Solving Recurrences

Prepared by Dr. Lee

Analysis of Recurrences

- How we can evaluate the running time/efficiency of the recursive algorithms?

Recurrence

- When an algorithm contains a recursive call to itself or if it is represented using a Divide-and-Conquer approach, its running time can often be described by a **recurrence equation** or **recurrence**
- It describes the overall running time on a problem of size *n* in terms of running time on smaller inputs

Solving Recurrences

- Solving recurrences means the asymptotic evaluation of their efficiency
- The recurrence can be solved using some mathematical tools and then bounds (big-O, big-Ω, and big-Θ) on the performance of the algorithm should be found according to the corresponding criteria

Composing Recurrences

- A recurrence for the running time of a divide-and-conquer algorithm is based on the three steps:
    1) Let ***T*(*n*)** be the running time of a problem of size *n*. If the problem size is small enough ($n \le c$) for some constant *c*, the straightforward solution takes constant time, i.e. **$\Theta(1)$**
    2) Suppose that our division of the problem yields *k* subproblems, each of which is $1/m$ size of the original.
    3) If we take ***D*(*n*)** time to divide the problem into subproblems and ***C*(*n*)** time to combine the solutions to the subproblems to the original problem, we got the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if} \quad n \le c \\ kT(n/m) + D(n) + C(n) & \text{otherwise} \end{cases}$$

Solving Recurrences

- Hence, solving recurrences means finding the asymtotic bounds (big-O, big-Ω, and big-Θ) for the function $T(n)$

Solving Recurrences

- **Substitution method** – we guess a bound and then use mathematical induction to prove our guess
- **Recursion-tree method** converts recursion into a tree whose nodes represent the "subproblems" and their costs. It is used to estimate a good guess
- **Master Theorem method** provides bounds for recurrences of the form

$$T(n) = aT(n/b) + f(n); \quad a \geq 1, \quad b > 1$$

$f(n)$ is a given function

Solving Recurrences

- **Master Theorem method**
  - Provides the immediate solution for recurrences of the form

$$T(n) = aT(n/b) + f(n); \quad a \geq 1, \quad b > 1$$

  - $f(n)$ is a given function, which satisfies some pre-determined conditions

Solving Recurrences

- **Recursion-tree method**
  - Converts recursion into a tree whose nodes represent the "subproblems" and their costs
  - Then the sum of these costs can be used as a "good guess" for the substitution method or the master theorem method

Solving Recurrences

- **Substitution method**
  - Known as a “good guess method”
  - The first step is: to guess a solution (a bound)
  - The second step is: to prove the correctness of the guess substituting the guess into the recurrence and using induction.

Substitution Method: Example

$$T(n) = \begin{cases} 1 & \text{if} \quad n = 1 \\ 2T\left(\frac{n}{2}\right) + n & \text{if} \quad n > 1 \end{cases}$$

- Guess for the exact solution: $g(n) = n\lg n + n$

Substitution Method (the exact solution)

- Induction: Guess: $T(n) = n \lg n + n$
- **Basis**: $n = 1 \Rightarrow T(n) = 1$; $T(n) = n \lg n + n = 1 \cdot \lg 1 + 1 = 1$
  $\rightarrow n_0 = 1$
- **Inductive step**: Inductive Hypothesis is
  $$T(k) = k \lg k + k, \qquad \forall k \geq n_0$$
- Let us use this hypothesis:

$$T(n) = 2T\left(\frac{n}{2}\right) + n = 2\left(\underbrace{\frac{n}{2}\lg\frac{n}{2} + \frac{n}{2}}_{substitution\ \ T(n/2)}\right) + n = n\lg\frac{n}{2} + n + n =$$

$$= n(\lg n - \lg 2) + n + n = n\lg n - n + n + n = n\lg n + n \qquad \square$$

Substitution Method

- Generally, we use asymptotic notation
  - We would write $T(n) = 2T(n/2) + \Theta(n)$
  - We assume $T(n) = O(1)$ for sufficiently small n
  - We express the solution by asymptotic notation:

    $$T(n) = \Theta(n \lg n)$$
- For the substitution method
  - Name the constant in the additive term
  - Show the upper(O) and lower (Ω) bounds separately. Might need to use different constants for each.

