Jump Search - GeeksforGeeks

Source: https://www.geeksforgeeks.org/jump-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 Jump Search Last Updated : 24 Apr, 2025 Like Binary Search , Jump Search is a searching algorithm for sorted arrays. The basic idea is to check fewer elements (than linear search) by jumping ahead by fixed steps or skipping some elements in place of searching all elements. For example, suppose we have an array arr[] of size n and a block (to be jumped) of size m. Then we search in the indexes arr[0], arr[m], arr[2m].....arr[km], and so on. Once we find the interval (arr[km] < x < arr[(k+1)m]), we perform a linear search operation from the index km to find the element x. Let’s consider the following array: (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610). The length of the array is 16. The Jump search will find the value of 55 with the following steps assuming that the block size to be jumped is 4. STEP 1: Jump from index 0 to index 4; STEP 2: Jump from index 4 to index 8; STEP 3: Jump from index 8 to index 12; STEP 4: Since the element at index 12 is greater than 55, we will jump back a step to come to index 8. STEP 5: Perform a linear search from index 8 to get the element 55. Performance in comparison to linear and binary search: If we compare it with linear and binary search then it comes out then it is better than linear search but not better than binary search. The increasing order of performance is: linear search < jump search < binary search What is the optimal block size to be skipped? In the worst case, we have to do n/m jumps, and if the last checked value is greater than the element to be searched for, we perform m-1 comparisons more for linear search. Therefore, the total number of comparisons in the worst case will be ((n/m) + m-1). The value of the function ((n/m) + m-1) will be minimum when m = √n. Therefore, the best step size is m = √ n. Algorithm steps Jump Search is an algorithm for finding a specific value in a sorted array by jumping through certain steps in the array. The steps are determined by the sqrt of the length of the array. Here is a step-by-step algorithm for the jump search: Determine the step size m by taking the sqrt of the length of the array n. Start at the first element of the array and jump m steps until the value at that position is greater than the target value. Once a value greater than the target is found, perform a linear search starting from the previous step until the target is found or it is clear that the target is not in the array. If the target is found, return its index. If not, return -1 to indicate that the target was not found in the array. Example 1 : C++ // C++ program to implement Jump Search #include <bits/stdc++.h> using namespace std ; int jumpSearch (int arr [], int x , int n) { // Finding block size to be jumped int step = sqrt (n); // Finding the block where element is // present (if it is present) int prev = 0 ; while (arr [min (step , n) -1] < x) { prev = step ; step += sqrt (n); if (prev >= n) return -1 ; } // Doing a linear search for x in block // beginning with prev. while (arr [prev] < x) { prev ++ ; // If we reached next block or end of // array, element is not present. if (prev == min (step , n)) return -1 ; } // If element is found if (arr [prev] == x) return prev ; return -1 ; } // Driver program to test function int main () { int arr [] = { 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 , 610 }; int x = 55 ; int n = sizeof (arr) / sizeof (arr [0]); // Find the index of 'x' using Jump Search int index = jumpSearch (arr , x , n); // Print the index where 'x' is located cout << " \n Number " << x << " is at index " << index ; return 0 ; } // Contributed by nuclode C #include <stdio.h> #include <math.h> int min (int a , int b){ if (b > a) return a ; else return b ; } int jumpsearch (int arr [], int x , int n) { // Finding block size to be jumped int step = sqrt (n); // Finding the block where element is // present (if it is present) int prev = 0 ; while (arr [min (step , n) -1] < x) { prev = step ; step += sqrt (n); if (prev >= n) return -1 ; } // Doing a linear search for x in block // beginning with prev. while (arr [prev] < x) { prev ++ ; // If we reached next block or end of // array, element is not present. if (prev == min (step , n)) return -1 ; } // If element is found if (arr [prev] == x) return prev ; return -1 ; } int main ()

