Data Structures Lab 4

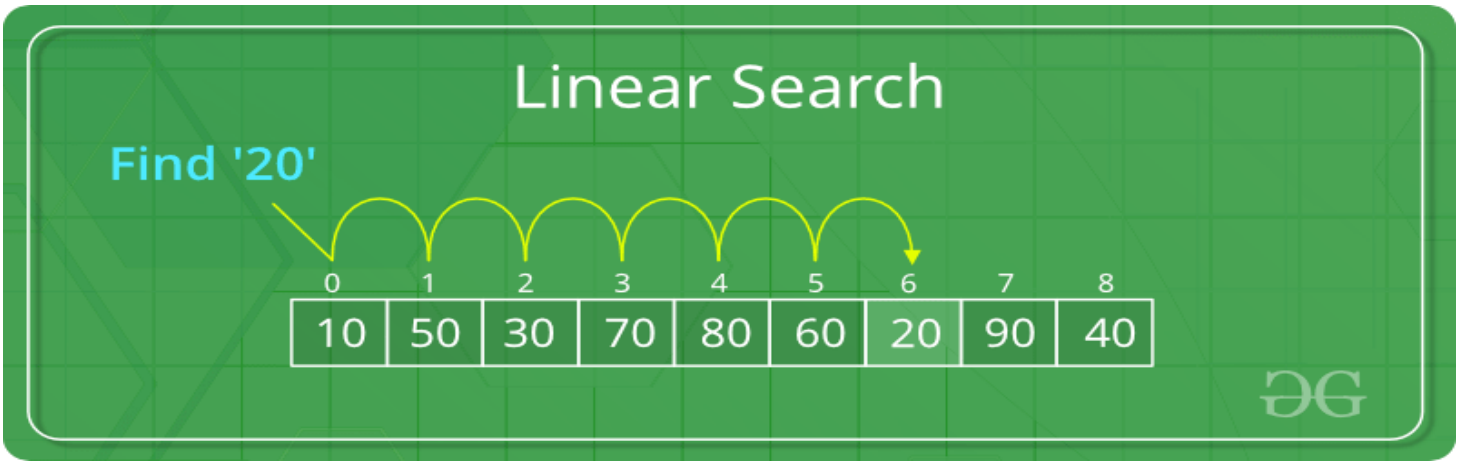
Course: Data Structures (CL2001) **Semester:** Spring 2024
Instructor: Bushra Sattar **T.A:** Avinash

Note:

- Lab manual cover following below Advance searching algorithms
- **{Linear Searching, Binary Searching, Interpolation Search}**
- Maintain discipline during the lab.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Linear Search

Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set.

