Binary Search

Easily one of the most extensively used computer algorithms.

Input: An array of sorted elements and an element "x" to be searched for in this array

output: "No" if x is not present in the array; "Yes" if x is present along with the index in the array at which x is present.

We assume for simplicity that the input array consists of distinct integers. In general it could consist of more complex elements such as strings.

The idea of the algorithm is to use the divide-and-conquer paradigm. The idea exploits the sorted nature of the array and reduces the size of the search space by half each time.

The essence of the idea is the following:

repeat

examine middle value of remaining data and eliminate half of the remaining data set

until (x is found or established that x is not present)

0	1	2	3	4	5	6	7	8	9
-10	-5	5	8	20	25	52	60	71	85

Initially we set low = 0 and high = 9 and compute the middle index as integer part of

$$\frac{low + high}{2}$$

steps if $x = 92$

(1) low = 0; high = 9; mid = 4

(2) a[mid] = 20 < 92
low = 5; high = 9; mid = 7

(3) a[mid] = 60 < 92
low = 8; high = 9; mid = 8

(4) a[mid] = 71 < 92
low = 9; high = 9; mid = 9

(5) a[mid] = 85 ≠ 92

steps if $x = -5$

(1) low = 0; high = 9; mid = 4

(2) a[mid] = 20 > −5
low = 0; high = 3; mid = 1

(3) a[mid] = −5
Thus −5 is present at index = 1

Pseudo-code for Binary Search

Algorithm (Binary Search)

Input : $a[0:n-1]$, an array consisting of ordered elements; and an element x to be searched for

output : "No" if x is not present and "yes" along with index if present

1. Initialize : low = 0 ; high = n − 1
2. mid = integer part of $\frac{low + high}{2}$
3. if $(a[mid] = x)$ or $(low > high)$ goto step 5
4. If $(x > a[mid])$ then low = mid + 1 else high = mid − 1 ; goto step 2
5. If $(a[mid] = x)$ then print (x is present at index = mid) else print (x is not present)
6. End

Excercise : Write a flow-chart for the above pseudocode.

Recursive Algorithm for Binary Search

Binary search is a naturally recursive algorithm since each time we reduce the search space by half and repeat the binary search on the reduced search space.

binarySearch (int a[], x, low, high)

1. mid = integer part of $\frac{low + high}{2}$;
2. if (x < a[mid]) then high = mid − 1;
   binarySearch (a, x, low, high)
3. if (x > a[mid]) then low = mid + 1;
   binarySearch (a, x, low, high)
4. if (x = a[mid]) then
   print (x is present at mid)
   else print (x is not present)

We invoke
binarySearch (a[], x, 0, n − 1)

Computational Complexity of Binary Search Algorithm

Suppose

$T(n)$ = number of steps (comparisons) required by binary search in the worst case

Let $n = 2^k$ where k is a non-negative integer

Then

$$T(1) = 1$$
$$T(n) = 1 + T\left(\frac{n}{2}\right)$$

In terms of k, this becomes

$$\begin{aligned} T(2^k) &= 1 + T(2^{k-1}) \\ &= 1 + (1 + T(2^{k-2})) \\ &= 2 + T(2^{k-2}) \\ &= 2 + (1 + T(2^{k-3})) \\ &= 3 + T(2^{k-3}) \\ &= \cdots \\ &= k + T(1) \\ &= k + 1 \\ &= \log_2 n + 1 \end{aligned}$$

Excercises

(1) Fibonacci Search

Binary search splits the array each time into approximately two equal portions. Fibonacci search is a variation of binary search that splits the array using fibonacci numbers.

- for example, if the array consists of 55 elements, fibonacci search splits the array into 21 and 34 element subarrays.; 34 is split into 21 and 13 element arrays, and so on.

Develop and implement iterative and recursive versions of fibonacci search.

(2) Searching in an Infinite array of 0's followed by 1's.

Consider an infinite array consisting of a string of unknown number of 0's followed by only 1's. It is required to find the index of the location in the array where the first 1 is present. Design and implement an algorithm for this problem which uses binary search or fibonacci search.

BSG