Interpolation Search - GeeksforGeeks

Source: https://www.geeksforgeeks.org/interpolation-search/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Interpolation Search Last Updated : 5 Feb, 2025 Given a sorted array of n uniformly distributed values arr[], write a function to search for a particular element x in the array. Linear Search finds the element in O(n) time, Jump Search takes O(n) time and Binary Search takes O(log n) time. The Interpolation Search is an improvement over Binary Search for instances, where the values in a sorted array are uniformly distributed. Interpolation constructs new data points within the range of a discrete set of known data points. Binary Search always goes to the middle element to check. On the other hand, interpolation search may go to different locations according to the value of the key being searched. For example, if the value of the key is closer to the last element, interpolation search is likely to start search toward the end side. To find the position to be searched, it uses the following formula. // The idea of formula is to return higher value of pos // when element to be searched is closer to arr[hi]. And // smaller value when closer to arr[lo] arr[] ==> Array where elements need to be searched x ==> Element to be searched lo ==> Starting index in arr[] hi ==> Ending index in arr[] pos = lo + [\frac{(x-arr[lo])*(hi-lo) }{ (arr[hi]-arr[Lo]) }] There are many different interpolation methods and one such is known as linear interpolation. Linear interpolation takes two data points which we assume as (x1,y1) and (x2,y2) and the formula is : at point(x,y). This algorithm works in a way we search for a word in a dictionary. The interpolation search algorithm improves the binary search algorithm. The formula for finding a value is: K = data-low/high-low. K is a constant which is used to narrow the search space. In the case of binary search, the value for this constant is: K=(low+high)/2. The formula for pos can be derived as follows. Let's assume that the elements of the array are linearly distributed. General equation of line : y = m*x + c. y is the value in the array and x is its index. Now putting value of lo,hi and x in the equation arr[hi] = m*hi+c ----(1) arr[lo] = m*lo+c ----(2) x = m*pos + c ----(3) m = (arr[hi] - arr[lo])/ (hi - lo) subtracting eqxn (2) from (3) x - arr[lo] = m * (pos - lo) lo + (x - arr[lo])/m = pos pos = lo + (x - arr[lo]) *(hi - lo)/(arr[hi] - arr[lo]) Algorithm The rest of the Interpolation algorithm is the same except for the above partition logic. Step1: In a loop, calculate the value of "pos" using the probe position formula. Step2: If it is a match, return the index of the item, and exit. Step3: If the item is less than arr[pos], calculate the probe position of the left sub-array. Otherwise, calculate the same in the right sub-array. Step4: Repeat until a match is found or the sub-array reduces to zero. Below is the implementation of the algorithm. C++ // C++ program to implement interpolation // search with recursion #include <bits/stdc++.h> using namespace std ; // If x is present in arr[0..n-1], then returns // index of it, else returns -1. int interpolationSearch (int arr [], int lo , int hi , int x) { int pos ; // Since array is sorted, an element present // in array must be in range defined by corner if (lo <= hi && x >= arr [lo] && x <= arr [hi]) { // Probing the position with keeping // uniform distribution in mind. pos = lo + (((double)(hi - lo) / (arr [hi] - arr [lo])) * (x - arr [lo])); // Condition of target found if (arr [pos] == x) return pos ; // If x is larger, x is in right sub array if (arr [pos] < x) return interpolationSearch (arr , pos + 1 , hi , x); // If x is smaller, x is in left sub array if (arr [pos] > x) return interpolationSearch (arr , lo , pos - 1 , x); } return -1 ; } // Driver Code int main () { // Array of items on which search will // be conducted. int arr [] = { 10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47 }; int n = sizeof (arr) / sizeof (arr [0]); // Element to be searched int x = 18 ; int index = interpolationSearch (arr , 0 , n - 1 , x); // If element was found if (index != -1) cout << "Element found at index " << index ; else cout << "Element not found." ; return 0 ; } // This code is contributed by equbalzeeshan C // C program to implement interpolation search // with recursion #include <stdio.h> // If x is present in arr[0..n-1], then returns // index of it, else returns -1. int

```
