Fibonacci Search - GeeksforGeeks

Source: https://www.geeksforgeeks.org/fibonacci-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 Fibonacci Search Last Updated : 29 Apr, 2025 Given a sorted array arr[] of size n and an integer x. Your task is to check if the integer x is present in the array arr[] or not. Return index of x if it is present in array else return -1 . Examples: Input: arr[] = [2, 3, 4, 10, 40], x = 10 Output: 3 Explanation: 10 is present at index 3. Input: arr[] = [2, 3, 4, 10, 40], x = 11 Output: -1 Explanation: 11 is not present in the given array. What is Fibonacci Search? Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array. Similarities of Fibonacci Search with Binary Search Works for sorted arrays A Divide and Conquer Algorithm. Has Log n time complexity. Differences of Fibonacci Search with Binary Search Fibonacci Search divides given array into unequal parts Binary Search uses a division operator to divide range. Fibonacci Search doesn't use /, but uses + and -. The division operator may be costly on some CPUs. Fibonacci Search examines relatively closer elements in subsequent steps. So when the input array is big that cannot fit in CPU cache or even in RAM, Fibonacci Search can be useful. Background Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1. First few Fibonacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ... Below observation is used for range elimination, and hence for the O(log(n)) complexity. F(n - 2) ≈ (1/3)*F(n) and (Note : This is an approximation, not equal) F(n - 1) ≈ (2/3)*F(n). Fibonacci Search Algorithm - O(log n) Time and O(1) Space The idea is to first find the smallest Fibonacci number that is greater than or equal to the length of the given array. Let the found Fibonacci number be fib (m’th Fibonacci number). We use (m-2)’th Fibonacci number as the index. Let (m-2)’th Fibonacci Number be i, we compare arr[i] with x, if x is same, we return i. Else if x is greater, we recur for subarray after i, else we recur for subarray before i. Let arr[0..n-1] be the input array and the element to be searched be x. Find the smallest Fibonacci Number greater than or equal to n. Let this number be fibM [m’th Fibonacci Number]. Let the two Fibonacci numbers preceding it be fibMm1 [(m-1)’th Fibonacci Number] and fibMm2 [(m-2)’th Fibonacci Number]. While the array has elements to be inspected: Compare x with the last element of the range covered by fibMm2 (variable a in the below code). If x matches, return index Else If x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array. Else x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate the elimination of approximately front one-third of the remaining array. Since there might be a single element remaining for comparison, check if fibMm1 is 1. If Yes, compare x with that remaining element. If match, return index C++ #include <bits/stdc++.h> using namespace std ; // Returns index of x if present, else returns -1 int search (vector < int > & arr , int x) { int n = arr . size (); // initialize first three fibonacci numbers int a = 0 , b = 1 , c = 1 ; // iterate while c is smaller than n // c stores the smallest Fibonacci // number greater than or equal to n while (c < n) { a = b ; b = c ; c = a + b ; } // marks the eliminated range from front int offset = -1 ; // while there are elements to be inspected // Note that we compare arr[a] with x. // When c becomes 1, a becomes 08 while (c > 1) { // check if a is a valid location int i = min (offset + a , n - 1); // if x is greater than the value at index a, // cut the subarray array from offset to i if (arr [i] < x) { c = b ; b = a ; a = c - b ; offset = i ; } // else if x is greater than the value at // index a,cut the subarray after i+1 else if (arr [i] > x) { c = a ; b = b - a ; a = c - b ; } // else if element found, return index else return i ; } // comparing the last element with x if (b && arr [offset + 1] == x) return offset + 1 ; // element not found, return -1 return -1 ; } int main () { vector < int > arr = { 2 , 3 , 4 , 10 , 40 }; int x = 10 ; cout << search (arr , x); return 0 ; } Java import java.io.* ;

class GfG { // Returns index of x if present, else returns -1 public static int search (int [] arr , int x) { int n = arr . length ; // initialize first three fibonacci numbers int a = 0 , b = 1 , c = 1 ; // iterate while c is smaller than n // c stores the smallest Fibonacci // number greater than or equal to n while (c < n) { a = b ; b = c ; c = a + b ; } // marks the eliminated range from front int offset = - 1 ; // while there are elements to be inspected // Note that we compare arr[a] with x. // When c becomes 1, a becomes 08 while (c > 1) { // check if a is a valid location int i = Math . min (offset + a , n - 1); // if x is greater than the value at index a, // cut the subarray array from offset to i if (arr [i] < x) { c = b ; b = a ; a = c - b ; offset = i ; } // else if x is greater than the value at // index a,cut the subarray after i+1 else if (arr [i] > x) { c = a ; b = b - a ; a = c - b ; } // else if element found, return index else return i ; } // comparing the last element with x if (b != 0 && arr [offset + 1] == x) return offset + 1 ; // element not found, return -1 return - 1 ; } public static void main (String [] args) { int [] arr = { 2 , 3 , 4 , 10 , 40 }; int x = 10 ; System . out . println (search (arr , x)); } } Python #!