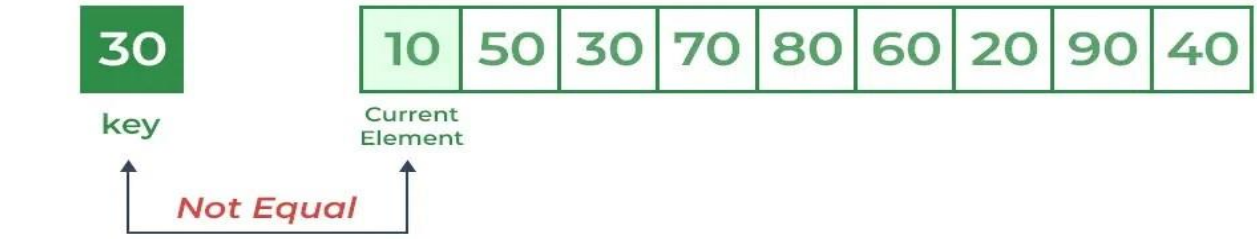


How Does Linear Search Algorithm Work?
In Linear Search Algorithm,

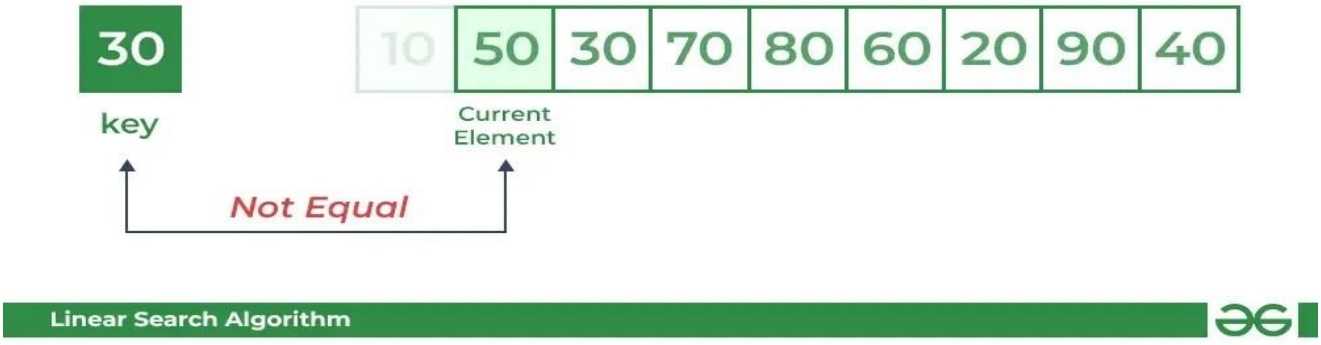
- Every element is considered as a potential match for the key and checked for the same.
- If any element is found equal to the key, the search is successful and the index of that element is returned.
- If no element is found equal to the key, the search yields "No match found".

For example: Consider the array arr[] = {10, 50, 30, 70, 80, 20, 90, 40} and key = 30

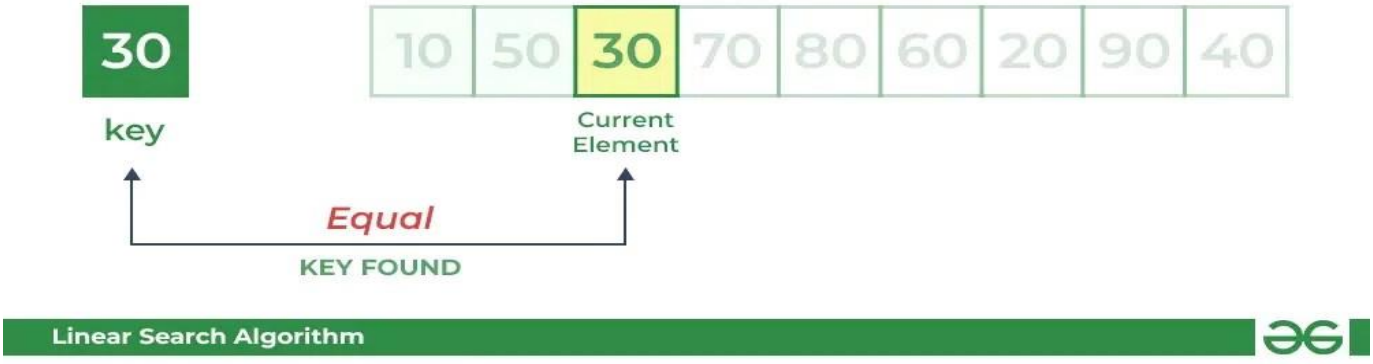
Step 1: Start from the first element (index 0) and compare key with each element (arr[i]).



Comparing key with first element arr[0]. SInce not equal, the iterator moves to the next element as a potential match.



Comparing key with next element arr[1]. SInce not equal, the iterator moves to the next element as a potential match.



Step 2: Now when comparing arr[2] with key, the value matches. So the Linear Search Algorithm will yield a successful message and return the index of the element when key is found (here 2).

Complexity Analysis of Linear Search:

Time Complexity:

Best Case: In the best case, the key might be present at the first index. So the best case complexity is **O(1)**
Worst Case: In the worst case, the key might be present at the last index i.e., opposite to the end from which the search has started in the list. So the worst-case complexity is **O(N)** where **N** is the size of the list.
Average Case: O(N)

Advantages of Linear Search:

- Linear search can be used irrespective of whether the array is sorted or not. It can be used on arrays of any data type.
- Does not require any additional memory.
- It is a well-suited algorithm for small datasets.

Drawbacks of Linear Search:

- Linear search has a time complexity of O(N), which in turn makes it slow for large datasets.
- Not suitable for large arrays.
- When to use Linear Search?
- When we are dealing with a small dataset.
- When you are searching for a dataset stored in contiguous memory.

