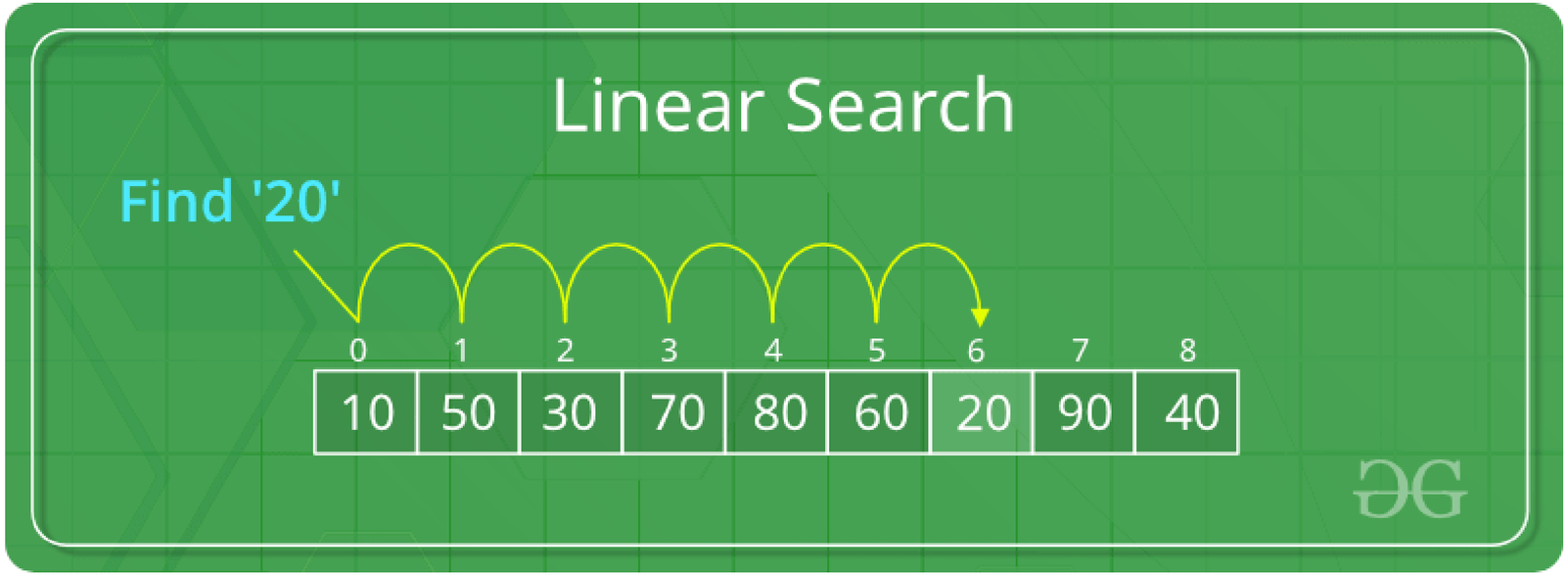
Searching algorithms

Linear Search

it searches the element one by one

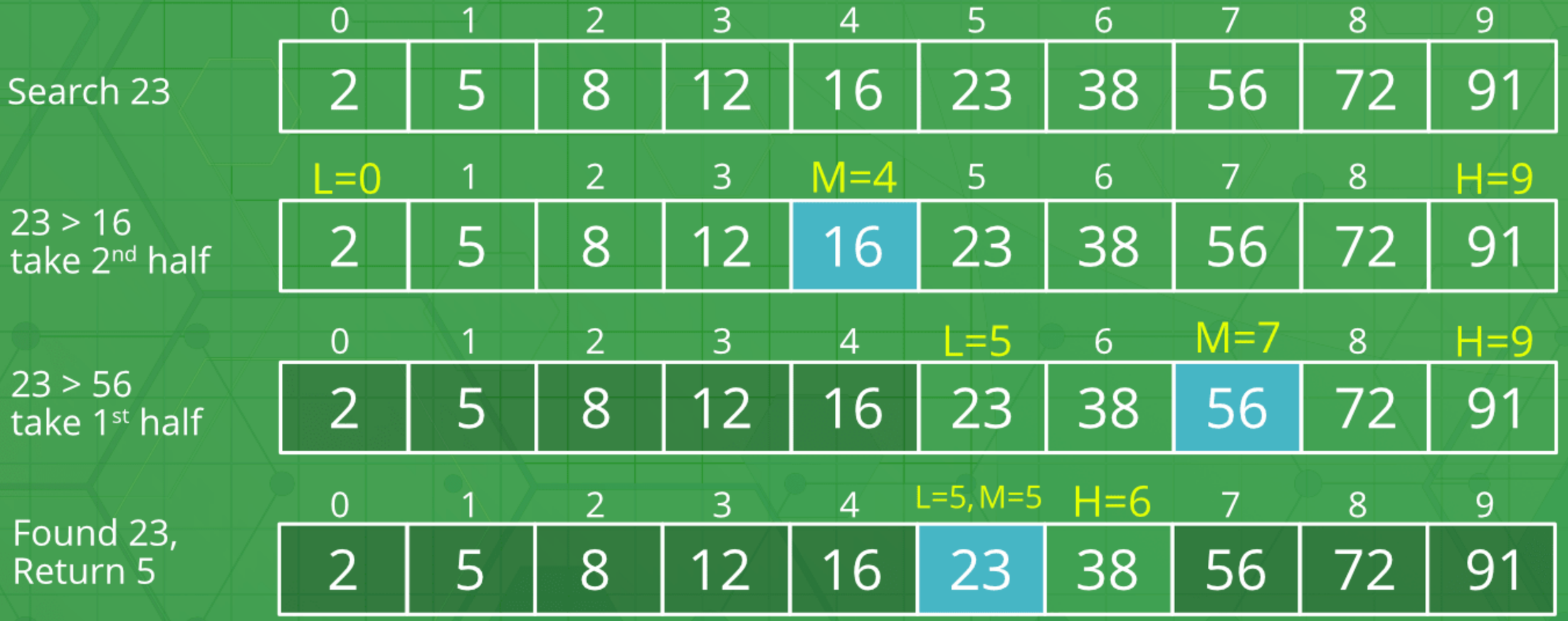
time complexity - O(n)