/usr/bin/env python3 # Returns index of x if present, else returns -1 def search (arr , x): n = len (arr) # initialize first three fibonacci numbers a = 0 b = 1 c = 1 # iterate while c is smaller than n # c stores the smallest Fibonacci # number greater than or equal to n while c < n : a = b b = c c = a + b # marks the eliminated range from front offset = - 1 # while there are elements to be inspected # Note that we compare arr[a] with x. # When c becomes 1, a becomes 08 while c > 1 : # check if a is a valid location i = min (offset + a , n - 1) # if x is greater than the value at index a, # cut the subarray array from offset to i if arr [i] < x : c = b b = a a = c - b offset = i # else if x is greater than the value at # index a,cut the subarray after i+1 elif arr [i] > x : c = a b = b - a a = c - b # else if element found, return index else : return i # comparing the last element with x if b and arr [offset + 1] == x : return offset + 1 # element not found, return -1 return - 1 if __name__ == "__main__" : arr = [2 , 3 , 4 , 10 , 40] x = 10 print (search (arr , x)) C# using System ; class GfG { // Returns index of x if present, else returns -1 public static int search (int [] arr , int x) { int n = arr . Length ; // initialize first three fibonacci numbers int a = 0 , b = 1 , c = 1 ; // iterate while c is smaller than n // c stores the smallest Fibonacci // number greater than or equal to n while (c < n) { a = b ; b = c ; c = a + b ; } // marks the eliminated range from front int offset = - 1 ; // while there are elements to be inspected // Note that we compare arr[a] with x. // When c becomes 1, a becomes 08 while (c > 1) { // check if a is a valid location int i = Math . Min (offset + a , n - 1); // if x is greater than the value at index a, // cut the subarray array from offset to i if (arr [i] < x) { c = b ; b = a ; a = c - b ; offset = i ; } // else if x is greater than the value at // index a,cut the subarray after i+1 else if (arr [i] > x) { c = a ; b = b - a ; a = c - b ; } // else if element found, return index else return i ; } // comparing the last element with x if (b != 0 && arr [offset + 1] == x) return offset + 1 ; // element not found, return -1 return - 1 ; } public static void Main () { int [] arr = { 2 , 3 , 4 , 10 , 40 }; int x = 10 ; Console . WriteLine (search (arr , x)); } } JavaScript // Returns index of x if present, else returns -1 function search (arr , x) { let n = arr . length ; // initialize first three fibonacci numbers let a = 0 , b = 1 , c = 1 ; // iterate while c is smaller than n // c stores the smallest Fibonacci // number greater than or equal to n while (c < n) { a = b ; b = c ; c = a + b ; } // marks the eliminated range from front let offset = - 1 ; // while there are elements to be inspected // Note that we compare arr[a] with x. // When c becomes 1, a becomes 08 while (c > 1) { // check if a is a valid location let i = Math . min (offset + a , n - 1); // if x is greater than the value at index a, // cut the subarray array from offset to i if (arr [i] < x) { c = b ; b = a ; a = c - b ; offset = i ; } // else if x is greater than the value at // index a,cut the subarray after i+1 else if (arr [i] > x) { c = a ; b = b - a ; a = c - b ; } // else if element found, return index else return i ; } // comparing the last element with x if (b && arr [offset + 1] === x) return offset + 1 ; // element not found, return -1 return - 1 ; } let arr = [2 , 3 , 4 , 10 , 40]; let x = 10 ; console . log (search (arr , x)); Output 3 Illustration: Let us understand the algorithm with the below example: Illustration assumption: 1-based indexing. Target element x is 85. Length of array n = 11. Smallest Fibonacci number greater than or equal to 11 is 13. As per our illustration, fibMm2 = 5, fibMm1 = 8, and fibM = 13. Another implementation detail is the offset variable (zero-initialized). It marks the range that has been eliminated, starting from the front. We will update it from time to time. Now since the offset value is an index and all indices including it and below it have been eliminated, it only makes sense to add something to it. Since fibMm2 marks approximately one-third of our array, as well as the indices it marks, are sure to be valid ones, we can add fibMm2 to offset and check the element at index i = min(offset + fibMm2, n). Visualization: Time Complexity analysis: The worst-case will occur when we have our target in the larger (2/3) fraction of the array, as we proceed to find it. In other words, we are eliminating the smaller (1/3) fraction of the array every time. We call once for n, then for(2/3) n, then for (4/9) n, and henceforth. Consider that: Auxiliary Space: O(1) Comment Article Tags: Article Tags: Searching DSA Fibonacci