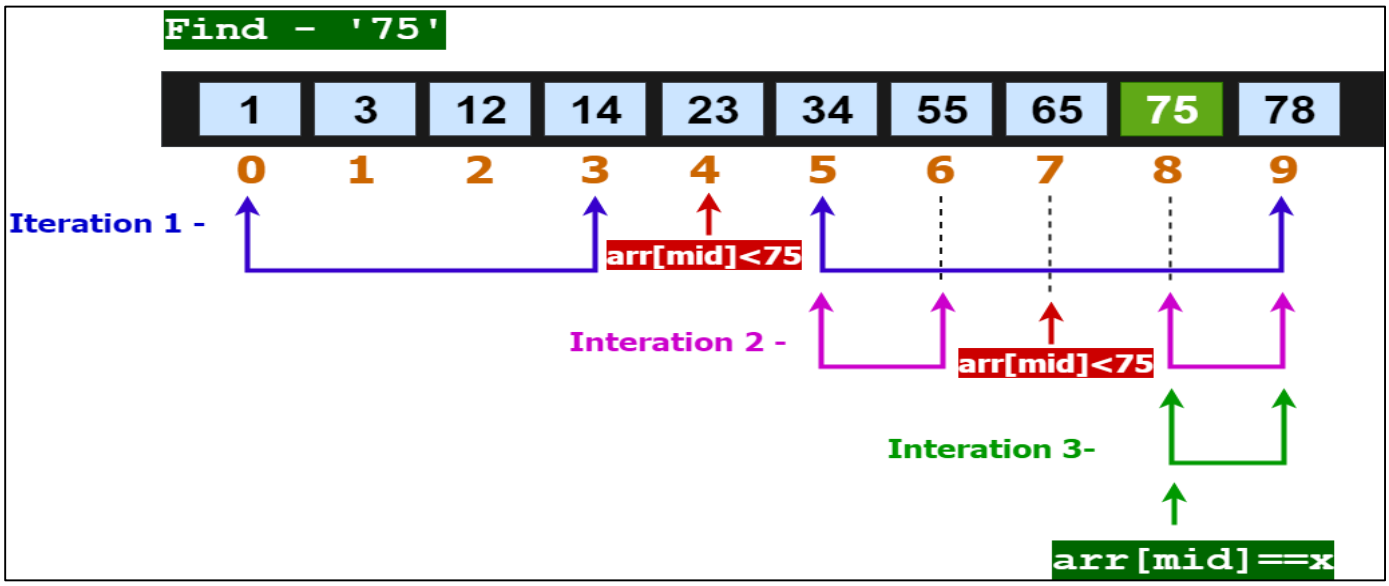
Task 1:
Given an array of integers and a target element, implement a linear search algorithm to find the index of the target element in the array. If the target element is not present, return -1.

<u>Binary Search</u>

Binary search algorithm falls under the category of interval search algorithms. This algorithm is much more efficient compared to linear search algorithm. Binary search only works on sorted data structures. This algorithm repeatedly target the center of the sorted data structure & divide the search space into half till the match is found.
Algorithm:

- ***Take input array, left, right & x***
- ***START LOOP – while(left greater than or equal to right)***
  - ***mid = left + (right-left)/2***
  - ***if(arr[mid]==x) then***
    - ***return m***
  - ***else if(arr[mid] less than x) then***
    - ***left = m + 1***
  - ***else***
    - ***right= mid – 1***

- *END LOOP*
- *return -1*



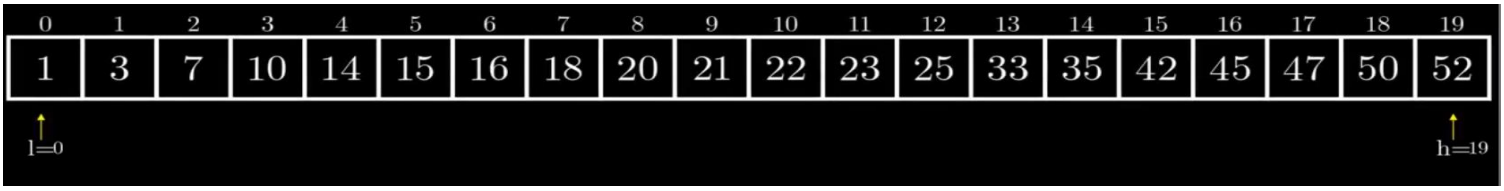
Task 2:
Given a sorted array of size N and an integer K, find the position (0-based indexing) at which K is present in the array using binary search.

