CSSE 230

Recurrence Relations
Sorting overview

$$T(N) = \begin{cases} \theta(N^{log_b a}) & \text{if } a > b^k \\ \theta(N^k logN) & \text{if } a = b^k \\ \theta(N^k) & \text{if } a < b^k \end{cases}$$

After today, you should be able to…
…write recurrences for code snippets
…solve recurrences using telescoping, recurrence trees, and the master method

More on Recurrence Relations

A technique for analyzing recursive algorithms

Recap: Recurrence Relation

- An equation (or inequality) that relates the N^{th} element of a sequence to certain of its predecessors (recursive case)
- Includes an initial condition (base case)
- **Solution**: A function of N.

Example. Solve using backward substitution.

$T(N) = 2T(N/2) + N$
$T(1) = 1$

Solution strategies

Forward substitution
Backward substitution

Simple
Often can't solve difficult relations

Recurrence trees

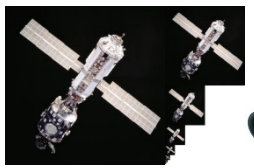
Visual
Great intuition for div-and-conquer

Telescoping

Widely applicable
Difficult to formulate
Not intuitive

Master Theorem

Immediate
Only for div-and-conquer
Only gives Big-Theta



Which telescope?

http://wiki.ogre3d.org/Mipmapping

Selection Sort: iterative version

```
static void selectionSort(int[] a) {
    for (int last = a.length-1; last > 0; last--) {
        int largest = a[0];
        int largestPosition = 0;
        for (int j=1; j<=last; j++) {
            if (largest < a[j]) {
                largest = a[j];
                largestPosition = j;
            }
        }
        a[largestPosition] = a[last];
        a[last] = largest;
    }
}
```

What's N?

Selection Sort: recursive version

```
static void selectionSortRec(int[] a) {
    selectionSortRec(a, a.length-1);
}

static void selectionSortRec(int[] a, int last) {
    if (last == 0) return;
    int largest = a[0];
    int largestPosition = 0;
    for (int j=1; j<=last; j++) {
        if (largest < a[j]) {
            largest = a[j];
            largestPosition = j;
        }
    }
    a[largestPosition] = a[last];
    a[last] = largest;
    selectionSortRec(a, last-1);
}
```

What's N?

Telescoping

- Basic idea: Set up equations so that when we sum all L sides and all R sides, we get an equation with lots of cancelation.
- Example: T(1) = 0, T(N) = T(N – 1) + N – 1

$$
\begin{aligned}
T(N) &= \cancel{T(N-1)} + N - 1 \\
\cancel{T(N-1)} &= \cancel{T(N-2)} + N - 2 \\
\cancel{T(N-2)} &= \cancel{T(N-3)} + N - 3 \\
&\vdots \\
\cancel{T(2)} &= \cancel{T(1)} + 1 \\
\cancel{T(1)} &= 0 \\
\hline
T(N) &= \sum_{i=1}^{N-1} i \qquad = \frac{(N-1)N}{2}
\end{aligned}
$$

Telescoping – View #2 of: T(x) = T(x–1) + x–1

$$
\begin{aligned}
T(N) &= T(N-1) + N-1 \\
T(N-1) &= T(N-2) + N-2 \\
T(N-2) &= T(N-3) + N-3 \\
&\vdots \\
T(2) &= T(1) + 1 \\
T(1) &= 0
\end{aligned}
$$

- x is a variable
- N is a constant

1. Use recurrence to create numerous equations
2. Add all these equations
3. Then use algebraic techniques to simplify

- In this example subtract like terms from both sides

$$T(1) + T(2) + \ldots + T(N-2) + T(N-1) + T(N) =$$

$$0 + T(1) + 1 + \ldots + T(N-3) + N-3 + T(N-2) + N-2 + T(N-1) + N-1$$

Telescoping – T(x) = 2T(x/2) + x

- In general, need to tweak the relation somehow so successive terms cancel

- Example: T(1) = 1, T(N) = 2T(N/2) + N
  where $N = 2^k$ for some k
- Divide by N to get a "piece of the telescope":

$$T(N) = 2T(\frac{N}{2}) + N$$

$$\Longrightarrow \frac{T(N)}{N} = \frac{2T(\frac{N}{2})}{N} + 1$$

$$\Longrightarrow \frac{T(N)}{N} = \frac{T(\frac{N}{2})}{\frac{N}{2}} + 1$$

Etc.

Telescoping

Weiss, Mark Allen.
Data structures & problem solving using Java / Mark Allen Weiss.-- 4th ed.

