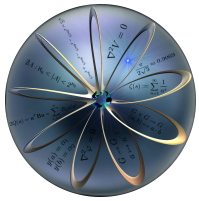

planetmath.org

Math for the people, by the people.

binary search

Canonical name	BinarySearch
Date of creation	2013-03-22 11:44:34
Last modified on	2013-03-22 11:44:34
Owner	mathcam (2727)
Last modified by	mathcam (2727)
Numerical id	18
Author	mathcam (2727)
Entry type	Algorithm
Classification	msc 68P10
Classification	msc 16S40
Classification	msc 20G42
Classification	msc 16W35
Classification	msc 81R50
Classification	msc 16W30
Classification	msc 57T05
Classification	msc 54-01
Related topic	TotalOrder
Related topic	PartialOrder
Related topic	SortingProblem
Related topic	InsertionSort

The Problem

Let $\leq$ be a total ordering on the set S. Given a sequence of n elements, $L = \{x_1 \leq x_2 \leq \cdots \leq x_n\}$, and a value $y \in S$, locate the position of any elements in L that are equal to y, or determine that none exist.

The Algorithm

The *binary search* technique is a fundamental method for locating an element of a particular value within a sequence of sorted elements (see Sorting Problem). The idea is to eliminate half of the search space with each comparison.

First, the middle element of the sequence is compared to the value we are searching for. If this element matches the value we are searching for, we are done. If, however, the middle element is "less than" the value we are chosen for (as specified by the `http://planetmath.org/Relation`relation used to specify a total order over the set of elements), then we know that, if the value exists in the sequence, it must exist somewhere *after* the middle element. Therefore we can eliminate the first half of the sequence from our search and simply repeat the search in the exact same manner on the remaining half of the sequence. If, however, the value we are searching for comes before the middle element, then we repeat the search on the first half of the sequence.

Pseudocode

Algorithm BINARY_SEARCH`(L, n, key)`
Input: A list L of n elements, and *key* (the search key)
Output: *Position* (such that $X[Position] = key$)
  begin
    $Position \leftarrow Find(L, 1, n, key)$;
end
function $Find(L, bottom, top, key)$
begin

```
    if bottom = top then
        if L[bottom] = key then
            Find ← bottom
        else
            Find ← 0
    else
        begin
            middle ← (bottom + top)/2;
            if key < L[middle] then
                Find ← Find(L, bottom, middle − 1, key)
            else
                Find ← Find(L, middle + 1, top, key)
        end
end
```

Analysis

We can specify the runtime complexity of this binary search algorithm by counting the number of comparisons required to locate some element y in L. Since half of the list is eliminated with each comparison, there can be no more than $\log_2 n$ comparisons before either the positions of all the y elements are found or the entire list is eliminated and y is determined to not exist in L. Thus the worst-case runtime complexity of the binary search is $\mathcal{O}(\log n)$. It can also be shown that the average-case runtime complexity of the binary search is approximately $\log_2 n - 1$ comparisons. This means that any single entry in a phone book containing one million entries can be located with at most 20 comparisons (and on average 19).