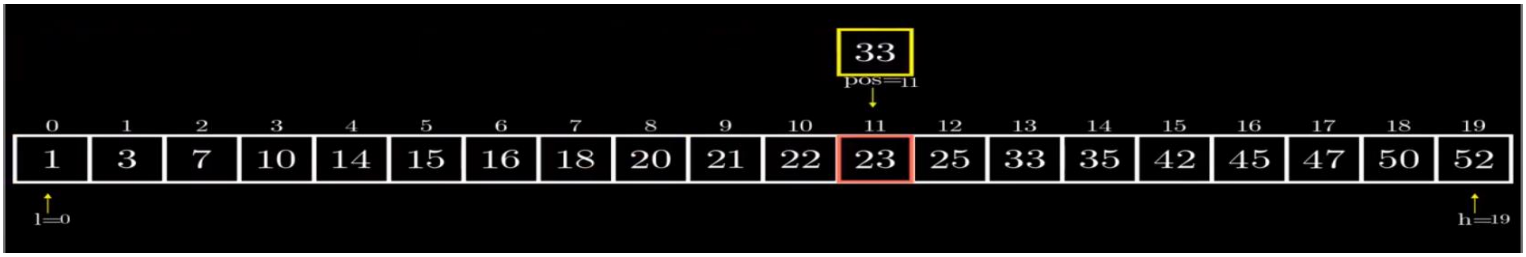
Example 1:
Input:
N = 5
arr[] = {1 2 3 4 5}
K = 4
Output: 3
Explanation: 4 appears at index 3.

Example 2:
Input:
N = 5
arr[] = {11 22 33 44 55}
K = 445
Output: -1

Interpolation Search
Interpolation search is an improvement over binary search. Binary Search always checks the value at middle index. But, interpolation search may check at different locations based on the value of element being searched. For interpolation search to work efficiently the array elements/data should be sorted and uniformly distributed.
Search key =33

$$Position = startindex + \frac{(element - Arr[startindex]) * (end\ value\ index - start\ value\ index)}{Arr[endindex] - Arr[startindex]}$$