Theorem 7.4 (Method 2)

We divide Equation 7.6 by N, yielding a new basic equation:

$$\frac{T(N)}{N} = \frac{T(N/2)}{N/2} + 1$$

$$\begin{aligned}
\frac{T(N)}{N} &= \frac{T(N/2)}{N/2} + 1 \\
\frac{T(N/2)}{N/2} &= \frac{T(N/4)}{N/4} + 1 \\
\frac{T(N/4)}{N/4} &= \frac{T(N/8)}{N/8} + 1 \\
&\cdots \\
\frac{T(2)}{2} &= \frac{T(1)}{1} + 1
\end{aligned} \tag{7.8}$$

Now we add the collective in Equation 7.8. That is, we add all the terms on the left-hand side and set the result equal to the sum of all the terms on the right-hand side. The term $T(N/2) / (N/2)$ appears on both sides and thus cancels. In fact, virtually all the terms appear on both sides and cancel. This is called a *telescoping sum*. After everything is added, the final result is

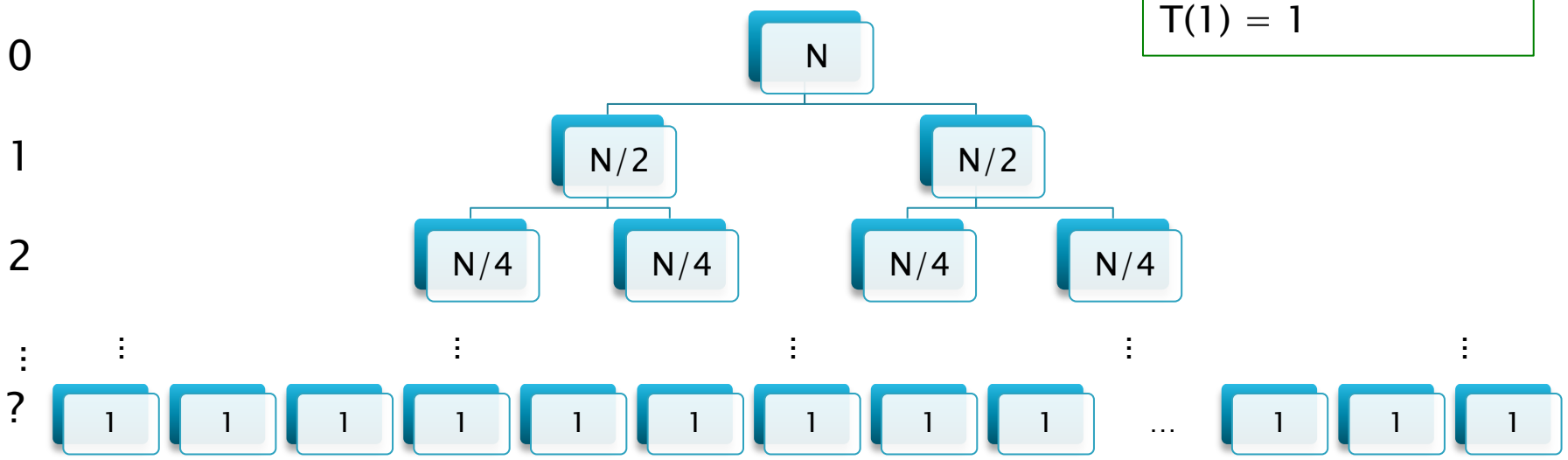
$$\frac{T(N)}{N} = \frac{T(1)}{1} + \log N$$

because all the other terms cancel and there are $\log N$ equations. Thus all the 1s at the end of these equations sum to $\log N$. Multiplying through by N gives the final answer, as before.

Note that, if we had not divided through by N at the start of the solution, the sum would not have telescoped. Deciding on the division required to ensure a telescoping sum requires some experience and makes the method a little more difficult to apply than the first alternative. However, once you have found the correct divisor, the second alternative tends to produce scrap work that fits better on a standard sheet of paper, leading to fewer mathematical errors.

