CSE 241 Class 4

Jeremy Buhler

September 2, 2015

On tap for today: more recurrences, more recursion trees, one new algorithm (binary search)

1 Recursion Trees: Two More Examples

Recursion trees are a general way to solve recurrences. I want to make sure you're comfortable with how to construct and use them.

- Start with a recurrence for $T(n)$.
- Sketch structure of recursive calls.
- Account local work performed for each call.
- Sum up work over entire tree.

Here's an example we haven't seen before:

$$T(n) \geq \begin{cases} c_0 & \text{if } n = 1 \\ 2T(\frac{n}{2}) + cn^2 & \text{if } n > 1 \end{cases}$$

Assume n is a power of 2. Let's draw the tree:

Now we sum over all the levels of the tree.

$$\begin{aligned} T(n) &\geq \sum_{k=0}^{\log n-1} \frac{cn^2}{2^k} + c_0 n \\ &= cn^2 \sum_{k=0}^{\log n-1} \frac{1}{2^k} + c_0 n \\ &= cn^2 \left[\sum_{k=0}^{\infty} \frac{1}{2^k} - \sum_{k=0}^{\infty} \frac{1}{2^{k+\log n}} \right] + c_0 n \\ &= cn^2 \left[2 - \frac{1}{2^{\log n}} \sum_{k=0}^{\infty} \frac{1}{2^k} \right] + c_0 n \\ &= cn^2 \left[2 - 2/n \right] + c_0 n \\ &= 2cn^2 + c'n. \end{aligned}$$

Asymptotic growth: conclude that $T(n) =$ **[wait]** $\Omega(n^2)$. Why? Because recurrence itself is just a lower bound on $T(n)$.

Here's another example:

$$T(n) = \begin{cases} c_0 & \text{if } n = 1 \\ 3T(\frac{n}{2}) + cn & \text{if } n > 1 \end{cases}$$

Assume n is a power of 2. Let's draw the tree:

Now we sum over all the levels of the tree.

$$
\begin{aligned}
T(n) &= \sum_{k=0}^{\log_2 n-1} cn\left(\frac{3}{2}\right)^k + c_0 3^{\log_2 n} \\
&= cn\sum_{k=0}^{\log_2 n-1} \left(\frac{3}{2}\right)^k + c_0 n^{\log_2 3} \\
&= cn\left[\frac{(3/2)^{\log_2 n}-1}{3/2-1}\right] + c_0 n^{\log_2 3} \\
&= 2cn\left[n^{\log_2(3/2)}-1\right] + c_0 n^{\log_2 3} \\
&= (c_0+2c)n^{\log_2 3} - 2cn.
\end{aligned}
$$

Asymptotic growth: conclude that $T(n) =$ **[wait]** $\Theta(n^{\log_2 3})$. Why? Because recurrence is exact – both upper and lower bound on $T(n)$.

To review sneaky summation and log tricks, see Section 3.2 and Appendix A of your text!

2 A New Recursion: Binary Search

Binary search is a classic example of a divide-and-conquer algorithm, albeit with nothing to combine.

- **Input**:
  - a sorted array of numbers $A[p \dots r]$
  - a number x
- **Returns**:
  - (an) index of x in A if it's present
  - "notFound" otherwise

```
BSEARCH(x, A, p, r)
  if p = r                                          ▷ base case
     if A[p] = x
        return p
     else
        return notFound

  mid ← ⌈(p + r)/2⌉
  if A[mid] > x
     return BSEARCH(x, A, p, mid − 1)
  else
     return BSEARCH(x, A, mid, r)
```

What is input size of binary search? $n = r - p + 1$!

3 Correctness of Binary Search

Prove by induction on n.

- **Base**: $n = 1$ – by inspection.
- **Inductive**: Consider $A[p \ldots r]$, which is sorted. If $x < A[\text{mid}]$ is present, it must be in the subarray $A[p \ldots \text{mid} - 1]$. Otherwise, it must be in $A[\text{mid} \ldots r]$. For each case, we recur on the correct subarray, which is shorter than $A[p \ldots r]$. By inductive hyp., recursive call returns correct position of x if present, or *notFound* otherwise. QED.

4 Running Time of Binary Search

Let's analyze the algorithm's running time.

- **Base case** takes $\Theta(1)$ time.
- **Inductive case** takes $\Theta(1)$ time, plus *one* recursive call on an array of half the size.

For simplicity, assume again that $n = r - p + 1$ is power of two. Our recurrence is then

$$T(n) = \begin{cases} c_0 & \text{if } n = 1 \\ T(n/2) + c & \text{if } n > 1 \end{cases}$$

Recursion tree is a "stick."

Now sum the cost over the tree...

$$\begin{aligned} T(n) &= \sum_{k=0}^{\log n - 1} c + c_0 \\ &= c \log n + c_0 \\ &= \Theta(\log n) \end{aligned}$$