Analyzing Insertion Sort as a Recursive Algorithm

- 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

Lecture 2, April 5, 2001

Recursion for Insertion Sort

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

\[
T(n) = \begin{cases} 
T(n-1) + n & \text{for } n > 1 \\
1 & \text{for } n = 1 
\end{cases}
\]

\[
T(n) = T(n-1) + n \\
= T(n-2) + (n-1) + n \\
= T(n-3) + (n-2) + (n-1) + n \\
= \ldots \\
= \sum_{i=1}^{n-1} i \\
= \Theta(n^2)
\]

- Formal proof: by induction.

- Another way of looking: split into $n$ subproblems, merge one by one.
Improving the insertion sort

- Simple insertion sort is good only for small $n$.

- Balance sorting vs. merging: Merge equal size chunks.

- How to merge:

  ```
  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 !!

Analysis

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

  $$\frac{n}{2}merge(1)+\frac{n}{4}merge(2)+\frac{n}{8}merge(4)+\cdots$$

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

- Intuitively right, but needs proof!
Analyzing Recursive Merge-Sort

- Another approach: recursive.
  - Divide into 2 equal size parts.
  - Sort each part recursively.
  - Merge.

- 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??

Summations

- Before dealing with recurrencies, need to read Chapter 3, in particular summations:
  \[ e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \]
  Harmonic function: \( H(n) = \sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1) \)

  Telescoping series:
  \[
  \sum_{k=1}^{n} \frac{1}{k(k+1)} = \sum_{k=1}^{n} \left( \frac{1}{k} - \frac{1}{k+1} \right) \\
  = \left( \frac{1}{1} - \frac{1}{2} \right) + \left( \frac{1}{2} - \frac{1}{3} \right) + \ldots + \left( \frac{1}{n} - \frac{1}{n+1} \right) \\
  = 1 - \frac{1}{n+1} \\
  \]

\[= 1 - \frac{1}{n} \]
More summations

- Another useful trick:

\[ \sum_{n=2}^{\infty} \frac{x^n}{n!} = x \sum_{n=1}^{\infty} \frac{x^n}{n} = \frac{d}{dx} \left( \frac{1}{1-x} \right) \]

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

Recurrences

- Chapter 4 in the textbook.

- Algorithm “calls itself” - recursive.

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

- First, solve for \( n = 2^k \)
  » Claim: \( T(n) = \lg n + 1 \)
  » Proof by induction: \( T(1) = 1 \)
  \[
  T(2^{k+1}) = T(2^k) + 1 \\
  = \lg(2^k) + 1 + 1 \\
  = k + 2 \\
  = \lg(2^{k+1}) + 1 \quad \text{QED}
  \]
What if n not a power of 2 ?

- Easy to prove by induction that \( T(n) \geq T(n-1) \)
- Now we can say: \( T(n) \leq T\left(2^\left\lfloor \log n \right\rfloor\right) = \left\lfloor \log n \right\rfloor + 1 = \Theta(\log n) \)
- 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 \( n=\text{power of 2} \).

Guessing the solution

- Instead of adding sequentially, lets divide into 2 parts, add each one recursively, and add the result:
  
  \[
  T(n) = T\left(\left\lfloor n/2 \right\rfloor\right) + T\left(\left\lceil n/2 \right\rceil\right) + 1
  \]

  Note that we omit the \( n=1 \) case for simplicity

  Guess: \( T(n) < cn \) for some constant \( c \)
  Then: \( T(n)=T\left(\left\lfloor n/2 \right\rfloor\right) + T\left(\left\lceil n/2 \right\rceil\right) + 1 \)

  \[
  < c\left\lfloor n/2 \right\rfloor + c\left\lceil n/2 \right\rceil + 1
  = cn + 1
  
  Oopssss....

  Need a stronger induction hypothesis !
  Assume: \( T(n) < cn - b \) for some constants \( c,b \)
  Then: \( T(n) = \cdots = cn - 2b + 1 < cn - b \) for \( b > 1 \)
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:** \( 4T\left(\frac{n}{2}\right) + n \leq c\frac{n^3}{2} + n = cn^3 + (n - \frac{c}{4}n^2) \)
  
  for \( c \geq 2, n \geq 1 \) ⇒ "rest" \( \leq 0 \) QED
- **But we can do better**: First try: \( T(n) \leq cn^2 \) is too weak!
  
  Assume: \( T(n) \leq cn^2 - c_2n \)
  Then: \( T(n) = 4T\left(\frac{n}{2}\right) + n \leq 4\left(c\left(\frac{n}{2}\right)^2 - c_2\frac{n}{2}\right) + n = cn^2 - 2c_2n + n \)

\[
= cn^2 - c_2n + \left(n - c_2n\right) \leq cn^2 - c_2n + \left(n - c_2n\right)
\]

Initial Conditions

- **Can initial conditions affect the solution?** → YES!