Recursion tree

Recurrence:
$T(N) = 2T(N/2) + N$
$T(1) = 1$



- How many nodes at level j? 2^j
- How much work at each node? $(N/2^j)$
- How much work at level j? $2^j (N/2^j) = N$
- Index of the 1st level 0
- Index of last level? $\log_2 (N)$

Total: $T(N) = \sum_{j=0}^{log_2(N)} N = N + N + N + \ldots + N = N + log_2(N)$

Master Theorem

- For Divide-and-conquer algorithms
  - Divide data into one or more parts
    - Each part is of the same size
    - Each part is smaller than original size of the problem
  - Solve problem on one or more of those parts
  - Combine "parts" solutions to solve whole problem
- Examples
  - Binary search
  - Merge Sort
  - MCSS recursive algorithm we studied last time

Theorem 7.5 in Weiss

Master Theorem

a = # of recursive calls
N/b = size of subproblem
$\theta(N^k)$ = D(N) + C(N)
D(N) = time to divide
C(N) = time to combine

- For any recurrence in the form:

$$T(N) = aT(N/b) + \theta(N^k)$$
$$\text{with } a \geq 1, b > 1$$

- The solution is

$$T(N) = \begin{cases} \theta(N^{log_b a}) & \text{if } a > b^k \\ \theta(N^k logN) & \text{if } a = b^k \\ \theta(N^k) & \text{if } a < b^k \end{cases}$$

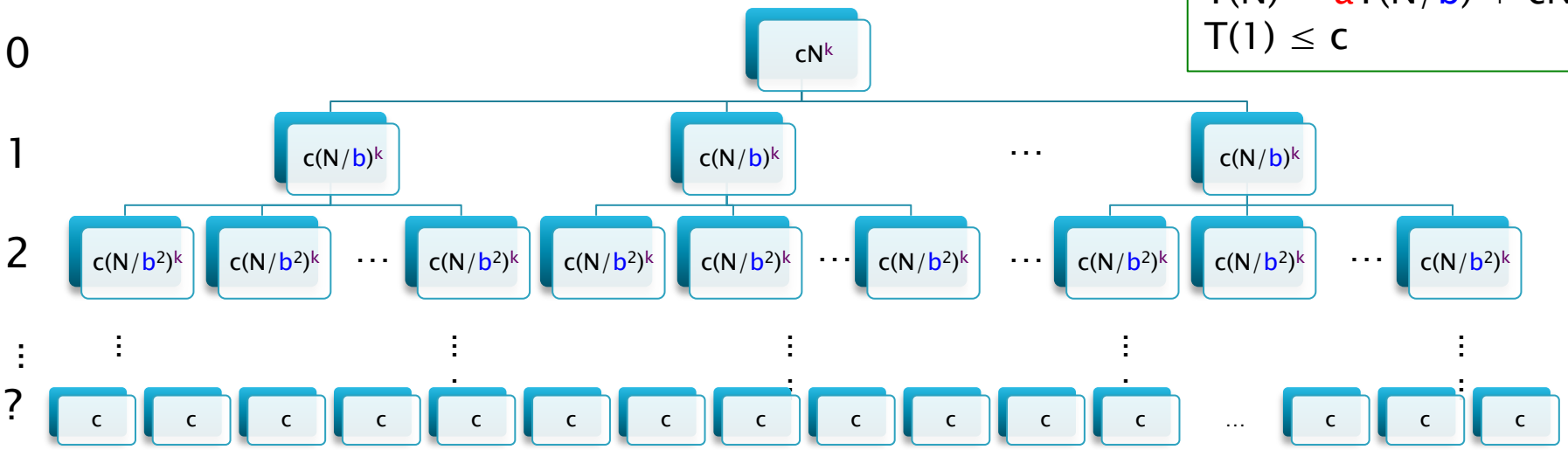
Example: 2T(N/4) + N

a = 2
b = 4
$\theta(N^k) = \theta(N^1)$ $\quad so\ k = 1$

Theorem 7.5 in Weiss

Master Recurrence Tree

Recurrence:
$T(N) = aT(N/b) + cN^k$
$T(1) \leq c$



- How many nodes at level i? a^i
- How much work at level i? $a^i\, c(N/b^i)^k = cN^k(a/b^k)^i$
- Index of last level? $\log_b N$

Summation: $$T(N) \leq cN^k \sum_{i=0}^{\log_b N} \left(\frac{a}{b^k}\right)^i$$

Interpretation

- Upper bound on work at level i: $cN^k \left(\frac{a}{b^k}\right)^i$

- a = "Rate of subproblem proliferation" 😟
- b^k = "Rate of work shrinkage" 😃

Case	😟 $a < b^k$ 😃	😟 $a = b^k$ 😃	😟 $a > b^k$ 😃
As level i increases…	😃 work goes down!	😐 work stays same	😟 work goes up!
T(N) dominated by work done at…	Root of tree	Every level similar	Leaves of tree
Master Theorem says T(N) in…	$\Theta(N^k)$	$\Theta(N^k \log N)$	$\Theta(N^{\log_b a})$