Substitution Method (with asymptotic notation)

- $T(n) = 2T(n/2) + \Theta(n)$
- If we want to show an upper bound of $T(n) = 2T(n/2) + O(n)$, we write $T(n) \leq 2T(n/2) + cn$ for some positive constant c

Substitution Method (with asymptotic notation)

$$T(n)=\begin{cases} 1 & \text{if} \quad n=1 \\ 2T\left(\frac{n}{2}\right)+n & \text{if} \quad n>1 \end{cases}$$

- Upper bound:
  - Guess: $T(n) \le dn \lg n$ for some positive constant d.
  - Substitution:

$$T(n) \le 2T(n/2)+cn = 2\left(d\frac{n}{2}\lg\frac{n}{2}\right)+cn = dn\lg\frac{n}{2}+cn =$$

$$= dn\lg n - dn + cn \le dn\lg n$$

if $-dn + cn \le 0$, $d \ge c$

Therefore, $T(n) = O(n \lg n)$

What about n_0?

$T(1) = 1 \le d1 \lg 1 = 0$ (no)

$T(2) = 4 \le d2 \lg 2 = 2d$ (yes)

$\Rightarrow d \ge 2,\ n_0 = 2$

Substitution Method (with asymptotic notation)

$$T(n) = \begin{cases} 1 & \text{if} \quad n = 1 \\ 2T\left(\frac{n}{2}\right) + n & \text{if} \quad n > 1 \end{cases}$$

- Lower bound: write $T(n) \geq 2T(n/2) + cn$ for some positive constant c
  - Guess: $T(n) \geq dn \lg n$ for some positive constant d.
  - Substitution:

$$T(n) \geq 2T(n/2) + cn = 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn = dn\lg\frac{n}{2} + cn =$$

$$= dn\lg n - dn + cn \geq dn\lg n$$

if $-dn + cn \geq 0$, $d \leq c$

Therefore, $T(n) = \Omega(n \lg n)$

What about n_0?

$T(1) = 1 \geq d1 \lg 1 = 0$ (yes)

$T(2) = 4 \geq d2 \lg 2 = 2d$ (yes)

$\Rightarrow d \leq 2, n_0 = 2$

- Therefore, $T(n) = \Theta(n \lg n)$ □

Solving Recurrences

- The substitution method
  - Examples:
    - $T(n) = 2T(n/2) + O(n)$ $\rightarrow$ $T(n) = O(n\lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor) + n$ $\rightarrow$???

Solving Recurrences

- The substitution method
  - Examples:
    - $T(n) = 2T(n/2) + O(n) \rightarrow T(n) = O(n\lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor) + n \rightarrow T(n) = O(n\lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n \rightarrow$???

Solving Recurrences

- The substitution method
  - Examples:
    - $T(n) = 2T(n/2) + O(n) \quad \rightarrow \quad T(n) = O(n\lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor) + n \quad \rightarrow \quad T(n) = O(n\lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n \rightarrow \quad T(n) = O(n\lg n)$

Recursion Tree

- A recursion tree is used to present a problem as a composition of subproblems. It is very suitable to present any divide-and-conquer algorithm
- Each node represents the cost of a single subproblem
- Usually each level of the tree corresponds to one step of the recursion

