Exponential Search - GeeksforGeeks

Source: https://www.geeksforgeeks.org/exponential-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 Exponential Search Last Updated : 23 Jul, 2025 The name of this searching algorithm may be misleading as it works in O(Log n) time. The name comes from the way it searches an element. Given a sorted array, and an element x to be searched, find position of x in the array. Input : arr[] = {10, 20, 40, 45, 55} x = 45 Output : Element found at index 3 Input : arr[] = {10, 15, 25, 45, 55} x = 15 Output : Element found at index 1 We have discussed, linear search , binary search for this problem. Exponential search involves two steps: Find range of indexes where element is present Do Binary Search in above found range. How to find the range where element may be present? The idea is to start with subarray size 1, compare its last element with x, then try size 2, then 4 and so on until last element of a subarray is not greater. Once we find an index i (after repeated doubling of i), we know that the element must be present between i/2 and i (Why i/2? because we could not find a greater value in previous iteration) Recursive Implementation - O(Log n) Time and O(Log n) Space We start with an index i equal to 1 and repeatedly double it until either i is greater than or equal to the length of the array or the value at index i is greater than or equal to the target value x. We then perform a binary search on the range [i/2, min(i, n-1)], where n is the length of the array. This range is guaranteed to contain the target value, if it is present in the array, because we know that the target value must be greater than or equal to the value at index i/2 and less than or equal to the value at index min(i, n-1). If we find the target value in the binary search, we return its index. Otherwise, we return -1 to indicate that the target value is not present in the array. We mainly use recursive implementation of binary search once we find the range. We use iterative code to find the range. C++ #include <bits/stdc++.h> using namespace std ; int binarySearch (vector < int >& arr , int l , int r , int x); // Returns position of first occurrence of // x in array int exponentialSearch (vector < int >& arr , int n , int x) { // If x is present at first location itself if (arr [0] == x) return 0 ; // Find range for binary search by // repeated doubling int i = 1 ; while (i < n && arr [i] <= x) i = i * 2 ; // Call binary search for the found range. return binarySearch (arr , i / 2 , min (i , n - 1), x); } // A recursive binary search function. It returns // location of x in given array arr[l..r] is // present, otherwise -1 int binarySearch (vector < int >& arr , int l , int r , int x) { if (r >= l) { int mid = l + (r - l) / 2 ; // If the element is present at the middle // itself if (arr [mid] == x) return mid ; // If element is smaller than mid, then it // can only be present n left subarray if (arr [mid] > x) return binarySearch (arr , l , mid - 1 , x); // Else the element can only be present // in right subarray return binarySearch (arr , mid + 1 , r , x); } // We reach here when element is not present // in array return -1 ; } // Driver code int main (void) { vector < int > arr = { 2 , 3 , 4 , 10 , 40 }; int n = arr . size (); int x = 10 ; int result = exponentialSearch (arr , n , x); (result == -1) ? cout << "Element is not present in array" : cout << "Element is present at index " << result ; return 0 ; } C // C++ program to find an element x in a // sorted array using Exponential search. #include <stdio.h> #include <time.h> #include <math.h> #define min int binarySearch (int arr [], int , int , int); // Returns position of first occurrence of // x in array int exponentialSearch (int arr [], int n , int x) { // If x is present at first location itself if (arr [0] == x) return 0 ; // Find range for binary search by // repeated doubling int i = 1 ; while (i < n && arr [i] <= x) i = i * 2 ; // Call binary search for the found range. return binarySearch (arr , i / 2 , min (i , n -1), x); } // A recursive binary search function. It returns // location of x in given array arr[l..r] is // present, otherwise -1 int binarySearch (int arr [], int l , int r , int x) { if (r >= l) { int mid = l + (r - l) / 2 ; // If the element is present at the middle // itself if (arr [mid] == x) return mid ; // If element is smaller than mid, then it // can only be present n left subarray if (arr [mid] > x) return binarySearch (arr , l , mid -1 , x); // Else the element can only be present // in right subarray return binarySearch (arr , mid + 1 , r , x); } // We reach here when element is not present // in array return -1 ; } // Driver code int main (void) { int arr [] = { 2 , 3 , 4 , 10 , 40 }; int n = sizeof (arr) / sizeof (arr [0]); int x = 10 ; int result = exponentialSearch (arr , n , x); (result == -1) ? printf ("Element is not present in array ") : printf ("Element is present at index % d ", result); return 0 ; } Java // Java program to // find an element x in a // sorted array using // Exponential search. import java.util.Arrays ; class GFG { // Returns position of // first occurrence of // x in array static int exponentialSearch (int arr [] , int n , int x) { // If x is present at first location itself if (arr [0] == x) return 0 ; // Find range for binary search by // repeated doubling int i = 1 ; while (i < n && arr [i] <= x) i = i * 2 ; // Call binary search for the found range. return Arrays . binarySearch (arr , i / 2 , Math . min (i , n - 1), x); } // Driver code public static void main (String args []) { int arr [] = { 2 , 3 , 4 , 10 , 40 }; int x = 10 ; int result = exponentialSearch (arr , arr . length , x); System . out . println ((result < 0) ? "Element is not present in array" : "Element is present at index " + result); } } Python # Python program to find an element x # in a sorted array using Exponential Search # A recursive binary search function returns # location of x in given array arr[l..r] is # present, otherwise -1 def binarySearch (arr , l , r , x): if r >= l : mid = l + (r - l) // 2 # If the element is present at # the middle itself if arr [mid] == x : return mid # If the element is smaller than mid, # then it can only be present in the # left subarray if arr [mid] > x : return binarySearch (arr , l , mid - 1 , x) # Else he element can only be # present in the right return binarySearch (arr , mid + 1 , r , x) # We reach here if the element is not present return - 1 # Returns the position of first # occurrence of x in array def exponentialSearch (arr , n , x): # IF x is present at first # location itself if arr [0] == x : return 0 # Find range for binary search # j by repeated doubling i = 1 while i < n and arr [i] <= x : i = i * 2 # Call binary search for the found range return binarySearch (arr , i // 2 , min (i , n - 1), x) # Driver Code arr = [2 , 3 , 4 , 10 , 40] n = len (arr) x = 10 result = exponentialSearch (arr , n , x) if result == - 1 : print ("Element not found in the array") else : print ("Element is present at index %d " % (result)) # This code is contributed by Harshit Agrawal C# // C# program to find an element x in a // sorted array using Exponential search. using System ; class GFG { // Returns position of first // occurrence of x in array static int exponentialSearch (int [] arr , int n , int x) { // If x is present at // first location itself if (arr [0] == x) return 0 ; // Find range for binary search // by repeated doubling int i = 1 ; while (i < n && arr [i] <= x) i = i * 2 ; // Call binary search for // the found range. return binarySearch (arr , i / 2 , Math . Min (i , n - 1), x); } // A recursive binary search // function. It returns location // of x in given array arr[l..r] is // present, otherwise -1 static int binarySearch (int [] arr , int l , int r , int x) { if (r >= l) { int mid = l + (r - l) / 2 ; // If the element is present // at the middle itself if (arr [mid] == x) return mid ; // If element is smaller than // mid, then it can only be // present n left subarray if (arr [mid] > x) return binarySearch (arr , l , mid - 1 , x); // Else the element can only // be present in right subarray return binarySearch (arr , mid + 1 , r , x); } // We reach here when element // is not present in array return - 1 ; } // Driver code public static void Main () { int [] arr = { 2 , 3 , 4 , 10 , 40 }; int n = arr . Length ; int