Solving Recurrences

- Method 1: **Recursion Trees**
- Method 2: **Master theorem**
  - Prove a general theorem covering a wide variety of recurrences.
  - Apply the theorem in “cookbook” fashion.
- Method 3: **Guessing and Checking method**
  - Guess an answer.
  - Prove it using induction.

Recursion Trees: Merge Sort (page 1)

- Merge sort: an analysis:

```
// Sort sequence A[l..r]
function merge_sort(l, r)
  // Base case: 1 element is always sorted
  if (l = r) then return;
  m := (l + r)/2;
  // We need to sort both subsequences l..m, m+1..r
  merge_sort(l, m);
  merge_sort(m + 1, r);
  // Finally: merge the two sorted sequences
  merge(l, m, r);
```

Recursion Trees: Merge Sort (page 2)

– In the previous code we used a merge function:

```
// Merge two sorted sequences l..m, m+1..r
function merge(l, m, r)
  copy A[l..m] to L; L[m – l + 2] := infinity;
  copy A[m + 1..r] to R; R[r – m + 1] := infinity;
  i := 1;   j := 1;   k := l;
  while (L[i] < infinity or R[j] < infinity) do
    if L[i] <= R[j] then
         A[k] := L[i];
         i := i + 1;    k := k + 1;
    else
         A[k] := R[j];
         j := j + 1;   k := k + 1;
```

Recursion Trees: Merge Sort (page 3)

– Given the recurrence for $T(n)$:

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \Theta(n) & \text{if } n > 1 \end{cases}$$

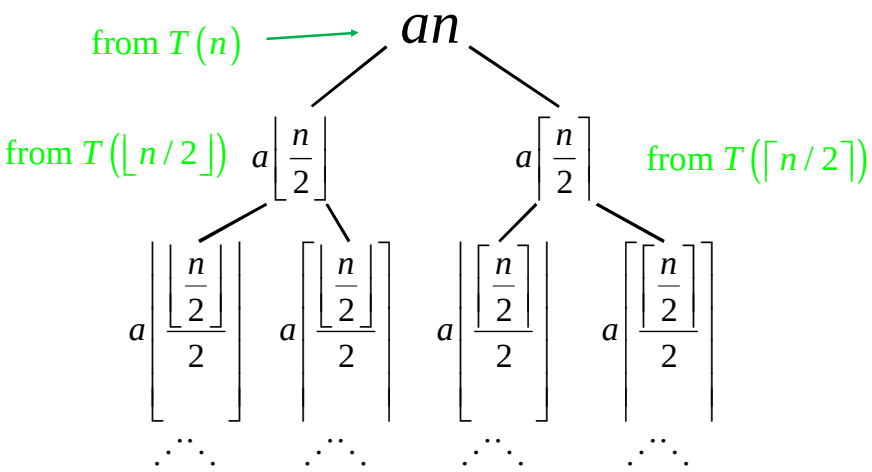
we display the computation in the form of a **tree** (one node for each time the function is invoked).

– Label each node in the tree with the **combine time cost** incurred at that node.

– **Leaves** are labeled with **base case cost** $T(0)$.

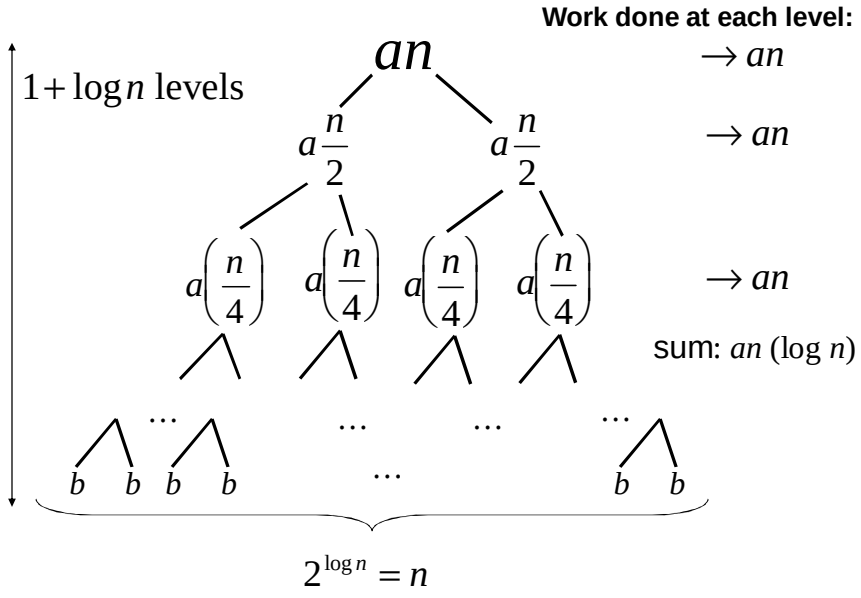
– The **sum of all costs** (including the leaves) is the **total running time** of the algorithm, so we sum up all costs in a convenient way (usually level by level)