Recursion Tree

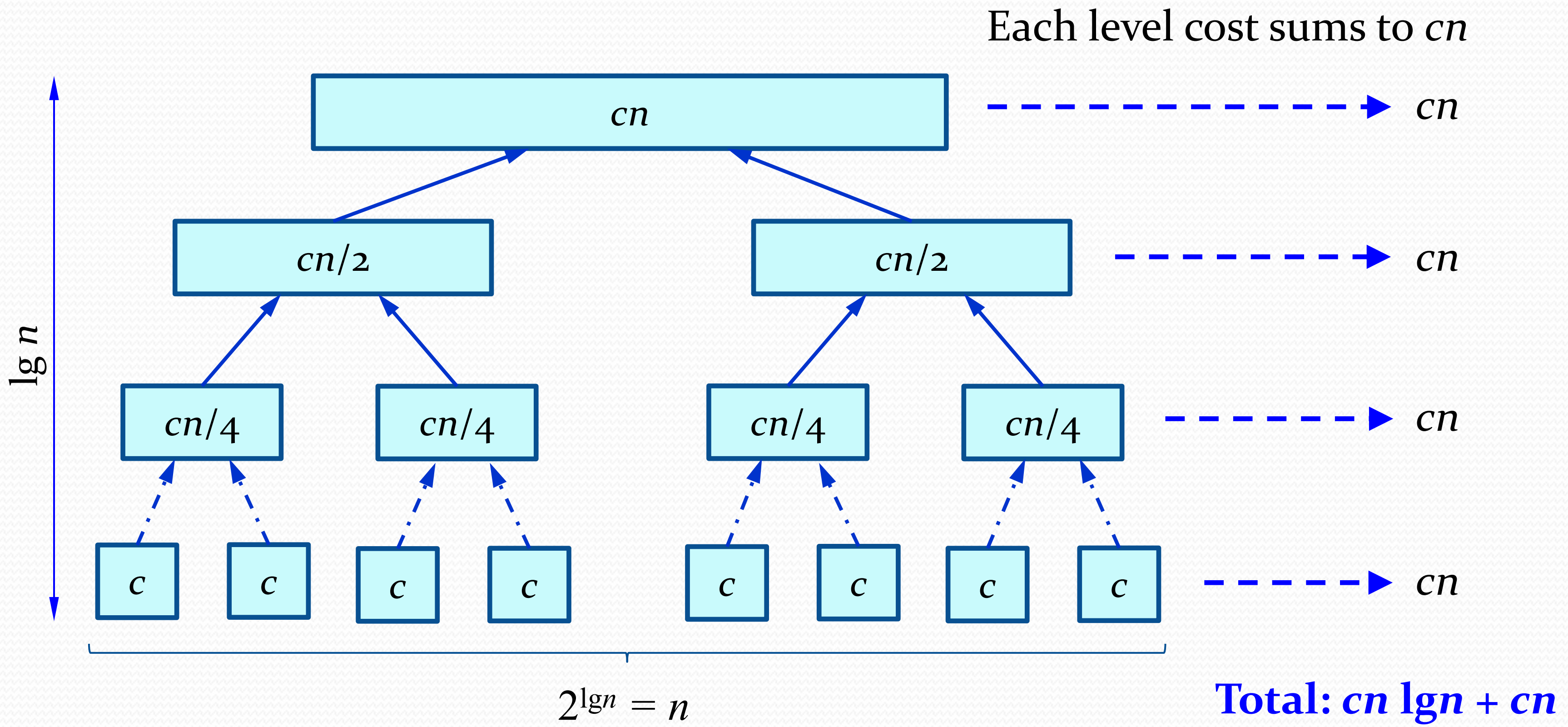
- We sum the costs within each level of the tree to obtain a set of per-level costs
- Then we sum all the per-level costs to determine the total cost of all levels of the recursion
- As a result, we generate a guess that can be then proven by the substitution method

Recursion Tree: Determination of a "Good" Asymptotic Bound

- Draw the tree based on the recurrence
- From the tree determine:
  - # of levels in the tree
  - cost per level
  - # of nodes in the last level
  - cost of the last level (which is based on the number of nodes in the last level)
- Write down the summation using $\sum$ notation – this summation sums up the cost of all the levels in the recursion tree
- Simplify the summation expression coming up with your "guess" in terms of Big-O, or Big-Ω depending on which type of asymptotic bound is being sought).
- Then use Substitution Method to prove that the "guess" is correct.

Recursion Tree: Example – Merge Sort

- Total number of elements per level is always n



Recursion Tree: Example – Merge Sort

- Close form solution as "guess"

$$T(n) = cn\ \lg n + cn = cn\ \lg n + O(n) = O(cn\ \lg n) + O(n) = O(n\ \lg n)$$

- Substitution method
  - Assume n is a power of 2 to avoid floor and cell complica.