interpolationSearch ( int arr [], int lo , int hi , int x ) { int pos ; // Since array is sorted, an element present // in array must be in range defined by corner if ( lo <= hi && x >= arr [ lo ] && x <= arr [ hi ]) { // Probing the position with keeping // uniform distribution in mind. pos = lo + ((( double )( hi - lo ) / ( arr [ hi ] - arr [ lo ])) * ( x - arr [ lo ])); // Condition of target found if ( arr [ pos ] == x ) return pos ; // If x is larger, x is in right sub array if ( arr [ pos ] < x ) return interpolationSearch ( arr , pos + 1 , hi , x ); // If x is smaller, x is in left sub array if ( arr [ pos ] > x ) return interpolationSearch ( arr , lo , pos - 1 , x ); } return -1 ; } // Driver Code int main () { // Array of items on which search will // be conducted. int arr [] = { 10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47 }; int n = sizeof ( arr ) / sizeof ( arr [ 0 ]); int x = 18 ; // Element to be searched int index = interpolationSearch ( arr , 0 , n - 1 , x ); // If element was found if ( index != -1 ) printf ( "Element found at index %d" , index ); else printf ( "Element not found." ); return 0 ; } Java // Java program to implement interpolation // search with recursion import java.util.* ; class GFG { // If x is present in arr[0..n-1], then returns // index of it, else returns -1. public static int interpolationSearch ( int arr [] , int lo , int hi , int x ) { int pos ; // Since array is sorted, an element // present in array must be in range // defined by corner if ( lo <= hi && x >= arr [ lo ] && x <= arr [ hi ] ) { // Probing the position with keeping // uniform distribution in mind. pos = lo + ((( hi - lo ) / ( arr [ hi ] - arr [ lo ] )) * ( x - arr [ lo ] )); // Condition of target found if ( arr [ pos ] == x ) return pos ; // If x is larger, x is in right sub array if ( arr [ pos ] < x ) return interpolationSearch ( arr , pos + 1 , hi , x ); // If x is smaller, x is in left sub array if ( arr [ pos ] > x ) return interpolationSearch ( arr , lo , pos - 1 , x ); } return - 1 ; } // Driver Code public static void main ( String [] args ) { // Array of items on which search will // be conducted. int arr [] = { 10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47 }; int n = arr . length ; // Element to be searched int x = 18 ; int index = interpolationSearch ( arr , 0 , n - 1 , x ); // If element was found if ( index != - 1 ) System . out . println ( "Element found at index " + index ); else System . out . println ( "Element not found." ); } } // This code is contributed by equbalzeeshan Python # Python3 program to implement # interpolation search # with recursion # If x is present in arr[0..n-1], then # returns index of it, else returns -1. def interpolationSearch ( arr , lo , hi , x ): # Since array is sorted, an element present # in array must be in range defined by corner if ( lo <= hi and x >= arr [ lo ] and x <= arr [ hi ]): # Probing the position with keeping # uniform distribution in mind. pos = lo + (( hi - lo ) // ( arr [ hi ] - arr [ lo ]) * ( x - arr [ lo ])) # Condition of target found if arr [ pos ] == x : return pos # If x is larger, x is in right subarray if arr [ pos ] < x : return interpolationSearch ( arr , pos + 1 , hi , x ) # If x is smaller, x is in left subarray if arr [ pos ] > x : return interpolationSearch ( arr , lo , pos - 1 , x ) return - 1 # Driver code # Array of items in which # search will be conducted arr = [ 10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47 ] n = len ( arr ) # Element to be searched x = 18 index = interpolationSearch ( arr , 0 , n - 1 , x ) if index != - 1 : print ( "Element found at index" , index ) else : print ( "Element not found" ) # This code is contributed by Hardik Jain C# // C# program to implement // interpolation search using System ; class GFG { // If x is present in // arr[0..n-1], then // returns index of it, // else returns -1. static int interpolationSearch ( int [] arr , int lo , int hi , int x ) { int pos ; // Since array is sorted, an element // present in array must be in range // defined by corner if ( lo <= hi && x >= arr [ lo ] && x <= arr [ hi ]) { // Probing the position // with keeping uniform // distribution in mind. pos = lo + ((( hi - lo ) / ( arr [ hi ] - arr [ lo ])) * ( x - arr [ lo ])); // Condition of // target found if ( arr [ pos ] == x ) return pos ; // If x is larger, x is in right sub array if ( arr [ pos ] < x ) return interpolationSearch ( arr , pos + 1 , hi , x ); // If x is smaller, x is in left sub array if ( arr [ pos ] > x ) return interpolationSearch ( arr , lo , pos - 1 , x ); } return - 1 ; } // Driver Code public static void Main () { // Array of items on which search will // be conducted. int [] arr = new int []{ 10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47 }; // Element to be searched int x = 18 ; int n = arr . Length ; int index = interpolationSearch ( arr , 0 , n - 1 , x ); // If element was found if ( index != - 1 ) Console . WriteLine ( "Element found at index " + index ); else Console . WriteLine ( "Element not found." ); } } // This code is contributed by equbalzeeshan JavaScript < script > // Javascript program to implement Interpolation Search // If x is present in arr[0..n-1], then returns // index of it, else returns -1. function interpolationSearch ( arr , lo , hi , x ){ let pos ; // Since array is sorted, an element present // in array must be in range defined by corner if ( lo <= hi && x >= arr [ lo ] && x <= arr [ hi ]) { // Probing the position with keeping // uniform distribution in mind. pos = lo + Math . floor ((( hi - lo ) / ( arr [ hi ] - arr [ lo ])) * ( x - arr [ lo ]));; // Condition of target found if ( arr [ pos ] == x ){ return pos ; } // If x is larger, x is in right sub array if ( arr [ pos ] < x ){ return interpolationSearch ( arr , pos + 1 , hi , x ); } // If x is smaller, x is in left sub array if ( arr [ pos ] > x ){ return interpolationSearch ( arr , lo , pos - 1 , x ); } } return - 1 ; } // Driver Code let arr = [ 10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47 ]; let n = arr . length ; // Element to be searched let x = 18 let index = interpolationSearch ( arr , 0 , n - 1 , x ); // If element was found if ( index != - 1 ){ document . write ( `Element found at index ${ index } ` ) } else { document . write ( "Element not found"
```

); } // This code is contributed by _saurabh_jaiswal < /script> PHP <?php // PHP program to implement $erpolation search // with recursion // If x is present in arr[0..n-1], then returns // index of it, else returns -1. function interpolationSearch ($arr , $lo , $hi , $x) { // Since array is sorted, an element present // in array must be in range defined by corner if ($lo <= $hi && $x >= $arr [$lo] && $x <= $arr [$hi]) { // Probing the position with keeping // uniform distribution in mind. $pos = (int)($lo + (((double)($hi - $lo) / ($arr [$hi] - $arr [$lo])) * ($x - $arr [$lo]))); // Condition of target found if ($arr [$pos] == $x) return $pos ; // If x is larger, x is in right sub array if ($arr [$pos] < $x) return interpolationSearch ($arr , $pos + 1 , $hi , $x); // If x is smaller, x is in left sub array if ($arr [$pos] > $x) return interpolationSearch ($arr , $lo , $pos - 1 , $x); } return - 1 ; } // Driver Code // Array of items on which search will // be conducted. $arr = array (10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47); $n = sizeof ($arr); $x = 47 ; // Element to be searched $index = interpolationSearch ($arr , 0 , $n - 1 , $x); // If element was found if ($index != - 1) echo "Element found at index " . $index ; else echo "Element not found." ; return 0 ; #This code is contributed by Susobhan Akhuli ?> Output Element found at index 4 Time Complexity: O(log 2 (log 2 n)) for the average case, and O(n) for the worst case Auxiliary Space Complexity: O(1) Another approach:- This is the iteration approach for the interpolation search. Step1: In a loop, calculate the value of "pos" using the probe position formula. Step2: If it is a match, return the index of the item, and exit. Step3: If the item is less than arr[pos], calculate the probe position of the left sub-array. Otherwise, calculate the same in the right sub-array. Step4: Repeat until a match is found or the sub-array reduces to zero. Below is the implementation of the algorithm. C++ // C++ program to implement interpolation search by using iteration approach #include <bits/stdc++.h> using namespace std ; int interpolationSearch (int arr [], int n , int x) { // Find indexes of two corners int low = 0 , high = (n - 1); // Since array is sorted, an element present // in array must be in range defined by corner while (low <= high && x >= arr [low] && x <= arr [high]) { if (low == high) { if (arr [low] == x) return low ; return -1 ; } // Probing the position with keeping // uniform distribution in mind. int pos = low + (((double)(high - low) / (arr [high] - arr [low])) * (x - arr [low])); // Condition of target found if (arr [pos] == x) return pos ; // If x is larger, x is in upper part if (arr [pos] < x) low = pos + 1 ; // If x is smaller, x is in the lower part else high = pos - 1 ; } return -1 ; } // Main function int main () { // Array of items on whighch search will // be conducted. int arr [] = { 10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47 }; int n = sizeof (arr) / sizeof (arr [0]); int x = 18 ; // Element to be searched int index = interpolationSearch (arr , n , x); // If element was found if (index != -1) cout << "Element found at index " << index ; else cout << "Element not found." ; return 0 ; } //this code contributed by Ajay Singh Java // Java program to implement interpolation // search with recursion import java.util.