x = 10 ; int result = exponentialSearch (arr , n , x); if (result == - 1) Console . Write ("Element is not present in array "); else Console . Write ("Element is present at index " + result); } } // This code is contributed by Smitha JavaScript < script > // Javascript program to find an element x // in a sorted array using Exponential Search // A recursive binary search // function. It returns location // of x in given array arr[l..r] is // present, otherwise -1 function binarySearch (arr , l , r , x) { if (r >= l) { let mid = l + (r - l) / 2 ; // If the element is present // at the middle itself if (arr [mid] == x) return mid ; // If element is smaller than // mid, then it can only be // present n left subarray if (arr [mid] > x) return binarySearch (arr , l , mid - 1 , x); // Else the element can only // be present in right subarray return binarySearch (arr , mid + 1 , r , x); } // We reach here when element // is not present in array return - 1 ; } // Returns position of first // occurrence of x in array function exponentialSearch (arr , n , x) { // If x is present at // first location itself if (arr [0] == x) return 0 ; // Find range for binary search // by repeated doubling let i = 1 ; while (i < n && arr [i] <= x) i = i * 2 ; // Call binary search for // the found range. return binarySearch (arr , i / 2 , Math . min (i , n - 1), x); } // Driver Code let arr = [2 , 3 , 4 , 10 , 40]; let n = arr . length ; let x = 10 ; let result = exponentialSearch (arr , n , x); if (result == - 1) document . write ("Element is not present in array"); else document . write ("Element is present at index " + result); < /script> PHP <?php // PHP program to find an element x in a // sorted array using Exponential search. // Returns position of first // occurrence of x in array function exponentialSearch ($arr , $n , $x) { // If x is present at // first location itself if ($arr [0] == $x) return 0 ; // Find range for binary search // by repeated doubling $i = 1 ; while ($i < $n and $arr [$i] <= $x) $i = $i * 2 ; // Call binary search // for the found range. return binarySearch ($arr , $i / 2 , min ($i , $n - 1), $x); } // A recursive binary search // function. It returns location // of x in given array arr[l..r] is // present, otherwise -1 function binarySearch ($arr , $l , $r , $x) { if ($r >= $l) { $mid = $l + ($r - $l) / 2 ; // If the element is // present at the middle // itself if ($arr [$mid] == $x) return $mid ; // If element is smaller // than mid, then it // can only be present // n left subarray if ($arr [$mid] > $x) return binarySearch ($arr , $l , $mid - 1 , $x); // Else the element // can only be present // in right subarray return binarySearch ($arr , $mid + 1 , $r , $x); } // We reach here when // element is not present // in array return - 1 ; } // Driver code $arr = array (2 , 3 , 4 , 10 , 40); $n = count ($arr); $x = 10 ; $result = exponentialSearch ($arr , $n , $x); if ($result == - 1) echo "Element is not present in array" ; else echo "Element is present at index " , $result ; // This code is contributed by anuj_67 ?> Output Element is present at index 3 Iterative Implementation - O(Log n) Time and O(1) Space Here we use iterative implementation of binary search with the same approach. C++ #include <bits/stdc++.h> using namespace std ; int exponential_search (vector < int > arr , int x){ int n = arr . size (); if (n == 0) return -1 ; // Find range for binary search by repeatedly doubling i int i = 1 ; while (i < n and arr [i] < x) i *= 2 ; // Perform binary search on the range [i/2, min(i, n-1)] int left = i / 2 ; int right = min (i , n -1); while (left <= right){ int mid = (left + right) / 2 ; if (arr [mid] == x) return mid ; else if (arr [mid] < x) left = mid + 1 ; else right = mid - 1 ; } return -1 ; } // Driver Code int main (){ vector < int > arr = { 2 , 3 , 4 , 10 , 40 }; int n = arr . size (); int x = 10 ; int result = exponential_search (arr , x); if (result == -1){ cout << "Element not found in the array" ; } else { cout << "Element is present at index " << result << endl ; } return 0 ; } // This code is contributed by Ajay singh Java // Java implementation of above approach import java.util.