Search Algorithms

What are Search Algorithms?

Algorithms used to retrieve an element from any data structure.

Types:

- Linear Search
- Binary Search
- Jump Search
- Interpolation Search
- Exponential Search

Linear Search – Concept

A linear search is also called a sequential search. This is a very simple technique used to search for a particular value in an array. A linear search works by comparing the value of the key being searched for with every element of the array in a linear sequence until a match is found.

A search will be unsuccessful if all the data elements are read and the desired element is not found. The following are some important points:

- It is the simplest way to search an element in a list.
- It searches the data element sequentially, no matter whether the array is sorted or unsorted.

```
Step 1: START
Step 2: Set I = 0, POS = -1
Step 3: Repeat while I<N
            IF (ARR[I] = VAL)
            POS = I
        PRINT POS
                Go to Step 5
            [End of IF]
        [End of Loop]
Step 4: IF (POS = -1)
        PRINT "VALUE NOT FOUND, SEARCH UNSUCCESSFUL"
        [End of IF]
Step 5: EXIT
```

Complexity of a Linear Search Algorithm

The execution time of a linear search is O(n),where n is the number of elements in the array. The algorithm is called a linear search because its complexity can be expressed as a linear function, which is that the number of comparisons to find the target item increases linearly with the size of the data. The best case of a linear search is when the data element to be searched for is equal to the first element of the array. The worst case will happen when the data element to be searched for is equal to the last element in the array.

Drawbacks of a Linear Search

It is a very time-consuming process, as it works sequentially.

It can be applied only to a small amount of data.

It is a very slow process, as almost every data element is accessed, especially when the data element is located near the end.

Case	Complexity	Description
Best	**O(1)**	Target is at the first index
Average	**O(n)**	Target is somewhere in the middle
Worst	**O(n)**	Target is at the last index or not present

Time Complexity

Binary Search

A binary search is an extremely efficient searching algorithm when it is compared to a linear search. A binary search works only when the array/list is already sorted.

In a binary search, we first compare the value VAL with the data element in the middle position of the array. If the match is found, then the position POS of that element is returned; otherwise, if the value is less than that of the middle element, then we begin our search in the lower half of the array and vice versa. So, we repeat this process on the lower and upper half of the array.

Binary Search Algorithm

1. Find the middle element of the array, that is, n/2 is the middle element of the array containing n elements.

2. Now, compare the middle element of the array with the data element to be searched.

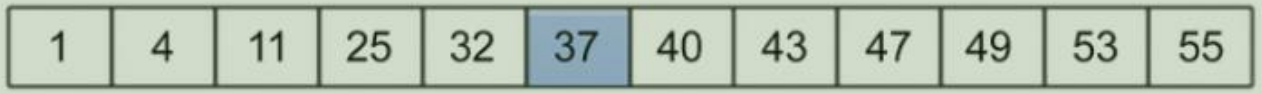
(a) If the middle element is the desired element, then the search is successful.

(b) If the data element to be searched for is less than the middle element of the array, then search only the lower half of the array, that is, those elements which are on the left side of the middle element.

(c) If the data element to be searched for is greater than the middle element of the array, then search only the upper half of the array, that is, those elements which are on the right side of the middle element.

```
Step 1: START
Step 2: Set BEG = lower_bound, END = upper_bound, POS = -1
Step 3: Repeat Steps 4 & 5 while BEG <= END
Step 4: Set MID = (BEG+END)/2
Step 5: IF (ARR[MID] = VAL)
        POS = MID
        PRINT POS
        Go to Step 7
        ELSE IF (ARR[MID] > VAL)
        Set END = MID - 1
        ELSE
        Set BEG = MID + 1
        [End of If]
        [End of Loop]
Step 6: IF (POS = -1)
              PRINT "VALUE NOT FOUND, SEARCH UNSUCCESSFUL"
        [End of IF]
Step 7: EXIT
```

