CISC 320 Introduction to Algorithms
Fall 2005

Lecture 3
Recurrences and Master theorem

General scheme for time complexity analysis

1. For a sequence of blocks, add up the cost of individual blocks
   - For a loop, the worst case = the loop range times the cost of a single iteration
2. With alternation, take the cost of the most costly branch
3. If recursive procedure called, add T(n'), where n' is the size at call.

Recursion for computation
A computation model is Turing complete when it can compute everything that can be computed by a Turing machine.

Pragmatically, a model (or a language) is Turing complete if it can do
- sequence
- branch
- repetition (either as loop or as recursion)

Recursion
- is as powerful as iteration in establishing a Turing complete model.
- is proof-friendly for proving correctness of algorithms. (Thus promoted in functional programming languages, such as ML.)
- Why? (Free of "Computing by Side Effect" problems using iterations)
- Myth: Loop is much faster than recursion
- Truth: recursion can be as efficient as iteration.

Iterations can be converted as recursions
For example, Sequential Search can be implemented recursively

int seqSearchRec(int E[], int m, int num, int K)
int ans;
1 if (m >= num)
2 ans = -1;
3 else if (E[m] == K)
4 ans = m;
5 else
6 ans = seqSearchRec(E, m+1, num, K);
7 return ans;

For example, the recursive Sequential Search can be analyzed using this scheme
int seqSearchRec(int E[], int m, int num, int K)
int ans;
1 if (m >= num)
2 ans = -1;
3 else if (E[m] == K)
4 ans = m;
5 else
6 ans = seqSearchRec(E, m+1, num, K);
7 return ans;

Let n = num – m as the initial size
T(n) = 1 + max(0, 1+ max(0, T(num-(m+1)))) = T(n-1) + 2

Divide and Conquer
E.g., Binary search of an ordered array.
Modify the seqSearchRec to do binary search. If the recursive implementation of sequential search is superficial, a recursive implementation of binary search is a real convenience (as compared to a loop based implementation).

T(n) = T(n/2) + Θ(1).

In general, the cost of solving a problem of size n is shared by the cost of a subproblems of size n/b, plus non-recursive overhead cost f(n):

T(n) = a T(n/b) + f(n)

This is a recurrence equation.
How to evaluate the cost T(n)?
Recursion-tree method

Example: 
\[ T(n) = T(n/2) + T(n/2) + n \]

\[ \begin{align*}
T(n) & \quad n \\
T(n/2) & \quad n/2 \\
T(n/4) & \quad n/4 \\
\vdots & \quad \ldots
\end{align*} \]

row-sum or per-level cost

N_(lg(n))

\[ T(n) = \Theta(n \log(n)) \]

Observations of recursion-tree method

1. \( T(n) \) = the sum of the nonrecursive costs of all nodes in the tree, which is the sum of the per-level costs at all levels;
2. Depth of the tree is \( D = \log_b n \);
3. Number of leaves is approximately \( L = a^D = n^E \) where \( E = \log_b a \);
4. If the per-level costs remain about constant at all depth, then \( T(n) \in \Theta(f(n) \log(n)) \);
5. If the per-level costs grow fast, the cost at the leaves would dominate, therefore \( T(n) \in \Theta(n^E) \);
6. If the per-level costs decrease fast, the cost at the root would dominate, therefore \( T(n) \in \Theta(f(n)) \);
7. And more formally,

The Master theorem (Theorem 4.1)
The recurrence equation
\[ T(n) = a \cdot T(n/b) + f(n), \]
where \( a \geq 1 \), \( b > 1 \), and \( n/b \) interpreted as either \( \lfloor n/b \rfloor \) or \( \lceil n/b \rceil \). Then \( T(n) \) can be bounded asymptotically as follows:

1. If \( f(n) = O(n^{E-\varepsilon}) \) for constant \( \varepsilon > 0 \), then \( T(n) = \Theta(n^E) \) where \( E = \log_b a \), called critical exponent. (Note: this means \( n^E \) is polynomially faster than \( f(n) \).)
2. If \( f(n) = \Theta(n^E) \), then \( T(n) = \Theta(f(n) \log(n)) \).
3. If \( f(n) = \Omega(n^{E+\varepsilon}) \) for \( \varepsilon > 0 \), and if \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \). (Note: this means \( f(n) \) is polynomially faster than \( n^E \).)