* ; class Main { // Exponential search function public static int exponential_search (ArrayList < Integer > arr , int x) { int n = arr . size (); if (n == 0) return - 1 ; // Find range for binary search by repeatedly doubling i int i = 1 ; while (i < n && arr . get (i) < x) i *= 2 ; // Perform binary search on the range [i/2, min(i, n-1)] int left = i / 2 ; int right = Math . min (i , n - 1); while (left <= right) { int mid = (left + right) / 2 ; // finding mid if (arr . get (mid) == x) return mid ; else if (arr . get (mid) < x) left = mid + 1 ; else right = mid - 1 ; } return - 1 ; } // Driver Code public static void main (String [] args) { ArrayList < Integer > arr = new ArrayList <> (Arrays . asList (2 , 3 , 4 , 10 , 40)); int n = arr . size (); int x = 10 ; int result = exponential_search (arr , x); if (result == - 1) { System . out . println ("Element not found in the array"); } else { System . out . println ("Element is present at index " + result); } } } Python def exponential_search (arr , x): n = len (arr) if n == 0 : return - 1 # Find range for binary search by repeatedly doubling i i = 1 while i < n and arr [i] < x : i *= 2 # Perform binary search on the range [i/2, min(i, n-1)] left = i // 2 right = min (i , n - 1) while left <= right : mid = (left + right) // 2 if arr [mid] == x : return mid elif arr [mid] < x : left = mid + 1 else : right = mid - 1 return - 1 # Driver Code arr = [2 , 3 , 4 , 10 , 40] n = len (arr) x = 10 result = exponential_search (arr , x) if result == - 1 : print ("Element not found in the array") else : print ("Element is present at index %d " % (result)) # This code is contributed by Ajay singh C# // C# Program for the above approach using System ; using System.Collections.Generic ; class Program { static int exponential_search (List < int > arr , int x) { int n = arr . Count ; if (n == 0) return - 1 ; // Find range for binary search by repeatedly doubling i int i = 1 ; while (i < n && arr [i] < x) i *= 2 ; // Perform binary search on the range [i/2, min(i, n-1)] int left = i / 2 ; int right = Math . Min (i , n - 1); while (left <= right) { int mid = (left + right) / 2 ; if (arr [mid] == x) return mid ; else if (arr [mid] < x) left = mid + 1 ; else right = mid - 1 ; } return - 1 ; } static void Main (string [] args) { List < int > arr = new List < int > { 2 , 3 , 4 , 10 , 40 }; int n = arr . Count ; int x = 10 ; int result = exponential_search (arr , x); if (result == - 1) { Console . WriteLine ("Element not found in the array"); } else { Console . WriteLine ("Element is present at index " + result); } } } // This code is contributed by princekumaras JavaScript function exponential_search (arr , x) { const n = arr . length ; if (n == 0) { // if array is empty, return -1 return - 1 ; } let i = 1 ; while (i < n && arr [i] < x) { // Find the range for binary search by repeatedly doubling i i *= 2 ; } const left = Math . floor (i / 2); // Set left boundary for binary search const right = Math . min (i , n - 1); // Set right boundary for binary search while (left <= right) { // Perform binary search on the range [i/2, min(i, n-1)] const mid = Math . floor ((left + right) / 2); // Find middle index if (arr [mid] == x) { // If element is found at mid index, return mid return mid ; } else if (arr [mid] < x) { // If element is less than mid index, search the right half of array left = mid + 1 ; } else { // If element is greater than mid index, search the left half of array right = mid - 1 ; } } return - 1 ; // If element is not found in array, return -1 } // Driver Code const arr = [2 , 3 , 4 , 10 , 40]; const n = arr . length ; const x = 10 ; const result = exponential_search (arr , x); if (result == - 1) { console . log ("Element not found in the array"); } else { console . log (`Element is present at index ${ result } `); } Output Element is present at index 3 Applications of Exponential Search: Exponential Binary Search is particularly useful for unbounded searches, where size of array is infinite. Please refer Unbounded Binary Search for an example. It works better than Binary Search for bounded arrays when the element to be searched is closer to the first element. Comment Article Tags: Article Tags: Searching DSA Binary Search