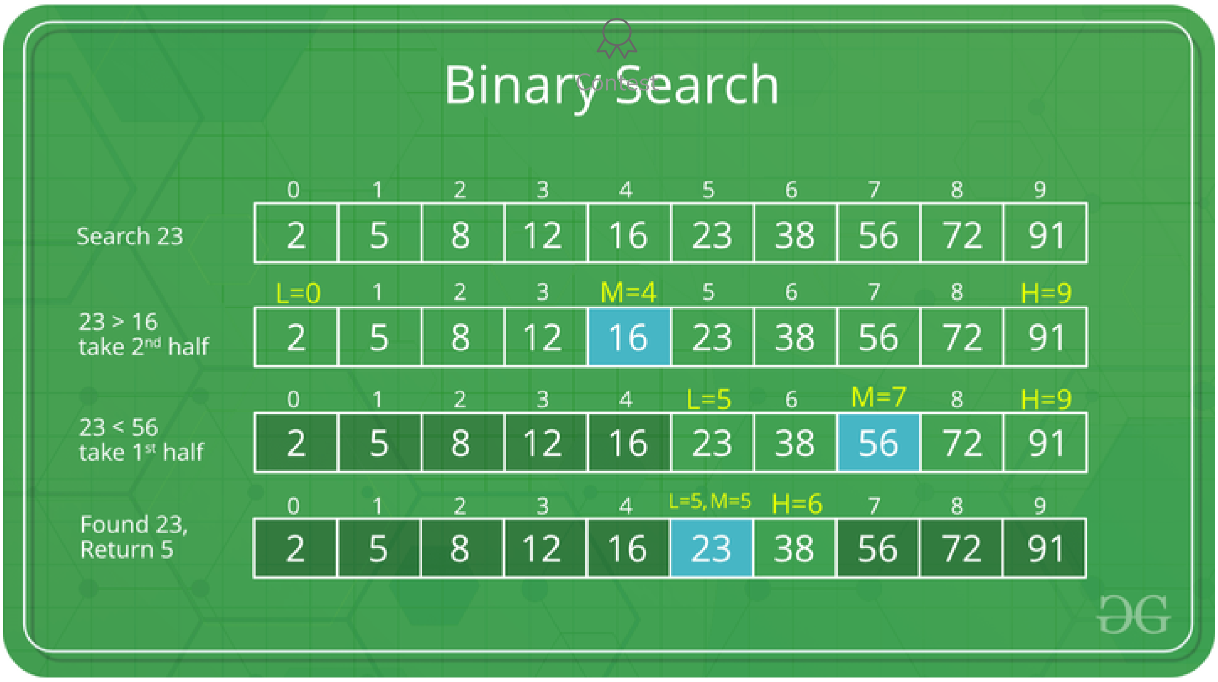
```
    binarySearch(arr, x, low, high)

            else
                mid = (low + high) / 2
                    if x == arr[mid]
                    return mid

                else if x > arr[mid]        // x is on the right side
                    return binarySearch(arr, x, mid + 1, high)

                else                        // x is on the left side
                    return binarySearch(arr, x, low, mid - 1)
```


Illustration of Binary Search Algorithm:



Example of Binary Search Algorithm

Step-by-step Binary Search Algorithm: We basically ignore half of the elements just after one comparison.

1. Compare x with the middle element.
2. If x matches with the middle element, we return the mid index.
3. Else If x is greater than the mid 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.

Dash

C++ Java

All

Articles

Videos

Problems

Quiz

Contest

```
#include <bits/stdc++.h>
using namespace std;

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is present,
// otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l) {
        int mid = l + (r - l) / 2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then
        // it can only be present in left subarray
        if (arr[mid] > x)
            return binarySearch(arr, l, mid - 1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid + 1, r, x);
    }

    // We reach here when element is not
    // present in array
    return -1;
}

int main(void)
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int n = sizeof(arr) / sizeof(arr[0]);
    int result = binarySearch(arr, 0, n - 1, x);
```

Menu

Track Progress

4 of **60** Complete. (7%)

Dash

```
: cout << "Element is present at index " << result;
```

All

Articles

Output

```
Element is present at index 3
```

Videos

Time Complexity: O(log n)
Auxiliary Space: O(log n)

Problems

Quiz

Marked as Read

Contest

Report An Issue

If you are facing any issue on this page. Please let us know.

Menu

Track Progress

4 of **60** Complete. (7%)