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).**

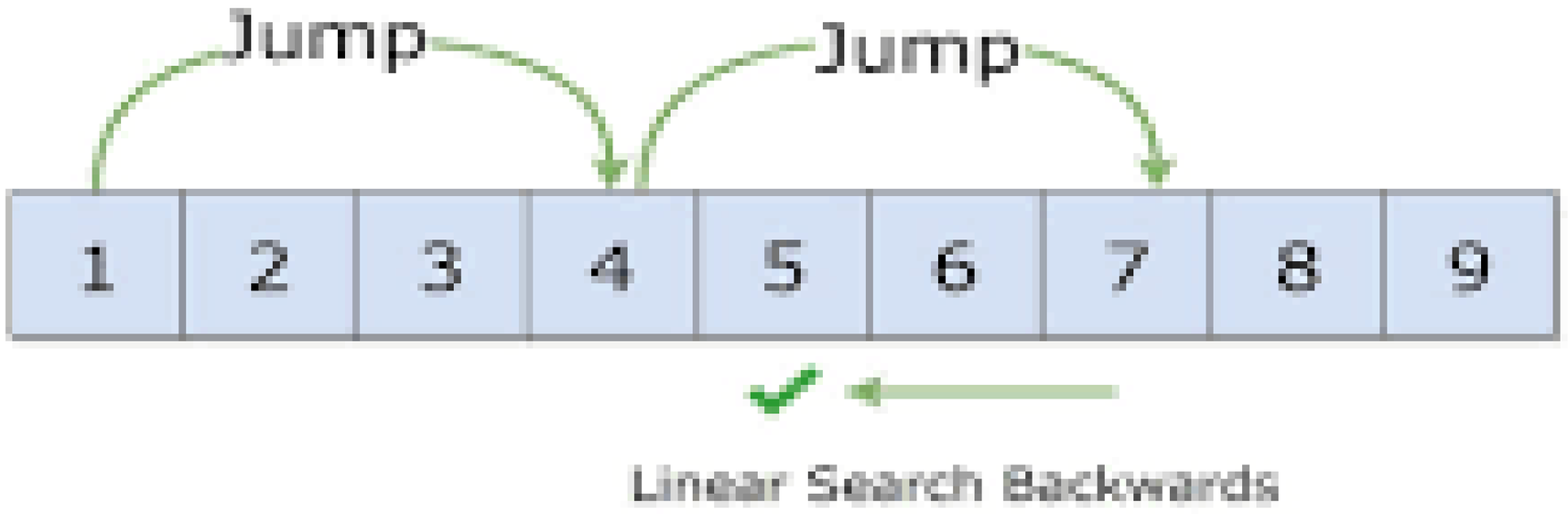
Binary Search



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.**
- **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.**

Jump Search



Interpolation search

- The Interpolation Search is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed. Binary Search always goes to the middle element to check. On the other hand, interpolation search may go to different locations according to the value of the key being searched. For example, if the value of the key is closer to the last element, interpolation search is likely to start search toward the end side.
- pos = lo + [(x-arr[lo])*(hi-lo) / (arr[hi]-arr[Lo])]
- arr[] ==> Array where elements need to be searched
- x ==> Element to be searched
- lo ==> Starting index in arr[]
- hi ==> Ending index in arr[]