\[
T(n) = \left\lfloor T(n/2^2) \right\rfloor
\]

\[
T(1) = 2 \quad \Rightarrow \quad T(n) = 2^n
\]

\[
T(1) = 3 \quad \Rightarrow \quad T(n) = 3^n
\]

\[
T(1) = 1 \quad \Rightarrow \quad T(n) = 1
\]

- \( n \) was assumed to be a power of 2.
Iterating recurrences

- **Example:** \( T(n) = 4T(n/2) + n \)
  
  \[
  = n + 4(n/2 + 4T(n/4)) = n + 2n + 16T(n/4)
  
  = n + 2n + 16\left\{n/4 + 4T(n/8)\right\} = n + 2n + 4n + 4T(n/8)
  
  = n + 2n + 4n + 8n + \cdots = n \sum_{i=0}^{k-1} 2^i + \Theta(n) T(1)
  
  \Theta(n^2)
  
  \Theta(n^2)
  
- **Disadvantages:**
  - Tedious
  - Error-prone

- **Use to generate initial guess,** and then prove by induction!

Recursion Tree

- **Example:** \( T(n) = T(n/4) + T(n/2) + n^2 \)

\[
\begin{array}{c|c|c}
T\left(\frac{n}{2}\right) & T\left(\frac{n}{4}\right) & T\left(\frac{n}{16}\right) \\
\frac{n^2}{2} & \frac{n^2}{4} & \frac{5}{16}n^2 \\
T\left(\frac{n}{2}\right) & T\left(\frac{n}{4}\right) & T\left(\frac{n}{16}\right) \\
\frac{n^2}{2} & \frac{25}{25}n^2 & \frac{5}{16}n^2 \\
\end{array}
\]

- **At k-th level we get a general formula:** \( i \) steps right, \( k-i \) left

\[
= n^2 \sum_{i}^{k} \left[2 \cdot 4^{(k-i)} \right]^2 = n^2 \sum_{i}^{k} \left[4 \cdot 16^{(k-i)} \right] = n^2 \left[ \frac{1}{4} + \frac{1}{16} \right] = n^2 \left[ \frac{5}{16} \right]
\]

- **Summing over all k, geometric sum, sums to \( \Theta(n^2) \)**
  (overcount, since \( T(1) = 1 \)
Master Method

- Consider the following recurrence $T(n) = aT(n/b) + f(n)$; $a \geq 1, b > 1$

1. $f(n) = \Theta(n^{\log_a b - \epsilon})$, $\epsilon > 0 \Rightarrow \Theta(n^{\log_a b})$

2. $f(n) = \Theta(n^{\log_a b} \log n)$, $k \geq 0 \Rightarrow \Theta(n^{\log_a b} \log^{k+1} n)$

3. $f(n) = \Omega(n^{\log_a b + \epsilon})$, $\epsilon > 0$
   - $af(n/b) \leq cf(n)$ for some $c < 1 \Rightarrow \Theta(f(n))$

- More general than the book.

- Let $Q = n^{\log_a b}$. Then the cases are:
  
  - $Q$ polynomially larger than $f$.
  - $f$ is larger than $Q$ by a polylog factor.
  - $Q$ polynomially smaller than $f$.

Build recursion tree

\[
\begin{array}{c}
\text{Last row: } \Theta(n^{\log_a b}) = \Theta(n^{\log_a b}) \text{ elements, each one } \Theta(1). \\
\text{Total: } \Theta(n^{\log_a b}) + \sum_{i=1}^{\log n} a^i f(n/b^i) \\
\end{array}
\]

Which term dominates?
First case: “f(n) small”

\[ \frac{n^{\Theta_{a,b}^\alpha}}{f(n)} = \Omega(n^\epsilon) \Rightarrow \exists c \text{ s.t. for “large enough n”, } f(n) \leq cn^{\Theta_{a,b}^\alpha}/n^\epsilon \]

\[ a^i f((n/b^i)) \leq ca^i(n/b^i)^{\Theta_{a,b}^\alpha} = cn^{\Theta_{a,b}^\alpha} \cdot a^i \cdot \frac{b^i}{b^{i+1}} = cn^{\Theta_{a,b}^\alpha} \cdot b^i \]

The ratio summed over all possible \( j \):

\[ \frac{b^j n^{\Theta_{a,b}^\alpha} - 1}{b^j - 1} = \Theta(n^\epsilon). \]

Total: \( O(n^{\Theta_{a,b}^\alpha}) \).

Lower bound is trivial (Why ?? First term in the original expression was already \( \Theta(n^{\Theta_{a,b}^\alpha}) \).)

Second case

\[ f(n) = \Theta(n^{\Theta_{a,b}^\alpha} \log^k n) \]

\[ \sum_{m=n^{\Theta_{a,b}^\alpha}} a^i \cdot \left( \frac{n}{b^j} \right) \log^k \left( \frac{n}{b^j} \right) = O(\log^{k+1} n) n^{\Theta_{a,b}^\alpha} \quad \text{(there are } O(\log n) \text{ elements in the sum)} \]

This is an UPPER bound ! How to prove the lower bound ??

Rough and easy approach:

\[ \sum_{i=1}^{\Theta_{a,b}^\alpha - 1} \log^i \left( \frac{n}{b^j} \right) \geq \sum_{i=1}^{\Theta_{a,b}^\alpha / 2} \log^i \left( \frac{n}{b^j} \right) \geq \sum_{i=1}^{\Theta_{a,b}^\alpha / 2} \left( \log n \right)^2 = (\text{const}) \log^{k+1} n \]

(Note that we use the assumption that \( k \geq 0 \))
Third case

\[ a^j f(n/b^j) \leq c^j f(n) \quad \text{for some } c < 1, \text{ and } f(n) = \Omega(n^{b_0^{a_0+c}}) \]

\[ \Rightarrow \sum_{j=0}^{\infty} c^j f(n) = \Theta(f(n)) \]

\[ \Rightarrow \sum_{j=0}^{\infty} a^j f(n/b^j) = O(f(n)) \quad \text{Note Big-Oh and not Theta!} \]

The first term is already \( \Theta(n^{b_0^{a_0+c}}) = O(f(n)) \)

TOTAL: \( \Theta(f(n)) \)