Binary search runs in $O(\log n)$ time.

Michael George

Tuesday March 29, 2005

This is a proof that binary search runs in $O(\log n)$ time. Here is the code:

```
BINSEARCH (A, x, a, b)
    if b = a then
        return false
    m ← \(\frac{b-a}{2} + a\)
    if A[m] > x then
        return BINSEARCH (A, x, a, m)
    else if A[m] = x then
        return true
    else if A[m] < x then
        return BINSEARCH (A, x, m, b)
```

Let $C$ be the amount of required to run all of the code in the procedure except for the two recursive calls, and let $T(n)$ be the total amount of time required to run the procedure when $b - a = n$. I claim that $T(n) \leq C \log n + T(1)$ for all $n \geq 1$.

I will prove this by strong induction. The base case (when $n = 1$) is clear:

$$C \log 1 + T(1) = 0 + T(1) = T(1)$$

Now, choose a particular $n > 1$. For our inductive hypothesis we will assume that for all $k < n$, that $T(k) \leq C \log k + T(1)$.

How long does BINSEARCH take to run if $b - a = n$? Well, there are three possibilities: we could take the first branch of the if statement (i.e. $A[m] > x$), we could take the second branch ($A[m] = x$), or we could take the third branch ($A[m] < x$).
In the first of these possibilities, we need at most $C$ time to execute everything other than the recursive calls, and we’ll need $T(m-a)$ time to do the recursive call. So:

$$T(n) \leq C + T(m-a) = C + T\left(\frac{b-a}{2} + a - a\right) = C + T\left(\frac{b-a}{2}\right)$$

*By our inductive hypothesis, since $\frac{b-a}{2} < b-a$, we can reduce this to*

$$T(n) \leq C + T\left(\frac{b-a}{2}\right) \leq C + \left(C \log \left(\frac{b-a}{2}\right) + T(1)\right) = C + C \log (b-a) - C + T(1) = C \log (n) + T(1)$$

If we’re in the second case, and we don’t make any recursive calls, then all we do is return true. In this case, we take at most $C$ amount of time, and since $n > 1$,

$$T(n) \leq C \leq C \log n + T(1)$$

Finally, we could take the third branch (i.e. $A[m]$ could be less than $x$). In this case the total amount of time will be $T(n) \leq C + T(b - m)$. Since

$$b - m = b - \left(\frac{b-a}{2} + a\right) = \frac{b-a}{2}$$

we see that $T(n) = C + T\left(\frac{b-a}{2}\right)$ so this case works out exactly like the first case.

Since these are all possible executions, and in all three cases we have used up at most $C \log n + T(1)$ time, we have shown that $T(n) \leq C \log n + T(1)$ by strong induction.

At this point, we see that if $n > 1$, that $T(n) \leq C \log n + T(1)$. Does this show that $T(n)$ is $O(\log n)$? The answer is yes, but it’s a little work. We want to find a witness pair $(c, n_0)$ such that for all $n > n_0$, $T(n)$ is less
than c log n. We can guarantee this if we just force $C \log n + T(1) < c \log n$
since we know that $T(n)$ is less than or equal to $C \log n + T(1)$. We’ll start
by choosing $c$ bigger than $C$, say $C + 1$. Then we can solve:

$$
C \log n + T(1) < c \log n
\iff
T(1) < (c - C) \log n
\iff
T(1) < \log n
$$

So as long as $n > 2^{T(1)}$, we see that $T(n) < c \log n$. Thus, our witness pair is
$\langle C + 1, 2^{T(1)} \rangle$, and we can conclude that $T(n)$ is $O(\log n)$. 

3