Binary Search to find item in sorted array

January 15, 2008

QUESTION: Suppose we are given a sorted list $A[1..n]$ (as an array), of n **real** numbers: $A[1] \leq A[2] \leq \cdots \leq A[n]$. Given a real number x, decide whether x is in the array, using **binary search**: compare with the middle, eliminate half the elements, etc. Write a short and elegant **pseudocode** for this algorithm. State the exact number of **comparisons between reals** your algorithm will make in the worst case. *Hint:* Solve the problem for all subarrays $A[a..b]$.

REMARK on cost. In the model described, comparison of reals is the only cost item. Arithmetic with integers as well as bookkeeping (maintaining pointers, copying integers) comes for free. We think of the comparisons of real data as being much more expensive than manipulation of addresses in the array. This simplification can be justified by practical considerations; comparison of data may require a physical experiment (such as scratching two precious stones to see which is harder or letting two teams play) or comparison of long strings. But such practical considerations are secondary to the *theoretical* significance of this simplification: (a) the model is now mathematically very simple; (b) once we minimized this critical cost element, the number of bookkeeping and integer arithmetic operations will be within a constant factor of this crucial quantity.

The solutions begin on the next page.

SOLUTION. We describe two solutions. **Study both solutions.**

First solution. First we give an explicit algorithm with no recursive calls to itself.

The procedure below will produce one of the following answers: "list empty," "not found," "found." In the last case it will also return an address i such that $x = A[i]$.

BinarySearch(A, a, b, x)

INPUT: a, b integers, $A[a..b]$ array of real numbers, x real number.

```
00  if $b < a$ then return NIL, exit     (list empty)
01  $\ell := a$, $u := b$ (initializing lower and upper limits of current range)
02  while $\ell < u$
03      $p := \lfloor (\ell + u)/2 \rfloor$     (pivot)
04      if $x \le A[p]$ then $u := p$
05          else $\ell := p + 1$
06  end(while)
07  if $x = A[\ell]$ then return $\ell$, exit     (item found)
08  return NIL, exit     (item not found)
```

Comments.

1. After the execution of line 01, the following statement holds:

   $$\text{If } x \text{ is on the list } A[a..b] \text{ then } A[\ell] \le x \le A[u].$$

   This statement is a *loop invariant,* i. e., if it holds before the algorithm enters a cycle of the **while** loop then it also holds after the execution of the cycle.

   (Why is this true? Consider carefully how lines 03, 04, 05 affect the validity of this statement. What happens if $A[p] < x < A[p + 1]$?)

2. The algorithm keeps halving the length $u - \ell$ of the current range $[\ell, u]$, rounding down if $u - \ell$ is odd. So eventually we arrive at the point where $u = \ell$. At that point we know by Comment 1 that either $x = A[\ell]$ or x is not on the list $A[a..b]$.

3. Note that Line 02 (comparison of two addresses) does not contribute to our count of comparisons: it does not relate to our data but rather their addresses, so it is essentially a bookkeeping operation.

The *correctness* of the algorithm follows from Comments 1 and 2. (Verify this statement!)

To calculate the *cost*, i. e., the number of comparisons, we observe that most of the comparisons occur on line 04, so we need to count how many times line 04 is executed, which is the same as the number of executions of the **while** loop.

Claim. If $b \geq a$ then the **while** loop is executed at most $\lceil \log(b-a+1) \rceil$ times (base 2 logarithm). This bound is tight (it is equal to the actual number of executions of the **while** loop in the worst case).

Proof. Let $m = u - \ell$ and let m_i denote the value of m after the i-th execution of the **while** loop. Clearly, $m_0 = b - a$ and $m_i \leq \lfloor m_{i-1}/2 \rfloor$. By induction on i we obtain that $(\forall i)(m_i = \lfloor m_0/2^i \rfloor)$. (Work out the inductive proof!)

Suppose the **while** loop is executed t times; then t is the smallest integer such that $m_t = 0$. Therefore t is not greater than the smallest integer k satisfying $\lfloor m_0/2^k \rfloor = 0$, i. e., $m_0/2^k < 1$, i. e., $m_0 < 2^k$, i. e., $m_0 + 1 \leq 2^k$, i. e., $\log(m_0 + 1) \leq k$. Since k is the smallest such integer, we conclude that

$$k = \lceil \log(m_0 + 1) \rceil.$$

It follows that $k \leq t$, which proves the Claim. To see that this bound is tight, observe what happens if $x \leq A[a]$. Show that in this case $(\forall i)(m_i = \lfloor m_{i-1}/2 \rfloor)$. Therefore in this case $t = k$. Note that this is one of those very rare cases when we can describe a "worst case" input.

To count the total number of comparisons, we need to add 1 to the above tally for line 07. So **the total cost of the algorithm is**

$$\lceil \log(b - a + 1) \rceil + 1.$$

Second solution. We state the same algorithm as a *recursive algorithm*, i. e., an algorithm which makes calls to itself on smaller instances.

RecursiveBinarySearch(A, a, b, x)

INPUT: a, b integers, $A[a..b]$ array of real numbers, x real number.

```
20   **if** $b < a$ **then return** NIL, **exit**     (list empty)
21   RBS($A, a, b, x$)
```

The recursive part of the algorithm is the procedure RBS to be described next.

RBS(A, a, b, x)

INPUT: as above; we assume $b \geq a$. (The validity of this assumption is guaranteed since we are past line 20.)

```
23   if $b = a$ then
24           if $x = A[a]$ then return $a$, exit     (item found)
25               else return NIL, exit     (item not found)
26       else             (now $b - a \geq 1$)
27           $p := \lfloor (a+b)/2 \rfloor$       (pivot)
28           if $x \leq A[p]$ then $b := p$
29               else $a := p + 1$
30           RBS($A, a, b, x$)
```

Comment. This algorithm does essentially the same as the one in the first solution. But the natural analysis is different; it requires the **solution of a recurrence** for the cost function.

Analysis. Let $C(m)$ denote the number of comparisons required by the RBS algorithm in the worst case, where $m = b - a \geq 0$. Note that $m + 1$ is the number of items in the array; we use the parameter m for convenience.

We observe that $C(0) = 1$ (line 24) and for $m \geq 1$,

$$C(m) = 1 + C(\lfloor m/2 \rfloor).$$

Indeed, the cost $C(m)$ consists of the cost of one comparison (line 28) plus the cost $C(\lfloor m/2 \rfloor)$ of solving the same problem on a half-size instance (line 30).

Now, by induction on i, we obtain that for all i,

$$C(m) = i + C(\lfloor m/2^i \rfloor).$$

(Work out the induction!)

Let k be the smallest integer such that $m/2^k < 1$. As in the first solution, this means that $k = \lceil \log(m+1) \rceil$.

On the other hand, now k is the number of recursive calls (line 30). Therefore the cost of the algorithm is

$$C(m) = k + C(\lfloor m/2^k \rfloor) = k + C(0) = k + 1 = \lceil \log(b - a + 1) \rceil + 1.$$

This agrees with the cost of the algorithm given in the first solution.

Last updated 1-22-2014