

Search in an Infinite Sized Array

Suppose you have a sorted array of infinite numbers, how would you search an element in the array?





If the array is infinite, that means we don't have proper bounds to apply binary search. So in order to find position of key, first we find bounds and then apply binary search algorithm.

Let low be pointing to 1st element and high pointing to 2nd element of array, Now compare key with high index element,

- ->if it is greater than high index element then copy high index in low index and double the high index.
- ->if it is smaller, then apply binary search on high and low indices found.

Below are implementations of above algorithm

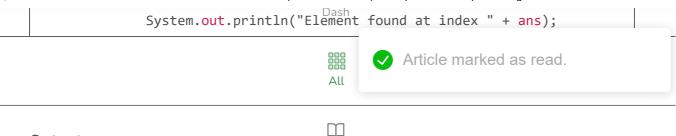
```
C++
        lava
 // Java program to demonstrate working of
// an algorithm that finds an element in an
 // array of infinite size
 class Test
 {
     // Simple binary search algorithm
     static int binarySearch(int arr[], int 1, int r, int x)
         if (r>=1)
         {
             int mid = 1 + (r - 1)/2;
             if (arr[mid] == x)
                 return mid;
             if (arr[mid] > x)
                 return binarySearch(arr, 1, mid-1, x);
```

Track Progress

28 of 60 Complete. (47%)

```
Article marked as read.
                                            ΑII
            // We don't know size of arr[] and we can assume size to be
            // infinite in this function.
            // NOTE THAT THIS FUNCTION ASSUMES arr[] TO BE OF INFINITE SIZE
            // THEREFORE, THERE IS NO INDEX_OUT OF BOUND CHECKING
            static int findPos(int arr[],int key)
            {
                int 1 = 0, h = 1;
                                            </>>
                int val = arr[0];
                                          Problems
                // Find h to do binary search
                while (val < key)
                                            Quiz
                {
                    1 = h; // store previous high
                    //check that 2*h doesn'tvexceeds array
                    //length to prevent ArrayOutOfBoundException
                    if(2*h < arr.length-1)</pre>
                    h = 2*h;
                    else
                    h = arr.length-1;
                    val = arr[h]; // update new val
                }
                // at this point we have updated low
                // and high indices, thus use binary
                // search between them
                return binarySearch(arr, 1, h, key);
            }
            // Driver method to test the above function
            public static void main(String[] args)
                int arr[] = new int[]{3, 5, 7, 9, 10, 90,}
                                     100, 130, 140, 160, 170};
                int ans = findPos(arr,10);
Menu
                if (ans==-1)
Track Progress
```

28 of 60 Complete. (47%)



Output

Articles

Element found at index 4



Let p be the position of element to be searched. Number of steps for finding high index 'h' is O(Log p). The value of 'h' must be less than 2*p. The number of elements between h/2 and h must be O(p). Therefore, time complexity of Binary Search step is also O(Log p) and overall time complexity is 2*O(Log p) which is O(Log p).



Approach: The problem can be solved based on the following observation:

- Since array is sorted we apply binary search but the length of array is infinite so that we take **start = 0** and **end** £ 1 test
- After that check value of target is greater than the value at end index,if it is true then change newStart = end + 1 and
 newEnd = end +(end start +1)*2 and apply binary search.
- Otherwise , apply binary search in the old index values.

Below are implementations of above algorithm:

```
C++ Java

// C++ program for the above approach
#include <bits/stdc++.h>
using namespace std;

// Simple binary search algorithm
int binarySearch(int arr[], int target, int start, int end)
{
    while (start <= end) {
    int mid = start + (end - start) / 2;

Menu

if (target < arr[mid]) {
    end = mid - 1;</pre>
```



Track Progress

28 of 60 Complete. (47%)

```
Article marked as read.
                                               All
                   }
                                               \mathbf{m}
                                              Articles
              return -1;
          }
                                               Videos
          // an algorithm that finds an element in an
          // array of infinite size
                                             Problems
          int findPos(int arr[], int target)
          {
                                               Quiz
              // first find the range
              // first start with a box of size 2
              int start = 0;
                                              Contest
              int end = 1;
              // condition for the target to lie in the range
              while (target > arr[end]) {
                   int temp = end + 1; // this is my new start
                   // double the box value
                   // end = previous end + sizeofbox*2
Courses
Tutorials
Jobs
              }
              return binarySearch(arr, target, start, end);
Practice
Contests
          // Driver code
          int main()
          {
              int arr[]
                   = { 3, 5, 7, 9, 10, 90, 100, 130, 140, 160, 170 };
              int target = 10;
              // Function call
  Menu
              int ans = findPos(arr, target);
              if (ans == -1)
  Track Progress
  28 of 60 Complete. (47%)
```