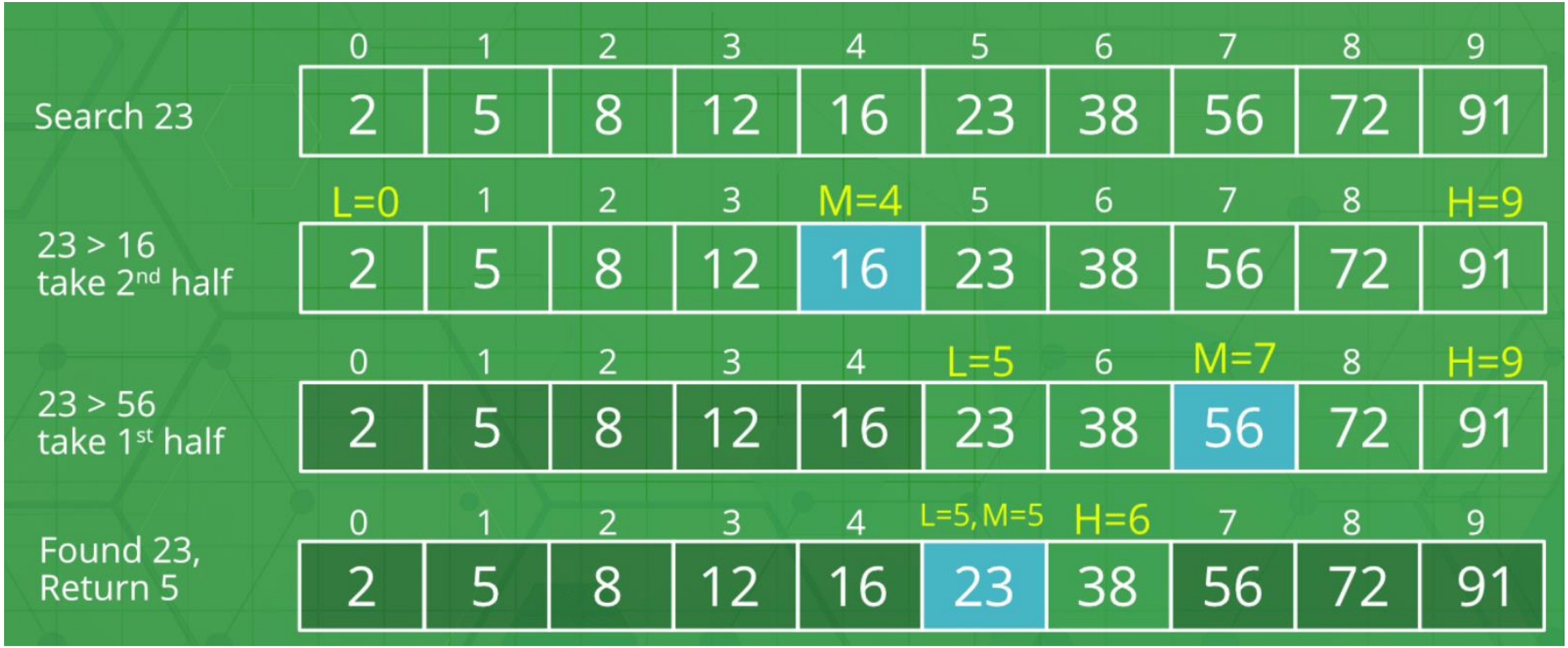
If we want to search 43 in the given list



since 43>37
We look only at the second half

since 43>37
We now look only at the first half of the list

Search item 43 is found.
The function will return the index position


Search 23
0 1 2 3 4 5 6 7 8 9
2 5 8 12 16 23 38 56 72 91
23 > 16
take 2nd half
L=0 1 2 3 M=4 5 6 7 8 H=9
2 5 8 12 16 23 38 56 72 91
23 > 56
take 1st half
0 1 2 3 4 L=5 6 M=7 8 H=9
2 5 8 12 16 23 38 56 72 91
Found 23,
Return 5
0 1 2 3 4 L=5, M=5 H=6 7 8 9
2 5 8 12 16 23 38 56 72 91

Complexity of a Binary Search Algorithm

In a binary search algorithm, the size of the search area is reduced by half. So, we can claim that the efficiency of the binary search in the worst case is O(log2n), where n is the total number of elements in the array. Obviously, the best case will happen when the value to be searched for is equal to the value of the array in the middle.

Time Complexity

Case	Complexity	Description
Best	**O(1)**	Target is found at the middle index on the first try
Average	**O(log n)**	The array is divided in half each time
Worst	**O(log n)**	Target is at the end or not present

Drawbacks of a Binary Search

A binary search requires that the data elements in the array be sorted; otherwise, a binary search will not work.

A binary search cannot be used where there are many insertions and deletions of data elements in the array.

Interpolation search

An interpolation search, also known as an extrapolation search, is a technique for searching for a particular value in an ordered array. This searching technique is more efficient than a binary search if the elements in the array are sorted.

The interpolation search algorithm works efficiently when there are uniformly distributed elements in the sorted list. In a binary search, we always start searching from the middle of the list, whereas in the interpolation search we compute the starting search position depending on the item to be searched. In the interpolation search algorithm, the starting search position is most likely to be close to the start or end of the list; if the search item is near the first element in the list, then the starting search position is likely to be near the start of the list and if the search item is near the end of the list, then the starting search position is likely to be near the end of the list.

Pros and Cons

Advantages:

Faster than binary search for uniformly distributed data.

Reduces number of probes in such cases.

Disadvantages:

Performance degrades to **O(n)** for skewed distributions.

Requires sorted data and non-zero value differences.

When to Use Interpolation Search

Data is **sorted**.

Data is **uniformly distributed**.

You need faster search than binary in such datasets.

Jump search

The jump search algorithm is an improvement over linear search for searching for a given element from an ordered (or sorted) list of elements. This uses the divide-and-conquer strategy in order to search for the required element. In linear search, we compare the search value with each element of the list, whereas in jump search, we compare the search value at different intervals in the list, which reduces the number of comparisons.

Case	Complexity
Best Case	O(1)
Average	O(√n)
Worst	O(√n)
Space	O(1)

Time and Space Complexity

Exponential search

Exponential search is mostly used when we have large numbers of elements in a list.

Exponential search is also known as galloping search and doubling search.

The exponential search algorithm works in the following two steps:

1. Given a sorted array of n data elements, we first determine the subrange in the original list where the desired search item may be present

2. Next, we use the binary search algorithm to find out the search value within the subrange of data elements identified in step 1

Case	Time	Notes
Best	O(1)	If the target is at index 0
Average	O(log n)	Same as binary search
Worst	O(log n)	Same as binary search
Space	O(1)	Iterative approach

Time and Space Complexity