Recursion Trees: Merge Sort (page 4)



- This is getting messy, so we use a sloppy form of the recurrence:

Recursion Trees: Merge Sort (page 5)



Working With Recursion Trees

1. Figure out general pattern of time cost labels.
2. Sum the first few levels.
3. Figure out the pattern of the level sums.
4. Compute the sum of all internal nodes.
5. Figure out the height of the tree.
6. Compute the number of the base case nodes.
7. Add up all leaves of the tree.
8. Compute the final total of all the labels.

The Master Theorem

- The Master theorem gives the solution of recurrences of the form: $T(n)=a\,T(n/b) + f(n)$
  - There are three cases, depending on the relationships among a, b, and f.
  - The theorem is applicable even when the recurrence involves the standard floors and ceilings in the partition, so long as the sloppy form looks like $T(n)=a\,T(n/b) + f(n)$.

Statement of Master Theorem

- Theorem. Let $a \geq 1$, $b > 1$ be constants. Let $T(n)$ be such that

$$T(n) = a\,T(n/b) + f(n)$$

where n/b denotes either $ceiling(n/b)$ or $floor(n/b)$. Then:

$$T(n) \in \begin{cases} \Theta(n^{\log_b a}) & \text{if } f(n) \in O(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a} \lg n) & \text{if } f(n) \in \Theta(n^{\log_b a}) \\ \Theta(f(n)) & \text{if } f(n) \in \Omega(n^{\log_b a + \varepsilon}) \\ & \text{and } a\,f(n/b) \leq c\,f(n) \end{cases}$$

Master Theorem

- There are cases not covered by this theorem.
  - For example: $f(n) \in \Theta(n^x \log n)$.
    - Cases 1 and 2 obviously do not apply.
    - Case 3 also does not apply because $n^x \log n$ is not in $\Omega(n^{\log_b a + \varepsilon}) = \Omega(n^{x+\varepsilon})$ no matter how small $\varepsilon > 0$ is.
    - Do not forget that cn^{ε} will eventually get larger than $\log n$ if $\varepsilon > 0$.

Using the Master Theorem

- Example: $T(n) = 3T(n/2) + \Theta(n)$
  - from multiplication of long numbers
  - Here: $a = 3$, $b = 2$, $x = \log_2 3 = 1.5849$.
  - Note: $f(n) < cn$ for n large enough implies $f(n) \in O(n^{1.5849 - \varepsilon})$ and the first case holds.
    - For example we could pick $\varepsilon = .0849$ to get $f(n) \in O(n^{1.5})$.
      Consequently: $T(n) \in \Theta(n^{1.5849})$.

- Example: $T(n) = T(n/2) + 1$
  - from binary search
  - In this recursion: $a = 1$, $b = 2$, $x = \log_2 1 = 0$, so $n^x = 1$, and the second case holds, implying:
    $T(n) \in \Theta(n^x \log n) = \Theta(\log n)$.

Using the Master Theorem (cont.)

- Example: $T(n) = 3T(n/3) + n^2$
  - Here: $a = 3$, $b = 3$, $x = \log_3 3 = 1$, $n^x = n$, so the third case holds implying, $T(n) \in \Theta(f(n)) = \Theta(n^2)$.
- It is possible that no case holds, for example:
  - Consider: $T(n) = 2T(n/2) + n \log n$
  - Then: $x = \log_2 2 = 1$.
  - But: $f(n) = n \log n$ is not in $O(n^{1-\varepsilon})$, is not in $\Theta(n)$, and is not in $\Omega(n^{1+\varepsilon})$.
  - So, we cannot apply the Master Theorem. ☹

Proof Sketch of Master Theorem

- See CLRS: Figure 4.3, p. 77:
  - We assume n is a power of b; then floors/ceilings can be removed.
  - The text uses a recursion tree in which every internal node has a children.
  - Cost of root is $f(n)$, of its children $f(n/b)$, of their children $f(n/b^2)$, and so on…
  - Height of the tree is thus $\log_b n$.
  - Cost at i^{th} level is $a^i f(n/b^i)$.
  - Sum is: $$T(n) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} a^i f(n/b^i)$$
    - Short hand version: $T(n) = LEAVES(n) + LEVELS(n)$.

