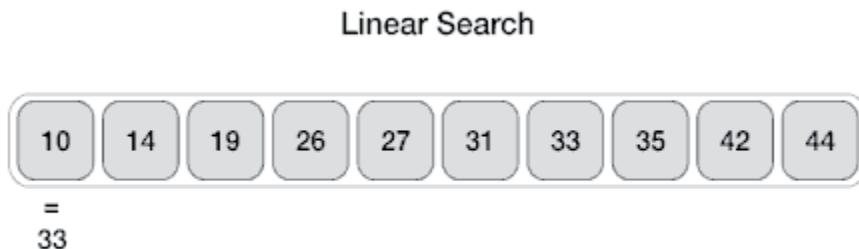


Algorithms

Search Techniques

Linear Search

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.



Algorithm

```
Linear Search ( Array A, Value x)

Step 1: Set i to 1
Step 2: if i > n then go to step 7
Step 3: if A[i] = x then go to step 6
Step 4: Set i to i + 1
Step 5: Go to Step 2
Step 6: Print Element x Found at index i and go to step 8
Step 7: Print element not found
Step 8: Exit
```

Pseudocode

```
procedure linear_search (list, value)

    for each item in the list
        if match item == value
            return the item's location
        end if
    end for

end procedure
```

Binary Search

Binary search is a fast search algorithm with run-time complexity of $O(\log n)$. This search algorithm works on the principle of divide and conquer. For this algorithm to work properly, the data collection should be in the sorted form.

Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned. If the middle item is greater than the item, then the item is searched in the sub-array to the left of the middle item. Otherwise, the item is searched for in the sub-array to the right of the middle item. This process continues on the subarray as well until the size of the subarray reduces to zero.

How Binary Search Works?

For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.

10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

First, we shall determine half of the array by using this formula –

$$\text{mid} = \text{low} + (\text{high} - \text{low}) / 2$$

Here it is, $0 + (9 - 0) / 2 = 4$ (integer value of 4.5). So, 4 is the mid of the array.



10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.

10	14	19	26	27	31	33	35	42	44
0	1	2	3	4	5	6	7	8	9

We change our low to $\text{mid} + 1$ and find the new mid value again.

```
low = mid + 1  
mid = low + (high - low) / 2
```

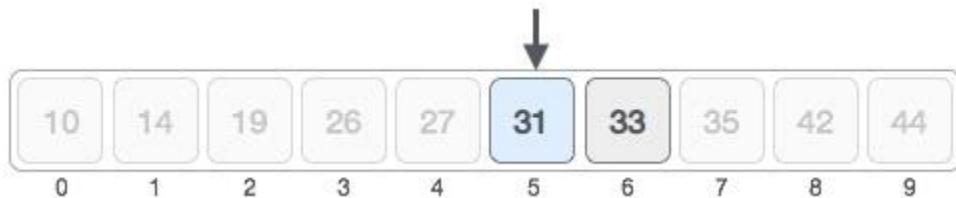
Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.



The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.



Hence, we calculate the mid again. This time it is 5.



We compare the value stored at location 5 with our target value. We find that it is a match.



We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

Pseudocode

The pseudocode of binary search algorithms should look like this –

```
Procedure binary_search
    A ← sorted array
    n ← size of array
    x ← value to be searched

    Set lowerBound = 1
    Set upperBound = n

    while x not found
        if upperBound < lowerBound
            EXIT: x does not exists.

        set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

        if A[midPoint] < x
            set lowerBound = midPoint + 1

        if A[midPoint] > x
            set upperBound = midPoint - 1

        if A[midPoint] = x
            EXIT: x found at location midPoint
    end while

end procedure
```

Interpolation Search

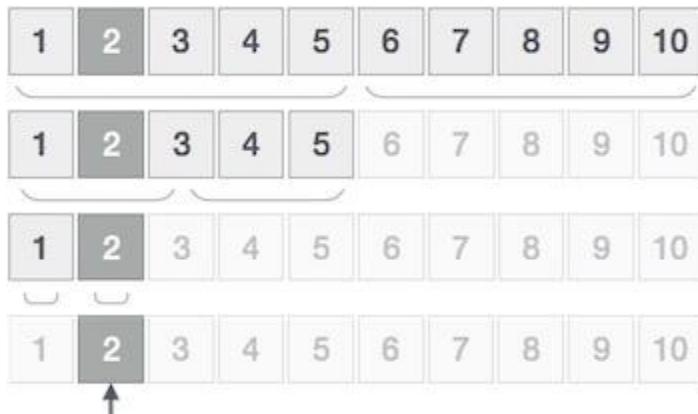
Interpolation search is an improved variant of binary search. This search algorithm works on the probing position of the required value. For this algorithm to work properly, the data collection should be in a sorted form and equally distributed.

Binary search has a huge advantage of time complexity over linear search. Linear search has worst-case complexity of $O(n)$ whereas binary search has $O(\log n)$.

There are cases where the location of target data may be known in advance. For example, in case of a telephone directory, if we want to search the telephone number of Morphius. Here, linear search and even binary search will seem slow as we can directly jump to memory space where the names start from 'M' are stored.

Positioning in Binary Search

In binary search, if the desired data is not found then the rest of the list is divided in two parts, lower and higher. The search is carried out in either of them.



Even when the data is sorted, binary search does not take advantage to probe the position of the desired data.

Position Probing in Interpolation Search

Interpolation search finds a particular item by computing the probe position. Initially, the probe position is the position of the middle most item of the collection.



If a match occurs, then the index of the item is returned. To split the list into two parts, we use the following method –

```
mid = Lo + ((Hi - Lo) / (A[Hi] - A[Lo])) * (X - A[Lo])
```

where –

A = list
Lo = Lowest index of the list
Hi = Highest index of the list
A[n] = Value stored at index n in the list

If the middle item is greater than the item, then the probe position is again calculated in the sub-array to the right of the middle item. Otherwise, the item is searched in the subarray to the left of the middle item. This process continues on the sub-array as well until the size of subarray reduces to zero.

Runtime complexity of interpolation search algorithm is **O(log (log n))** as compared to **O(log n)** of BST in favorable situations.

Algorithm

As it is an improvisation of the existing BST algorithm, we are mentioning the steps to search the 'target' data value index, using position probing –

Step 1 – Start searching **data** from middle of the list.
Step 2 – If it is a match, return the index of the item, and exit.
Step 3 – If it is not a match, probe position.
Step 4 – Divide the list using probing formula and find the new midle.
Step 5 – If data is greater than middle, search in higher sub-list.
Step 6 – If data is smaller than middle, search in lower sub-list.
Step 7 – Repeat until match.

Pseudocode

```
A → Array list
N → Size of A
X → Target Value

Procedure Interpolation_Search()

    Set Lo → 0
    Set Mid → -1
    Set Hi → N-1

    While X does not match

        if Lo equals to Hi OR A[Lo] equals to A[Hi]
            EXIT: Failure, Target not found
        end if

        Set Mid = Lo + ((Hi - Lo) / (A[Hi] - A[Lo])) * (X - A[Lo])

        if A[Mid] = X
            EXIT: Success, Target found at Mid
        else
            if A[Mid] < X
                Set Lo to Mid+1
            else if A[Mid] > X
                Set Hi to Mid-1
            end if
        end if
    End While

End Procedure
```