Course Name: **A Level (2nd Sem)**
Topic: **Interpolation Search in C++**
Subject: **Data Structure using C++**
Date: **24-04-2020**

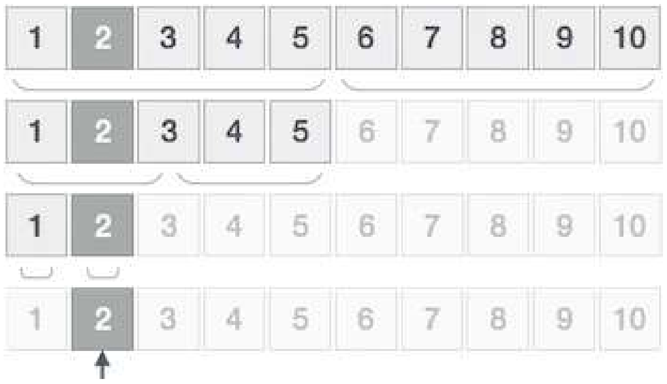
Interpolation search is an improved variant of binary search. This search algorithm works on the probing position of the required value. For this algorithm to work properly, the data collection should be in a sorted form and equally distributed.

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

There are cases where the location of target data may be known in advance. E.g. 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

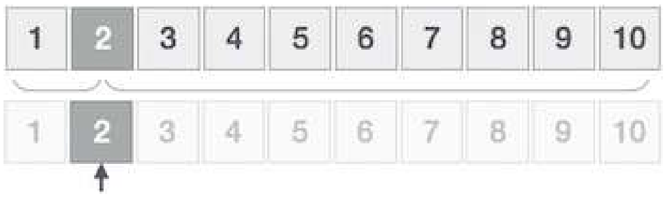
In binary search, if the desired data is not found then the rest of the list is divided in two parts, lower and higher. The search is carried out in either of them.



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 −

$$\mathbf{mid = Lo + ((Hi - Lo) / (A[Hi] - A[Lo])) * (X - A[Lo])}$$

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 sub-array to the left of the middle item. This process continues on the sub-array as well until the size of sub-array reduces to zero.

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

Coding

```
#include<stdio.h>
#define MAX 10

// array of items on which linear search will be conducted.
int list[MAX] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44 };

int find(int data)
 {
  int lo = 0;
  int hi = MAX - 1;
  int mid = -1;
  int comparisons = 1;
  int index = -1;

  while(lo <= hi)
 {
    printf("\nComparison %d  \n" , comparisons ) ;
    printf("lo : %d, list[%d] = %d\n", lo, lo, list[lo]);
    printf("hi : %d, list[%d] = %d\n", hi, hi, list[hi]);

    comparisons++;

    // probe the mid point
    mid = lo + (((double)(hi - lo) / (list[hi] - list[lo])) * (data - list[lo]));
    printf("mid = %d\n",mid);

    // data found
    if(list[mid] == data)
    {
      index = mid;
      break;
    }
   else
   {
      if(list[mid] < data)
```

```
        {
          // if data is larger, data is in upper half
          lo = mid + 1;
        }
        else
         {
          // if data is smaller, data is in lower half
          hi = mid - 1;
        }
      }
    }

    printf("\nTotal comparisons made: %d", --comparisons);
    return index;
}

int main()
{
    //find location of 33
    int location = find(33);

    // if element was found
    if(location != -1)
      printf("\nElement found at location: %d" ,(location+1));
    else
      printf("Element not found.");

    return 0;
}
```

Output

```
Comparison 1
lo : 0, list[0] = 10
hi : 9, list[9] = 44
mid = 6

Total comparisons made: 1
Element found at location: 7
```