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

  - Inductive Hypothesis (IH):
    - Assume: $T(k/2) \le d\ k/2\ \lg\ k/2$
    - Show: $T(k) = 2\ T(k/2) + ck \le d\ k\ \lg\ k$

Recursion Tree: Example – Merge Sort

- $T(k)$ $= 2T(k/2) + ck$ Recurrence
  $\leq 2(dk/2 \lg k/2) + ck$ Substitute IH
  $= dk \lg k/2 + ck$
  $= dk \lg k - dk + ck \leq dk \lg k$

- Find d that satisfies the last line

$$
\begin{aligned}
dk \lg k - dk + ck &\leq dk \lg k \\
-dk + ck &\leq 0 \\
ck &\leq dk \\
c &\leq d
\end{aligned}
$$

Satisfied by $d \geq c$

Recursion Tree: Example – Merge Sort

- Basis:
  $T(1) = 2T(1/2) + c \cdot 1 = c \leq d \cdot 1 \lg 1 = 0$
  since need $n \geq n_0$ for n a power of 2, choose $n_0 = 2$

- Use as basis:
  $T(2) = d2 \lg 2 = 2d$

- By the recurrence, where c is the constant divide and combine time:
  $T(2) = 2T(2/2) + 2c$
  $= T(1) + T(1) + 2c$
  $= c + c + 2c = 4c$

Recursion Tree: Example – Merge Sort

$$\text{Need } T(2) = 4c \le d2 \lg 2 = 2d$$
$$4c \le 2d$$

$$\text{so let} \qquad d = 2c$$

$$\text{Satisfied } d = 2c \ge c$$

- O($n \lg n$): $0 \le T(n) \le dn \lg n$ for $d > 0$, for $^{\forall} n \ge n_0$ satisfied by $d \ge 2c > 0$, for $^{\forall} n \ge n_0 = 2$

APPENDIX

Substitution Method (with asymptotic notation)

- Induction: Guess: $T(n) = O(n \lg n)$
- Basis: $n = 1 \rightarrow T(1) = 1 > c \cdot g(1) = c \cdot 1 \cdot \lg 1 = 0$
  $n = 2 \rightarrow T(2) = 2 \cdot T(1) + 2 = 4 \leq c \cdot g(2) = c(2 \cdot \lg 2) = 2c \rightarrow 2 \leq c$
- Inductive Hypothesis:

  $T(n) = O(n \lg n), \quad \forall n \geq n_0 \qquad \exists c > 0,\ n_0 = 2\text{:}\ T(n) \leq cn\lg n$

- Inductive step

$$T(n) = 2T\left(\frac{n}{2}\right) + n \leq 2\left(c\frac{n}{2}\lg\frac{n}{2}\right) + n = cn\lg\frac{n}{2} + n = cn\left(\lg n - \lg 2\right) + n =$$

$$= cn\lg n - cn\lg 2 + n = cn\lg n - cn + n = cn\lg n - n(c-1) \leq cn\lg n$$

$$n(c-1) \geq 0; n > 0, c > 0 \Rightarrow c - 1 \geq 0 \Rightarrow c \geq 1$$

Substitution Method (with asymptotic notation)

- Analysis: Guess: $T(n) = O(n \lg n)$
- We have to find such $c \geq 1$ and n_0 that

$$\forall n \geq n_0 : T(n) \leq c n \lg n$$

$$n_0 = 1;\ T(1) = 1;\ g(n) = 1 \cdot \lg 1 = 0;$$

$$cg(n) = c \cdot 1 \cdot \lg 1 = c \cdot 0 = 0;\ T(1) = 1 > 0 \quad \rightarrow \quad n_0 > 1$$

$$n_0 = 2;\ T(2) = 2 \cdot T(1) + 2 = 2 \cdot 1 + 2 = 4;\ g(2) = 2 \cdot \lg 2;$$

$$c \cdot 2 \cdot \lg 2 = 2c; \qquad 4 \leq 2c \quad \forall c \geq 2 \qquad \rightarrow \quad n_0 = 2;\ c \geq 2$$