{ int arr [] = { 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 , 610 }; int x = 55 ; int n = sizeof (arr) / sizeof (arr [0]); int index = jumpsearch (arr , x , n); if (index >= 0) printf ("Number is at %d index" , index); else printf ("Number is not exist in the array"); return 0 ; } // This code is contributed by Susobhan Akhuli Java // Java program to implement Jump Search. public class JumpSearch { public static int jumpSearch (int [] arr , int x) { int n = arr . length ; // Finding block size to be jumped int step = (int) Math . floor (Math . sqrt (n)); // Finding the block where element is // present (if it is present) int prev = 0 ; for (int minStep = Math . min (step , n) - 1 ; arr [minStep] < x ; minStep = Math . min (step , n) - 1) { prev = step ; step += (int) Math . floor (Math . sqrt (n)); if (prev >= n) return - 1 ; } // Doing a linear search for x in block // beginning with prev. while (arr [prev] < x) { prev ++ ; // If we reached next block or end of // array, element is not present. if (prev == Math . min (step , n)) return - 1 ; } // If element is found if (arr [prev] == x) return prev ; return - 1 ; } // Driver program to test function public static void main (String [ ] args) { int arr [] = { 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 , 610 }; int x = 55 ; // Find the index of 'x' using Jump Search int index = jumpSearch (arr , x); // Print the index where 'x' is located System . out . println ("\nNumber " + x + " is at index " + index); } } Python # Python3 code to implement Jump Search import math def jumpSearch (arr , x , n): # Finding block size to be jumped step = math . sqrt (n) # Finding the block where element is # present (if it is present) prev = 0 while arr [int (min (step , n) - 1)] < x : prev = step step += math . sqrt (n) if prev >= n : return - 1 # Doing a linear search for x in # block beginning with prev. while arr [int (prev)] < x : prev += 1 # If we reached next block or end # of array, element is not present. if prev == min (step , n): return - 1 # If element is found if arr [int (prev)] == x : return prev return - 1 # Driver code to test function arr = [0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 , 610] x = 55 n = len (arr) # Find the index of 'x' using Jump Search index = jumpSearch (arr , x , n) # Print the index where 'x' is located print ("Number" , x , "is at index" , " %.0f " % index) # This code is contributed by "Sharad_Bhardwaj". C# // C# program to implement Jump Search. using System ; public class JumpSearch { public static int jumpSearch (int [] arr , int x) { int n = arr . Length ; // Finding block size to be jumped int step = (int) Math . Sqrt (n); // Finding the block where the element is // present (if it is present) int prev = 0 ; for (int minStep = Math . Min (step , n) - 1 ; arr [minStep] < x ; minStep = Math . Min (step , n) - 1) { prev = step ; step += (int) Math . Sqrt (n); if (prev >= n) return - 1 ; } // Doing a linear search for x in block // beginning with prev. while (arr [prev] < x) { prev ++ ; // If we reached next block or end of // array, element is not present. if (prev == Math . Min (step , n)) return - 1 ; } // If element is found if (arr [prev] == x) return prev ; return - 1 ; } // Driver program to test function public static void Main () { int [] arr = { 0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 , 610 }; int x = 55 ; // Find the index of 'x' using Jump Search int index = jumpSearch (arr , x); // Print the index where 'x' is located Console . Write ("Number " + x + " is at index " + index); } } JavaScript < script > // Javascript program to implement Jump Search function jumpSearch (arr , x , n) { // Finding block size to be jumped let step = Math . sqrt (n); // Finding the block where element is // present (if it is present) let prev = 0 ; for (int minStep = Math . Min (step , n) - 1 ; arr [minStep] < x ; minStep = Math . Min (step , n) - 1) { prev = step ; step += Math . sqrt (n); if (prev >= n) return - 1 ; } // Doing a linear search for x in block // beginning with prev. while (arr [prev] < x) { prev ++ ; // If we reached next block or end of // array, element is not present. if (prev == Math . min (step , n)) return - 1 ; } // If element is found if (arr [prev] == x) return prev ; return - 1 ; } // Driver program to test function let arr = [0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 , 610]; let x = 55 ; let n = arr . length ; // Find the index of 'x' using Jump Search let index = jumpSearch (arr , x , n); // Print the index where 'x' is located document . write (`Number ${ x } is at index ${ index } `); // This code is contributed by _saurabh_jaiswal < /script> PHP <?php // PHP program to implement Jump Search function jumpSearch ($arr , $x , $n) { // Finding block size to be jumped $step = sqrt ($n); // Finding the block where element is // present (if it is present) $prev = 0 ; while ($arr [min ($step , $n) - 1] < $x) { $prev = $step ; $step += sqrt ($n); if ($prev >= $n) return - 1 ; } // Doing a linear search for x in block // beginning with prev. while ($arr [$prev] < $x) { $prev ++ ; // If we reached next block or end of // array, element is not present. if ($prev == min ($step , $n)) return - 1 ; } // If element is found if ($arr [$prev] == $x) return $prev ; return - 1 ; } // Driver program to test function $arr = array (0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 , 55 , 89 , 144 , 233 , 377 , 610); $x = 55 ; $n = sizeof ($arr) / sizeof ($arr [0]); // Find the index of '$x' using Jump Search $index = jumpSearch ($arr , $x , $n); // Print the index where '$x' is located echo "Number " . $x . " is at index " . $index ; return 0 ; ?> Output: Number 55 is at index 10 Time Complexity : O(?n) Auxiliary Space : O(1) Advantages of Jump Search: Better than a linear search for arrays where the elements are uniformly distributed. Jump search has a lower time complexity compared to a linear search for large arrays. The number of steps taken in jump search is proportional to the square root of the size of

the array, making it more efficient for large arrays. It is easier to implement compared to other search algorithms like binary search or ternary search. Jump search works well for arrays where the elements are in order and uniformly distributed, as it can jump to a closer position in the array with each iteration. Important points: Works only with sorted arrays. The optimal size of a block to be jumped is (? n). This makes the time complexity of Jump Search O(? n). The time complexity of Jump Search is between Linear Search ((O(n)) and Binary Search (O(Log n)). Binary Search is better than Jump Search, but Jump Search has the advantage that we traverse back only once (Binary Search may require up to O(Log n) jumps, consider a situation where the element to be searched is the smallest element or just bigger than the smallest). So, in a system where binary search is costly, we use Jump Search. References: https://en.wikipedia.org/wiki/Jump_search If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above. Comment Article Tags: Article Tags: Misc Searching DSA