

Analyzing Insertion Sort as a Recursive Algorithm

Lecture 3, Sept 29, 2009

- Basic idea: **divide and conquer**
 - » Divide into 2 (or more) subproblems.
 - » Solve each subproblem recursively.
 - » Combine the results.
- Insertion sort is just a bad divide & conquer!
 - » Subproblems: (a) last element
(b) all the rest
 - » Combine: find where to put the last element

28

Recursion for Insertion Sort

- We get a recursion for the running time $T(n)$:

$$\begin{aligned} T(n) &= \begin{cases} T(n-1)+n & \text{for } n > 1 \\ 1 & \text{for } n = 1 \end{cases} \\ T(n) &= T(n-1)+n \\ &= T(n-2)+(n-1)+n \\ &= T(n-3)+(n-2)+(n-1)+n \\ &= \dots \\ &= \sum_{i=1}^n i \\ &= \Theta(n^2) \end{aligned}$$

- Formal proof: by induction.
- Another way of looking: split into n subproblems, **merge one by one**.

29

Improving the insertion sort

- Simple insertion sort is good only for small n .
- Balance sorting vs. merging: Merge equal size chunks.
- How to merge:
(details of what happens when i or j reach end of the arrays are omitted)

```
i=1, j=1
for k=1 to 2n
  if A(i)<B(j)
    then
      C(k)=A(i)
      i++
    else
      C(k)=B(j)
      j++
end
```

- $O(n)$ time to merge

30

Analysis

- Iterative approach:
 - » Merge size-1 chunks into size-2 chunks
 - » Merge size-2 chunks into size-4 chunks
 - » etc.

$$T(n) = \frac{n}{2} \text{merge}(1) + \frac{n}{4} \text{merge}(2) + \frac{n}{8} \text{merge}(4) + \dots$$

Overall: $\Theta(n \log n)$

- Intuitively right, but **needs proof!**

31

Analyzing Recursive Merge-Sort

- Another approach: recursive.
 - » Divide into 2 equal size parts.
 - » Sort each part recursively.
 - » Merge.
- Recursion is a way of thinking.
- Easy to design recursive algorithms.
- We directly get the following recurrence:
$$T(n) = \begin{cases} 2T(n/2) + \Theta(n) & n > 1 \\ 1 & n = 1 \end{cases}$$
- How to formally solve recurrence ?
 - » For example, does it matter that we have $\Theta(n)$ instead of an exact expression ??
 - » Does it matter that we sometimes have n not divisible by 2 ??

32

Tools

- Standard summations:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

$$\text{Harmonic function: } H(n) = \sum_{i=1}^n \frac{1}{i} = \ln n + O(1)$$

$$\begin{aligned} \text{Telescoping series: } \sum_{k=1}^{n-1} \frac{1}{k(k+1)} &= \sum_{k=1}^{n-1} \left(\frac{1}{k} - \frac{1}{k+1} \right) \\ &= \sum_{k=1}^{n-1} \left(\frac{1}{k} \right) - \sum_{k=1}^{n-1} \left(\frac{1}{k+1} \right) \\ &= \sum_{k=1}^{n-1} \left(\frac{1}{k} \right) - \sum_{k=2}^n \left(\frac{1}{k} \right) \\ &= 1 - \frac{1}{n} \end{aligned}$$

33

More tools

- Another useful trick:

$$\sum_{k=0}^{\infty} kx^k = x \frac{d}{dx} \sum_{k=0}^{\infty} x^k = x \frac{d}{dx} \frac{1}{1-x} = \frac{x}{(1-x)^2}$$

- Summary:
 - » Learn to recognize standard simplifications
 - » Try going opposite direction
 - » If all fails - apply tricks one by one...

34

Recurrences

- Algorithm “calls itself” - *recursive*.

$$T(n) = \begin{cases} 1 & n=1 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1 & \text{otherwise} \end{cases}$$

- First, solve for $n = 2^k$

» Claim: $T(n) = \lg n + 1$

» Proof by induction (on k):

$$\begin{aligned} T(1) &= 1 \\ T(2^{k+1}) &= T(2^k) + 1 \\ &= \lg(2^k) + 1 + 1 \\ &= k + 2 \\ &= \lg(2^{k+1}) + 1 \quad \text{QED} \end{aligned}$$

35

What if n not a power of 2 ?

- Easy to prove by induction that $T(n) \geq T(n-1)$
- Now we can say: $T(n) \leq T(2^{\lceil \lg n \rceil}) = \lceil \lg n \rceil + 1 = \Theta(\log n)$
- Observe that we **did not prove Theta**, only **big-Oh** !
- Technically, we should be careful about floor/ceiling, but usually we can safely concentrate on **$n = \text{power of 2}$** .

36

Guessing the solution

- Instead of adding n numbers sequentially, lets divide into 2 parts, compute sum of each part recursively, and add the results:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

Note that we omit the $n=1$ case for simplicity

Guess: $T(n) \leq cn$ for some constant c

$$\text{Then: } T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

$$\leq c \lfloor n/2 \rfloor + c \lceil n/2 \rceil + 1$$

$$= cn + 1$$

Are we done ?

- Need a **stronger induction hypothesis** !
Assume: $T(n) \leq cn - b$ for some constants c, b
Then: $T(n) = \dots = cn - 2b + 1 \leq cn - b$ for $b \geq 1$
- Value of c can be computed from initial conditions.
 If $T(1) = 1$, and $b = 1$, then choosing $c = 2$ is one possible choice.

Other choices for b and c??

37

Another example

- Consider recursion: $T(n) = 4T\left(\frac{n}{2}\right) + n$
- First guess: $T(n) \leq cn^3$
- We omit base case.
 Induction step: $4T\left(\frac{n}{2}\right) + n \leq c\frac{n^3}{2} + n = cn^3 + \underbrace{\left(n - \frac{c}{2}n^3\right)}_{\text{rest}}$
 for $c \geq 2, n \geq 1 \Rightarrow$ "rest" ≤ 0 QED

- But we can do better! First try: $T(n) \leq cn^2$.
 Does not work, too weak !

Second try: Assume: $T(n) \leq c_1n^2 - c_2n$

$$\begin{aligned} \text{Then: } T(n) &= 4T\left(\frac{n}{2}\right) + n \leq 4\left(c_1\left(\frac{n}{2}\right)^2 - c_2\frac{n}{2}\right) + n = c_1n^2 - 2c_2n + n \\ &= c_1n^2 - c_2n + \underbrace{(n - c_2n)}_{\text{REST}} \end{aligned}$$