**Algorithms**

**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.

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

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

1. Compare x with the middle element.
2. If x matches with middle element, we return the mid index.
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.
4. Else (x is smaller) recur for the left half.

public int RecursiveBinarySearch(int[] arr, int startIndex, int endIndex, int val)

{

if (endIndex >= startIndex)

{

int mid = (startIndex + endIndex) / 2;

// If the element is present at the

// middle itself

if (arr[mid] == val)

{

return mid;

}

// If element is smaller than mid, then

// it can only be present in left subarray

if (arr[mid] > val)

{

return RecursiveBinarySearch(arr, startIndex, mid - 1, val);

}

// Else the element can only be present

// in right subarray

return RecursiveBinarySearch(arr, mid + 1, endIndex, val);

}

// We reach here when element is not present

// in array

return -1;

}

**Jump Search :** Like Binary Search, Jump Search is a searching algorithm for sorted arrays. The basic idea is to check fewer elements (than linear search) by jumping ahead by fixed steps or skipping some elements in place of searching all elements**.**

What is the optimal block size to be skipped?

In the worst case, we have to do n/m jumps and if the last checked value is greater than the element to be searched for, we perform m-1 comparisons more for linear search. Therefore the total number of comparisons in the worst case will be ((n/m) + m-1). The value of the function ((n/m) + m-1) will be minimum when m = √n. Therefore, the best step size is m = √n.