Binary Search (II)

CSCD 300 – Data Structures

Eastern Washington University

Outline

1. How to measure the speed of an program (or algorithm) ?

2. The time complexity of the recursion-based Binary Search

3. The time complexity of the iteration-based binary search

How to measure the speed of an program (or algorithm) ?

A few common agreements

- We just count how many constant-time operations (or steps) are executed.
- We are often more interested in that count in the worst case. Namely, the worst-case time complexity.
- Constant-time operations include: add, sub, multi, div, modulo, assignment, comparison, ...
- A constant-time operation can also be a constant number of contant-time operations, because a contant times another contant is still a constant.
- We are often more concerned with the most significant term in the expression of a time complexity and ignore the leading constant factors.

Let's look at a concrete example

The time complexity of the Binary Search ...

The time complexity of the recursion-based Binary Search

```
/* Find a location in A[low...high] whose value is equal to x */
binary_search_recursion(A, low, high, x)
{
   if(low > high)  return -1          -----+
                                           | All these together costs
   mid = floor((low+high)/2)               | a contant amount of time,
   if(A[mid] == x)                         | denoted by c.
      return mid                           |
   else if(A[mid] < x)                -----+
      return binary_search_recursion(A, mid+1, high, x)
   else
      return binary_search_recursion(A, low, mid-1, x)
}
```

Initial call: `binary_search_recursion(A, 0, n-1, x)`, where $n = A.length$.

Let $T(n)$ denote the time cost of `binary_search_recursion(A, 0, n-1, x)`.
Then, $T(n)$ be expressed as:

$$T(n) \leq c + T(n/2)$$

(continue ...)

$$\begin{aligned}
T(n) &\le c + T(n/2) \le c + (c + T(n/4)) \\
&= c + c + T(n/4) \le c + c + (c + T(n/8)) \\
&= c + c + c + T(n/8) \\
&\vdots \\
&= \underbrace{c + c + \ldots + c}_{\log_2 n \text{ terms (why ?)}} + T(1) \\
&= c \log_2 n + c' = O(\log n)
\end{aligned}$$

- Every recursive function call reduces the problem size to a half, so the total number of recursions that can happen is no more than $\log_2 n$.
- The final recursive function call that can happen is a binary search on an array of size 1, for which the time cost is no more than another constant, denoted as c'.
- We ignore the contant term c' and the leading constat factor c. That gives us the knowledge that the time cost of the recursion-based binary search is no more than **the order of** $\log n$.
- You can say the same thing in different ways: (1) the time cost of the recursion-based binary search is upper bounded by $\log n$, or (2) the time complexity of the recursion-based binary search is $O(\log n)$.

Some notes about the big-oh notation

- The introduction of the big-oh notation in the previous slide is not formal and precise. It only gives you an intuitive understanding of what it is and how to use it.
- The precise definition of the big-oh notation will be given in the CSCD320 course, where several other asymptotic notations will also be precisely defined and used.
- Inside the big-oh notation (as well as other asymptotic notations), we often do not need to give the base of the logarithmic, because

$$\begin{aligned} O(\log_a f(n)) &= O\left(\frac{\log f(n)}{\log a}\right) && \text{(property of logarithmic)} \\ &= O(\log f(n)) && \left(\text{because } \frac{1}{\log a} \text{ is a constant}\right) \end{aligned}$$

for any function $f(n)$ and any constant base a.

The time complexity of the iteration-based binary search

```
binary_search_iteration(A, x)
{
   low = 0; high = A.lengh - 1;   ----> constant time cost: c1
   while (low <= high)            ----> no more than \log_2 n steps
      mid = floor((low + high)/2) ----+
      if(A[mid] == x)                 |
         return mid                   | Time cost of all these in
      else if(A[mid] < x)             | one ‘while’ loop step
         low = mid + 1                | is constant: c
      else                            |
         high = mid - 1           ----+
   return -1  ------------------------> constant time cost: c2
}
```

- Each `while` loop step cuts the array size to be a half.
- Then, how many `while` loop steps can we have ? No more than than $\log_2 n$, where n is the array size.
- So, the total time cost is: $T(n) = c_1 + c \log_2 n + c_2 = O(\log n)$

The recursion-based and iteration-based binary searches have the same time complexity. In practice, the iteration-based can be faster as it does not have the space and time overhead needed by the system to maintain the recursive calls, especially when the array size is huge. (You can implement both and do some experiments if you want.)