Proof Sketch of Master Theorem: **Heavy Leaves**

- If $f(n) \in O(n^{x-\varepsilon})$ for some $\varepsilon > 0$: *Where* $x = \log_b a$
  - We will show that the running time is dominated by the $LEAVES(n)$ term which is $\Theta(n^x)$.
    - Need to show that $LEVELS(n)$ is the same order or a lower-order term, i.e.: $LEVELS(n) \in O(LEAVES(n))$.
    - Doing this means we can ignore $LEVELS(n)$ and hence we will get $T(n) \in \Theta(n^x) = \Theta(n^{\log_b a})$ as required.
  - So let us prove $LEVELS(n) \in O(n^x)$:
    - Using $f(n) \in O(n^{\log_b a - \varepsilon}) = O(n^{x-\varepsilon})$ that typifies case 1 we get:
$$LEVELS(n) \in O\left(\sum_{j=0}^{\log_b n - 1} a^j (n/b^j)^{x-\varepsilon} \right).$$

- Then:

$$\sum_{j=0}^{\log_b n-1} a^j (n/b^j)^{x-\varepsilon} = n^{x-\varepsilon} \sum_{j=0}^{\log_b n-1} \frac{a^j}{b^{xj}} (b^{\varepsilon j})$$

$$= n^{x-\varepsilon} \sum_{j=0}^{\log_b n-1} b^{\varepsilon j}$$

$$= n^{x-\varepsilon} \left(\frac{b^{\varepsilon \log_b n} - 1}{b^{\varepsilon} - 1} \right)$$

$$\in O\left(n^{x-\varepsilon} n^{\varepsilon}\right) = O\left(n^{x}\right).$$

n^{ε}

DON'T FORGET:
Setting
$x = \log_b a$
is the same as
$a = b^x$.

Sum of an n term power series.

Proof Sketch of Master Theorem: **Heavy Levels**

- If $f(n) \in \Theta(n^x)$:
  - This is the "**balanced case**" where the work is equally distributed between all levels of the tree.
  - We will show that each level contributes to the total time by the term $\Theta(n^x)$.
    - Once we have proven this then:
  - Since there are $\log_b n \in \Theta(\log n)$ levels, total time is $LEVELS(n) + LEAVES(n) \in \Theta(n^x \log n) + \Theta(n^x) = \Theta(n^x \log n)$.
  - It will suffice to prove that the time contributed by j^{th} level is $\Theta(f(n))$, that is: $a^j f(n/b^j) \in \Theta(f(n))$.

- Note:

$$a^j f\left(\frac{n}{b^j}\right) = \Theta\left(a^j \left(\frac{n}{b^j}\right)^x\right)$$

$$= \Theta\left(n^x \left(\frac{a}{b^x}\right)^j\right)$$

$$= \Theta\left(n^x\right)$$

$$= \Theta\left(f\left(n\right)\right).$$

Don't forget in this second case:
$f(n) \in \Theta(n^{\log_b a}) = \Theta(n^x)$

As required.

Proof Sketch of Master Theorem: **Heavy Top**

- If $f(n) \in \Omega(n^{x+\varepsilon})$ for some $\varepsilon > 0$, and if the regularity condition holds
  - The regularity condition is: $a\, f(n/b) \leq c\, f(n)$ for some $c < 1$ and n sufficiently large:
  - In this case the running time is dominated by the "combine" step at the top-most level.
  - First, we want to prove: $T(n) \in \Omega(f(n))$.
    - Note, that $f(n)$ is part of:

$$T\left(n\right) = \Theta(n^{\log_b a}) + \sum_{i=0}^{\log_b n - 1} a^i f\left(n / b^i\right)$$

and all terms in this sum are non-negative.

– Therefore $T(n) \geq f(n)$ implying $T(n) \in \Omega(f(n))$.
– Second, we want to prove $T(n) \in O(f(n))$:
  • First of all, the sum is dominated by the $LEVELS(n)$ part (because $LEAVES(n) \in o(f(n))$).
  • From the regularity condition, we have:

$$LEVELS(n) \leq f(n) + cf(n) + c^2 f(n) + \ldots$$
$$= f(n)\left(\frac{1}{1-c}\right)$$

Recall $c < 1$.

Note: regularity condition implies $a^i f(n/b^i) \leq c^i f(n)$.

  • Therefore $T(n) \in O(f(n))$ as required. So: $T(n) \in \Theta(f(n))$ because $T(n) \in \Omega(f(n))$ and $T(n) \in O(f(n))$.

Removing Floors and Ceilings

– For n an exact power of b, there is no difference between the sloppy and precise forms of a recurrence.
– The text discusses reasoning about precise form in section 4.4.2. (This section is optional reading).

Using the Master Theorem

- State your recurrences in precise form and in sloppy form.
- If The Master Theorem applies, use it on sloppy form, then the solution will apply to the precise case.
- If The Master Theorem does not apply, use recursion trees to compute the initial guess, then prove by induction.

