Lab 08: Implementing Interpolation Search

ICT 2202: Algorithm Analysis Design and Lab

Submitted By

Group 1

Tahshin Islam Tuhin(ID: 1957)

Nobin Chandra(ID: 1958)

Md. Shaon Khan(ID: 1984)

Submitted To

Professor Fahima Tabassum, PhD

Institute of Information Technology

Jahangirnagar University

Submission Date: 06/1/2026

Overview:

Interpolation Search is an efficient algorithm designed to locate a specific value within a sorted array by mimicking the way humans search for a name in a telephone directory. Unlike Binary Search, which always splits the data in the exact middle, Interpolation Search calculates a probable position for the target element based on the value of the key and the range of data. It uses a mathematical formula to "probe" the position, effectively jumping closer to where the item is likely to be—checking near the beginning for smaller values and near the end for larger ones. This approach allows the algorithm to converge on the target much faster when the data values are uniformly distributed.

The probing position formula used in Interpolation Search is:

$$\text{pos} = \text{low} + \frac{(\text{target} - A[\text{low}]) \times (\text{high} - \text{low})}{A[\text{high}] - A[\text{low}]}$$

Here:

- A is the sorted array
- low is the starting index
- high is the ending index
- target is the value to be searched

In terms of performance, this method offers a significant improvement over standard searching techniques for large, uniform datasets, achieving an average time complexity of $O(\log(\log n))$. This makes it extremely fast compared to the $O(\log n)$complexity of Binary Search in ideal conditions. However, the algorithm is highly sensitive to the distribution of the data; if the elements are not spread out evenly, the performance can degrade to that of a linear search, or $O(n)$. Therefore, while Interpolation Search is a powerful tool for reducing the number of comparisons in predictable data, it requires the dataset to be both sorted and uniformly distributed to maintain its efficiency advantage.

Problem-01: Interpolation Search

Algorithm:

1. Initialize two indices, low and high, representing the lower and upper bounds of the sorted array.
2. Verify the search condition by ensuring that low ≤ high and that the target key ‘target’ lies within the current search range, i.e.,

$$\text{target} \geq A[\text{low}] \text{ and } K \leq A[\text{high}]$$

   If this condition is not satisfied, terminate the search.

3. Compute the estimated position of the target element using the interpolation formula:

$$\text{pos} = \text{low} + \left(\frac{\text{target} - A[\text{low}]}{A[\text{high}] - A[\text{low}]}\right) \times (\text{high} - \text{low})$$

Pseudocode:

```
INTERPOLATION_SEARCH(A, n, target)
  low = 0
  high = n − 1

  WHILE low ≤ high AND target ≥ A[low] AND target ≤ A[high] DO
    IF A[low] = A[high] THEN
      IF A[low] = target THEN
        RETURN low
      ELSE
        RETURN -1
      END IF
    END IF

    pos = low + ((target − A[low]) × (high − low)) / (A[high] − A[low])

    IF A[pos] = target THEN
      RETURN pos
    ELSE IF A[pos] < target THEN
      low = pos + 1
    ELSE
      high = pos − 1
    END IF
  END WHILE

  RETURN -1
END
```

Code:

```
#include <iostream>
using namespace std;

int interpolationSearch(int A[], int n, int target)
{
   int low = 0;
   int high = n - 1;

   while (low <= high && target >= A[low] && target <= A[high])
   {
      if (A[low] == A[high])
      {
         if (A[low] == target)
            return low;
         else
            return -1;
      }

      int pos = low + ((target - A[low]) * (high - low)) / (A[high] - A[low]);

      if (A[pos] == target)
         return pos;
      else if (A[pos] < target)
         low = pos + 1;
      else
         high = pos - 1;
   }

   return -1;
}

int main()
{
   int A[] = {11, 21, 31, 41, 51, 61};
   int n = sizeof(A) / sizeof(A[0]);
   int target = 41;

   int result = interpolationSearch(A, n, target);

   if (result != -1)
      cout << "Element found at index: " << result << endl;
   else
      cout << "Element not found" << endl;

   return 0;
}
```

Output:

```
Element found at index: 3
```

Explanation:

Interpolation Search is an advanced searching technique used to find a target element in a sorted array. Unlike Binary Search, which always checks the middle element, Interpolation Search estimates the likely position of the target by considering the value of the target and the range of values in the array. This estimation allows the algorithm to jump closer to the expected location of the element, making it faster when the data is uniformly distributed.

The algorithm starts by setting two indices, low and high, representing the first and last positions of the array. It then checks whether the target lies within the current range of values. If the condition is satisfied, the probable position of the target is calculated using the interpolation formula. The value at this estimated position is compared with the target. If they are equal, the search is successful. If the value is smaller than the target, the search continues in the upper portion of the array by updating low. If the value is larger, the search moves to the lower portion by updating high. This process repeats until the target is found or the search range becomes invalid.

Interpolation Search works efficiently only when the array elements are sorted and uniformly distributed. If the distribution is uneven, the estimated positions may be inaccurate, causing the algorithm to behave like a linear search.

Time Complexity:

- Best Case: O(1), when the target is found at the first estimated position.
- Average Case: O(log log n), when the data is uniformly distributed and the estimation is accurate.
- Worst Case: O(n), when the data is not uniformly distributed, leading to poor position estimates.

Space Complexity:

- Best, Average, and Worst Case: O(1), since the algorithm uses only a constant amount of extra memory.

This makes Interpolation Search a powerful and efficient algorithm for large, well-distributed datasets, but less suitable for data with irregular value distribution.

Discussion:

Interpolation Search is a value-based searching technique that improves upon Binary Search by estimating the most probable position of the target element instead of blindly dividing the array into two equal halves. When the data is sorted and uniformly distributed, this estimation significantly reduces the number of comparisons, making the algorithm extremely fast for large datasets.

However, the efficiency of Interpolation Search is highly dependent on the distribution of data. If the elements are evenly spaced, the algorithm performs exceptionally well with an average time complexity of $O(\log \log n)$. On the other hand, if the data is clustered or irregularly distributed, the estimated positions become inaccurate. In such cases, the algorithm may repeatedly narrow the search range by small amounts, causing the performance to degrade to $O(n)$, similar to linear search.

Another important consideration is that Interpolation Search requires the array to be sorted and numeric in nature, as the probing formula relies on arithmetic calculations. This makes it unsuitable for non-numeric data or datasets where sorting is expensive or impractical.

In practice, Interpolation Search is best used in scenarios involving large, static datasets with uniformly distributed values, such as searching in indexed files or databases. For general-purpose searching where data distribution is unknown, Binary Search is often preferred due to its consistent and predictable performance.