## Binary search

Binary search is an efficient algorithm for finding an item from a sorted list of items.

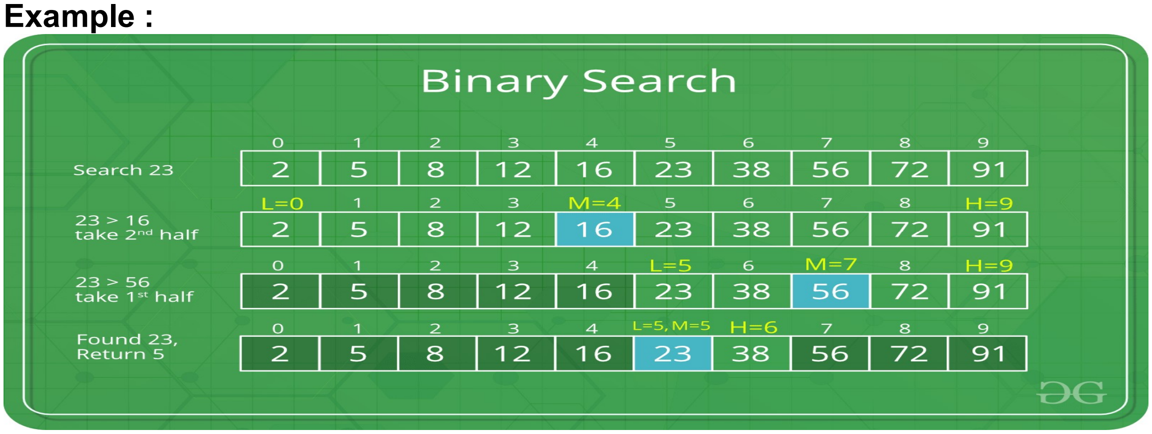
- It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one.
- We used binary search in the guessing game in the introductory tutorial.
- One of the most common ways to use binary search is to find an item in an array.
- For example, the Tycho-2 star catalog contains information about the brightest 2,539,913 stars in our galaxy.
- Suppose that you want to search the catalog for a particular star, based on the star's name.
- If the program examined every star in the star catalog in order starting with the first, an algorithm called **linear search**, the computer might have to examine all 2,539,913 stars to find the star you were looking for, in the worst case.
- If the catalog were sorted alphabetically by star names, **binary search** would not have to examine more than 22 stars, even in the worst case.

### Binary Search Approach

- Given a sorted array arr[] of n elements, write a function to search a given element x in arr[].
- A simple approach is to do **linear search.**
- The time complexity of above algorithm is O(n).
- Another approach to perform the same task is using Binary Search.

### Binary Search:

- 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.
- **Example :**



- The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

**Algorithm**

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

Step-1: Compare x with the middle element.

Step-2: If x matches with middle element, we return the mid index.

Step-3: Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.

Step-4: Else (x is smaller) recur for the left half.

**Recursive** implementation of Binary Search

```
// C++ program to implement recursive Binary Search
#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);
    (result == -1) ? cout << "Element is not present in array"
                   : cout << "Element is present at index " << result;
    return 0;
}
```

**Output :**
Element is present at index 3

**Iterative** implementation of Binary Search

```
// C++ program to implement recursive Binary Search
#include <bits/stdc++.h>
using namespace std;

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

        // Check if x is present at mid
        if (arr[m] == x)
            return m;

        // If x greater, ignore left half
        if (arr[m] < x)
            l = m + 1;

        // If x is smaller, ignore right half
        else
```

```
        r = m - 1;
    }

    // if we reach here, then element was
    // not present
    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);
    (result == -1) ? cout << "Element is not present in array"
                   : cout << "Element is present at index " << result;
    return 0;
}
```

**Output :**
Element is present at index 3

**Time Complexity:**
The time complexity of Binary Search can be written as

T(n) = T(n/2) + c

The above recurrence can be solved either using Recurrence Tree method or Master method. It falls in case II of Master Method and solution of the recurrence is

**Ɵ(Log n)**

**Auxiliary Space:** O(1) in case of iterative implementation. In case of recursive implementation, O(Logn) recursion call stack space.

- In computer science, binary search, also known as half-interval search, logarithmic search, or binary chop, is a search algorithm that finds the position of a target value within a sorted array.
- Binary search compares the target value to the middle element of the array.

Worst complexity: O(log n)
Average complexity: O(log n)
Best complexity: O(1)
Space complexity: O(1)
Data structure: Array
Class: Search algorithm