* ; class GFG { // If x is present in arr[0..n-1], then returns // index of it, else returns -1. public static int interpolationSearch (int arr [] , int lo , int hi , int x) { int pos ; if (lo <= hi && x >= arr [lo] && x <= arr [hi]) { // Probing the position with keeping // uniform distribution in mind. pos = lo + (((hi - lo) / (arr [hi] - arr [lo])) * (x - arr [lo])); // Condition of target found if (arr [pos] == x) return pos ; // If x is larger, x is in right sub array if (arr [pos] < x) return interpolationSearch (arr , pos + 1 , hi , x); // If x is smaller, x is in left sub array if (arr [pos] > x) return interpolationSearch (arr , lo , pos - 1 , x); } return - 1 ; } // Driver Code public static void main (String [] args) { // Array of items on which search will // be conducted. int arr [] = { 10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47 }; int n = arr . length ; // Element to be searched int x = 18 ; int index = interpolationSearch (arr , 0 , n - 1 , x); // If element was found if (index != - 1) System . out . println ("Element found at index " + index); else System . out . println ("Element not found."); } } Python # Python equivalent of above C++ code # Python program to implement interpolation search by using iteration approach def interpolationSearch (arr , n , x): # Find indexes of two corners low = 0 high = (n - 1) # Since array is sorted, an element present # in array must be in range defined by corner while low <= high and x >= arr [low] and x <= arr [high]: if low == high : if arr [low] == x : return low ; return - 1 ; # Probing the position with keeping # uniform distribution in mind. pos = int (low + (((float (high - low) / (arr [high] - arr [low])) * (x - arr [low])))) # Condition of target found if arr [pos] == x : return pos # If x is larger, x is in upper part if arr [pos] < x : low = pos + 1 ; # If x is smaller, x is in lower part else : high = pos - 1 ; return - 1 # Main function if __name__ == "__main__" : # Array of items on whighch search will # be conducted. arr = [10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47] n = len (arr) x = 18 # Element to be searched index = interpolationSearch (arr , n , x) # If element was found if index != - 1 : print ("Element found at index" , index) else : print ("Element not found") C# // C# program to implement interpolation search by using // iteration approach using System ; class Program { // Interpolation Search function static int InterpolationSearch (int [] arr , int n , int x) { int low = 0 ; int high = n - 1 ; while (low <= high && x >= arr [low] && x <= arr [high]) { if (low == high) { if (arr [low] == x) return low ;

return - 1 ; } int pos = low + (int)(((float)(high - low) / (arr [high] - arr [low])) * (x - arr [low])); if (arr [pos] == x) return pos ; if (arr [pos] < x) low = pos + 1 ; else high = pos - 1 ; } return - 1 ; } // Main function static void Main (string [] args) { int [] arr = { 10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47 }; int n = arr . Length ; int x = 18 ; int index = InterpolationSearch (arr , n , x); if (index != - 1) Console . WriteLine ("Element found at index " + index); else Console . WriteLine ("Element not found"); } } // This code is contributed by Susobhan Akhuli JavaScript // JavaScript program to implement interpolation search by using iteration approach function interpolationSearch (arr , n , x) { // Find indexes of two corners let low = 0 ; let high = n - 1 ; // Since array is sorted, an element present // in array must be in range defined by corner while (low <= high && x >= arr [low] && x <= arr [high]) { if (low == high) { if (arr [low] == x) { return low ; } return - 1 ; } // Probing the position with keeping // uniform distribution in mind. let pos = Math . floor (low + (((high - low) / (arr [high] - arr [low])) * (x - arr [low]))); // Condition of target found if (arr [pos] == x) { return pos ; } // If x is larger, x is in upper part if (arr [pos] < x) { low = pos + 1 ; } // If x is smaller, x is in lower part else { high = pos - 1 ; } } return - 1 ; } // Main function let arr = [10 , 12 , 13 , 16 , 18 , 19 , 20 , 21 , 22 , 23 , 24 , 33 , 35 , 42 , 47]; let n = arr . length ; let x = 18 ; // Element to be searched let index = interpolationSearch (arr , n , x); // If element was found if (index != - 1) { console . log ("Element found at index" , index); } else { console . log ("Element not found"); } Output Element found at index 4 Time Complexity: O(log2(log2 n)) for the average case, and O(n) for the worst case Auxiliary Space Complexity: O(1) Comment Article Tags: Article Tags: DSA