Example 1
\[ T(n) = 7T(n/2) + n^2 \]
1. Recognize \( a, b, \) and \( f(n) \):
   \( a = 7, b = 2, \) and \( f(n) = n^2 \)
2. Compute \( E = \log_b a = \log(7) \)
3. Compare \( f(n) \) and \( n^E \) asymptotically
   \[ f(n)/n^E = n^2/\log(7) = 2 - \varepsilon \quad \text{for some } \varepsilon > 0 \]
4. Apply appropriate case of Master Theorem
   case 1 applies: \( T(n) = \Theta(n^2) \)

Example 2
\[ T(n) = 4T(n/2) + n^2 \log(n) \]
1. Recognize \( a, b, \) and \( f(n) \):
   \( a = 4, b = 2, \) and \( f(n) = n^2 \log(n) \)
2. Compute \( E = \log_2 a = \log(4) = 2 \)
3. Compare \( f(n) \) and \( n^E \) asymptotically
   \[ f(n)/n^E = n^2 \log(n)/n^2 = \log(n) \]
4. Determine appropriate case of Master Theorem and apply
   case 1: \( f(n)/n^E = \log(n) \leq O(n^\varepsilon) \) for some \( \varepsilon > 0 \) NO
   case 2: \( f(n)/n^E = \log(n) \leq \Theta(1) \) NO
   case 3: \( f(n)/n^E = \log(n) \leq \Omega(n^\varepsilon) \) for some \( \varepsilon > 0 \) NO
   Note: \( \log(n) \) is faster than \( O(1) \) but slower than \( x^\varepsilon \) for any \( \varepsilon > 0 \) (Exercise).
Lesson: There are gaps between cases in Master Theorem, therefore Master Theorem does not cover all recurrence equations of that form.

Example 3
\[ T(n) = T(n/4) + T(n/2) + n^2 \]
1. Recognize \( a, b, \) and \( f(n) \):
   \( a = 2, b = 4, \) and \( f(n) = n^2 \)
2. Compute \( E = \log_4 a = \log(2) = 1 \)
3. Compare \( f(n) \) and \( n^E \) asymptotically
   \[ f(n)/n^E = n^2/\log(2) = 25/16 \quad \Theta(n^2) \]
4. Determine appropriate case of Master Theorem and apply
   row-sum
   \[ n^2 \]
   \[ (5/16)n^2 \]
   \[ (25/256)n^2 \]
   \[ \ldots \]
   \[ \Theta(n^E) \]

Exercise: \( T(n) = n^2(1 + 5/16 + (5/16)^2 + \ldots) \leq (16/11)n^2 = \Theta(n^2) \)
Substitution method

- Make a guess
- Substitute it into the recurrence
- Prove the recurrence hold by mathematical induction

Example: $T(n) = T(n/4) + T(n/2) + n^2$.

From decision tree method, we have a good guess that $T(n) = \Theta(n^2)$.

Let $T(n) \leq cn^2$, where $c$ is a suitable positive constant.

Plug it into the RHS of the recurrence.

\[ T(n) \leq c (n/4)^2 + c(n/2)^2 + n^2 = (c/16 + c/4 + 1) n^2 \leq cn^2, \text{ when } c \geq 16/5 \]

Induction Proofs

A mechanic procedure with mainly 3 steps

Step 1: prove base case(s), e.g., $n=0$.

Step 2: assume the goal is true for arbitrary $n$, say $n=k$.

Step 3: then prove it is also true for $n=k+1$.

Example: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Base case $n = 1$

LHS = 1 and RHS = $1(1+1)/2 = 1$

Note: we can do this manually for $n=2, 3, ...$

Let's assume it holds for arbitrary $n \geq 1$, we now prove it also holds for $n+1$.

LHS($n+1$) = $\sum_{i=n+1}^{n+1} i = (\sum_{i=1}^{n} i) + (n+1)$

= $\frac{n(n+1)}{2} + (n+1)$

= $\frac{n^2 + n + 2n + 2}{2}$

= $\frac{n(n+1)(n+2)}{2}$

= RHS($n+1$)

Since we have proved manually it is true when $n=1$. Now we know if it is true for $n=1$ it must be true for $n=2$, and if it is true for $n=2$ it must be true for $n=3$, and on and on.

Note: such a procedure is like to unravel a recursive call in a reversed order, i.e., from base case to more general cases.