Master Theorem – End of Proof

$$cN^k \sum_{i=0}^{\log_b N} \left(\frac{a}{b^k}\right)^i$$

- Case 1. a < b^k

$$cN^k \left(\frac{1-(a/b^k)^{\log_b N+1}}{1-(a/b^k)}\right) \approx cN^k \left(\frac{1}{1-(a/b^k)}\right)$$

- Case 2. a = b^k

$$cN^k \sum_{i=0}^{\log_b N} 1 = cN^k(\log_b N + 1)$$

- Case 3. a > b^k

$$cN^k \left(\frac{(a/b^k)^{\log_b N+1}-1}{(a/b^k)-1}\right) \approx cN^k(a/b^k)^{\log_b N} = ca^{\log_b N} = cN^{\log_b a}$$

Summary: Recurrence Relations

- Analyze code to determine relation
  - Base case in code gives base case for relation
  - Number and "size" of recursive calls determine recursive part of recursive case
  - Non-recursive code determines rest of recursive case
- Apply a strategy
  - Guess and check (substitution)
  - Telescoping
  - Recurrence tree
  - Master theorem

Sorting overview

- Quick look at several sorting methods
- Focus on quicksort
- Quicksort average case analysis

Elementary Sorting Methods

- Name as many as you can
- How does each work?
- Running time for each (sorting N items)?
  - best
  - worst
  - average
  - extra space requirements
- Spend 10 minutes with a group of 2-3, answering these questions. Then we will summarize

Put list on board

INEFFECTIVE SORTS

```
DEFINE HALFHEARTEDMERGESORT(LIST):
    IF LENGTH(LIST) < 2:
        RETURN LIST
    PIVOT = INT(LENGTH(LIST) / 2)
    A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
    B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
    // UMMMMM
    RETURN [A, B] // HERE. SORRY.
```

```
DEFINE FASTBOGOSORT(LIST):
    // AN OPTIMIZED BOGOSORT
    // RUNS IN O(N LOG N)
    FOR N FROM 1 TO LOG(LENGTH(LIST)):
        SHUFFLE(LIST):
        IF ISSORTED(LIST):
            RETURN LIST
    RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

```
DEFINE JOBINTERVIEWQUICKSORT(LIST):
    OK SO YOU CHOOSE A PIVOT
    THEN DIVIDE THE LIST IN HALF
    FOR EACH HALF:
        CHECK TO SEE IF IT'S SORTED
            NO, WAIT, IT DOESN'T MATTER
        COMPARE EACH ELEMENT TO THE PIVOT
            THE BIGGER ONES GO IN A NEW LIST
            THE EQUAL ONES GO INTO, UH
            THE SECOND LIST FROM BEFORE
        HANG ON, LET ME NAME THE LISTS
            THIS IS LIST A
            THE NEW ONE IS LIST B
        PUT THE BIG ONES INTO LIST B
        NOW TAKE THE SECOND LIST
            CALL IT LIST, UH, A2
        WHICH ONE WAS THE PIVOT IN?
        SCRATCH ALL THAT
        IT JUST RECURSIVELY CALLS ITSELF
        UNTIL BOTH LISTS ARE EMPTY
            RIGHT?
        NOT EMPTY, BUT YOU KNOW WHAT I MEAN
    AM I ALLOWED TO USE THE STANDARD LIBRARIES?
```

```
DEFINE PANICSORT(LIST):
    IF ISSORTED(LIST):
        RETURN LIST
    FOR N FROM 1 TO 10000:
        PIVOT = RANDOM(0, LENGTH(LIST))
        LIST = LIST[PIVOT:] + LIST[:PIVOT]
        IF ISSORTED(LIST):
            RETURN LIST
    IF ISSORTED(LIST):
        RETURN LIST:
    IF ISSORTED(LIST): //THIS CAN'T BE HAPPENING
        RETURN LIST
    IF ISSORTED(LIST): // COME ON COME ON
        RETURN LIST
    // OH JEEZ
    // I'M GONNA BE IN SO MUCH TROUBLE
    LIST = [ ]
    SYSTEM("SHUTDOWN -H +5")
    SYSTEM("RM -RF ./")
    SYSTEM("RM -RF ~/*")
    SYSTEM("RM -RF /")
    SYSTEM("RD /S /Q C:\*") //PORTABILITY
    RETURN [1, 2, 3, 4, 5]
```

http://www.xkcd.com/1185/

Stacksort connects to StackOverflow, searches for “sort a list”, and downloads and runs code snippets until the list is sorted.