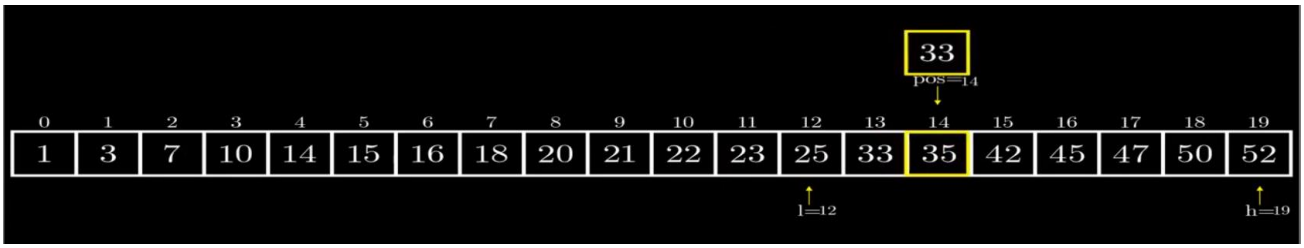


$$pos = 0 + \frac{(33-1)*(19-0)}{(52-1)} = 11$$

Total Comparisons: 0

33 >23 Set low = 12

Total Comparisons: 1

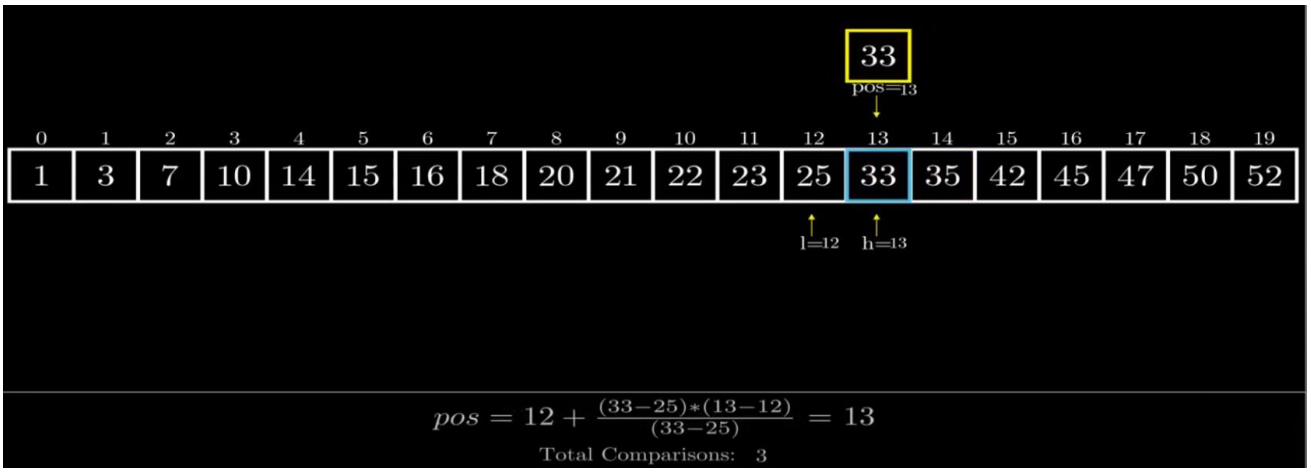


$$pos = 12 + \frac{(33-25)*(19-12)}{(52-25)} = 14$$

Total Comparisons: 1

33 <35 Set high = 13

Total Comparisons: 2



Algorithm

1. **start = 0 & end = n-1**
2. **calculate position to start searching at using formula:**

$$Position = startindex + \frac{(element - Arr[startindex]) * (end\ value\ index - start\ value\ index)}{Arr[endindex] - Arr[startindex]}$$

3. **If A [pos] == Element, element found at index pos.**
4. **Otherwise if element > A[pos] we make start = pos + 1**
5. **Else if element < A[pos] we make end = pos -1**
6. **Do steps 2,3, 4, 5, While : start <= end && element >= A[start] && element =< A[end]**
   - **Start <= end - that is until we have elements in the sub-array.**
   - **Element >= A[start] - element we are looking for is greater than or equal to the starting element of sub-array we are looking in.**
   - **Element =< A[end] - element we are looking for is less than or equal to the last element of sub-array we are looking in.**

Tasks 3:
Write code for interpolation and Binary Search and show the iteration of both algorithm, for uniformly distributed array.
Note: An array is considered as uniformly distributed when the difference between the elements is equal or almost same. Example 1: 1,2,3,4,5,6 (Difference is 1)

Task 4:
Given an integer array and another integer element. The task is to find if the given element is present in array or not.
Example 1:
Arr={14,6,18,23,4,67,48,78,3,2,74,76,8}
Key=74

Example 1:
Arr={4,76,188,693,884,367,8.2.102,3,7}
Key=94

Tasks 5:

a. Write a Java function that uses linear search to find and return the maximum element in an array of integers.
b. Write a Java function that uses binary search to find and return the minimum element in an array of integers.

Tasks 6:

Write a C++/Java function that conducts a linear search to identify and return the index of the last occurrence of a specified target element in an array. Additionally, the function should count and return the total number of occurrences of the target element in the array. If the target element is not found, the function should return -1

Tasks 7:

Given a sorted array of distinct positive integers, each representing a unique positive value, there exists a special number in the array that is a multiple of all other elements. Your task is to implement an interpolation search algorithm in C++ and Java to efficiently find this special number.

Task 8:

Write a Java program that compares the two algorithms, interpolation search with binary search on a large dataset. Your program should include the following:

- **Data Generation:**
  Generate a large sorted array of integers (e.g., 100 elements) with a known pattern. You can use random numbers or follow a specific pattern.

- **Function Signatures:**
  Implement two search functions:
  public static int interpolationSearch(int[] arr, int target)
  public static int binarySearch(int[] arr, int target)

- **Input:**
  Both functions should take two parameters:
  An array of sorted integers, arr.
  An integer, target, representing the value to be searched.

- **Output:**
  Both functions should return the index of the target element in the array arr. If the element is not present, return -1.