LINEAR, BINARY & INTERPOLATION SEARCH USING ARRAYS AND LINKED LISTS

WHAT ARE ALGORITHMS

An algorithm is a process or a set of rules required to perform calculations or some other problem-solving operations especially by a computer. The formal definition of an algorithm is that it contains the finite set of instructions which are being carried in a specific order to perform the specific task. It is not the complete program or code; it is just a solution (logic) of a problem, which can be represented either as an informal description using a Flowchart or Pseudocode.

EXAMPLE

The following are the steps required to add two numbers entered by the user:

Step 1: Start

Step 2: Declare three variables a, b, and sum.

Step 3: Enter the values of a and b.

Step 4: Add the values of a and b and store the result in the sum variable, i.e., sum=a+b.

Step 5: Print sum

Step 6: Stop

TYPES OF ALGORITHMS

- **Sort:** Algorithm developed for sorting the items in a certain order.
- **Search:** Algorithm developed for searching the items inside a data structure.

ALGORITHM COMPLEXITY

The performance of the algorithm can be measured in two factors:

- **Time complexity:** The time complexity of an algorithm is the amount of time required to complete the execution. The time complexity of an algorithm is denoted by the big O notation. Here, big O notation is the asymptotic notation to represent the time complexity. The time complexity is mainly calculated by counting the number of steps to finish the execution. Let's understand the time complexity through an example.

```
1. sum=0;
2. // Suppose we have to calculate the sum of n numbers.
3. for i=1 to n
4. sum=sum+i;
5. // when the loop ends then sum holds the sum of the n numbers
6. return sum;
```

In the above code, the time complexity of the loop statement will be atleast n, and if the value of n increases, then the time complexity also increases. While the complexity of the code, i.e., return sum will be constant as its value is not dependent on the value of n and will provide the result in one step only. We generally consider the worst-time complexity as it is the maximum time taken for any given input size.

- **Space complexity:** An algorithm's space complexity is the amount of space required to solve a problem and produce an output. Similar to the time complexity, space complexity is also expressed in big O notation.

For an algorithm, the space is required for the following purposes:

1. To store program instructions
2. To store constant values
3. To store variable values
4. To track the function calls, jumping statements, etc.

Auxiliary space: The extra space required by the algorithm, excluding the input size, is known as an auxiliary space. The space complexity considers both the spaces, i.e., auxiliary space, and space used by the input.

So,

Space complexity = Auxiliary space + Input size

LINEAR SEARCH

Linear search is also called as **sequential search algorithm.** It is the simplest searching algorithm. In Linear search, we simply traverse the list completely and match each element of the list with the item whose location is to be found. If the match is found, then the location of the item is returned; otherwise, the algorithm returns NULL.

TIME COMPLEXITY: The worst-case time complexity of linear search is O(n).

ALGORITHM

```
Linear_Search(a, n, val) // 'a' is the given array, 'n' is the size of given array, 'val' is the value to search
Step 1: set pos = -1
Step 2: set i = 1
Step 3: repeat step 4 while i <= n
Step 4: if a[i] == val
set pos = i
print pos
go to step 6
[end of if]
set ii = i + 1
[end of loop]
Step 5: if pos = -1
print "value is not present in the array "
[end of if]
Step 6: exit
```

BINARY SEARCH

Binary search is the search technique that works efficiently on sorted lists. Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.

Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list. If the match is found

then, the location of the middle element is returned. Otherwise, we search into either of the halves depending upon the result produced through the match.

Binary Search Algorithm Steps:

1. Start with the entire list as the search interval.
2. Find the middle element of the interval.
3. If the middle element is equal to the target value, return the index of the middle element.
4. If the target value is less than the middle element, narrow the search to the lower half of the interval.
5. If the target value is greater than the middle element, narrow the search to the upper half of the interval.
6. Repeat steps 2-5 until the target value is found or the interval is empty.

Time Complexity: O(log n)

INTERPOLATION SEARCH

Interpolation Search is a searching algorithm that uses an interpolation formula to estimate the position of the target value in a sorted array or list.

Unlike binary search, which always selects the middle element, Interpolation Search makes a more intelligent guess based on the distribution of the data. It uses a formulaic approach to determine the position of the target element within the array.

It is particularly effective when the elements are uniformly distributed.

The uniform distribution of the dataset means the interval between the elements should be uniform (does not have a large difference).

HOW INTERPOLATION SEARCH WORKS:

It uses the idea of the interpolation formula to estimate the probable location of the target element.

It calculates the probable position using an interpolation formula that considers the range and values of the data elements.

Interpolation Formula = low + [(high - low) * (X - A[low])]/(A[high] - A[low])

low - Left pointer

high - Right pointer

A[low] - Element at the left pointer

A[high] - Element at the right pointer

X - Target element to be searched

This estimation guides the algorithm to narrow down the search range and thus achieve faster retrieval.

ALGORITHM:

The algorithm can be summarized in the following steps:

1. Initialize low and high indices to the start and end of the array, respectively.
2. Calculate the probe position using the interpolation formula.
3. Compare the probe element with the target element.
    - If they are equal, the search is successful.
    - If the probe element is greater, update the high index to the probe position minus one.
    - If the probe element is smaller, update the low index to the probe position plus one.
4. Repeat steps 2-3 until the target element is found or the low index exceeds the high index.

TIME COMPLEXITY ANALYSIS:

Average Case: O(log logn) - When the data is uniformly distributed.

Worst Case: O(n) - When the data is not uniform, making it less efficient than the binary search.