Guessing and Checking

- The guessing and checking method for solving recurrences consists of guessing the form of the function $T(n)$, which is then followed by an induction to prove that the guess was correct.
- Example:

  $T(n) = T(n\text{-}1) + 1$

  We can try a few values:

  $T(n) = T(n\text{-}2) + 1 + 1 = T(n\text{-}2) + 2$

  $= T(n\text{-}3) + 1 + 2 = T(n\text{-}3) + 3$

  …

  $= T(0) + n = n$ assuming $T(0) = 0$

  So it looks like $T(n) = n$ is the solution.

Guessing and Checking

- Looks like $T(n) = n$ is the solution. We can confirm this using induction.
- Base case $n = 0$, $T(0) = 0$. OK.
- Assume true for $n - 1$, i.e. $T(n - 1) = n - 1$. Show it is true for n.
  - We know that $T(n) = T(n - 1) + 1$, which by the induction hypothesis equals $T(n) = (n - 1) + 1 = n$ as required.

Guessing and Checking

- Another example: $T(n) = 3T(n/3) + n$

  Looks similar to MergeSort, so we guess

  $$T(n) = n \log_3(n)$$
- By induction: Base case $n = 1$, $T(1) = 0$.
- Assume true for $m \leq n - 1$. Show it's true for n.
  - We know that $T(n) = 3T(n/3) + n$, which by induction hypothesis equals

    $T(n) = 3\,(n/3 \cdot \log_3(n/3)) + n$

    $= n \log_3(n/3) + n$

    $= n \log_3(n) - n \log_3(3) + n$

    $= n \log_3(n) - n + n$

    $= n \log_3(n)$.

Guessing and Checking

- Guessing is a mysterious art.
- We can compare the recurrence to similar recurrences.
- Use certain unspecified parameters in our guesses and fix them as necessary during the course of the proof.
  - For example, in the merge sort recurrence, we guess an upper bound of $cn \log n$ for constant c (to be adjusted later).
    - See the "More Thorough Merge Sort Analysis" later.
- G&C is also called "The Substitution Method"

Guessing and Checking

- Be careful when doing the recursion.
- For example consider the recurrence:

$$T(n) \leq 2T(n/2) + cn, \qquad T(1) \leq c.$$

  - We guess $T(n) \in O(n)$ with the following "inductive proof" that $T(n) \leq k\,n$ by induction on n:
  - Base case: true if $k \geq c$.
  - Induction step: Assuming that $T(m) < k\,m$ for all $m < n$, we get

$$T(n) \leq 2T(n/2) + cn \leq 2(k\,n/2) + cn \qquad \text{since } n/2 < n.$$

  - So, $T(n) \leq k\,n + c\,n = (k + c)\,n$.
    If we now jump to the conclusion that $T(n) \in O(n)$ because $T(n)$ is "less than a constant times n" then we are in error because we did not prove the exact form of the recurrence as dictated by the induction hypothesis.
  - In fact: recall, the recurrence had solution $T(n) \in O(n \log n)$.

G&C: Cautionary Notes

- Another error to avoid:
  - Sometimes a small change is the difference between a correct and an incorrect guess.
- For example consider the recurrence:

$$T(n) \leq 2T(n/2) + c, \quad T(1) = c.$$

  - First guess: $T(n) \in O(n)$ with the following inductive proof that $T(n) \leq k\,n$ (induction on n):
  - Base case: true if $k \geq c$.
  - Induction step: Assuming that $T(m) < km$ for all $m < n$, we get:

$$T(n) \leq 2T(n/2) + c \leq 2(k\,n/2) + c \qquad \text{since } n/2 < n.$$

    So, $T(n) \leq k\,n + c$.

  Since this is not the same form as the induction hypothesis, should we try for $T(n) \in O(n \log n)$? No! Because:

G&C: Cautionary Notes (cont.)

- A small modification gives us $T(n) \in O(n)$:
- Consider the same recurrence:

$$T(n) \leq 2T(n/2) + c, \quad T(1) = c.$$

  - But now use the guess that $T(n) \leq kn - b$:
  - Base case: true if $k - b \geq c$.
  - Induction step:
    Assuming that $T(m) < km - b$ for all $m < n$, we get:

$$T(n) \leq 2T(n/2) + c \leq 2(kn/2 - b) + c \qquad \text{since } n/2 < n.$$

    So, $T(n) \leq kn - 2b + c = kn - b - b + c < kn - b$ as long as: $b > c$.
  - Now having proved that $T(n) \leq kn - b$, we are justified in claiming that $T(n) \in O(n)$.