



CSSE 230

Recurrence Relations Sorting overview

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

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*

Selection Sort

```

public static void selectionSort(int[] a) {
    //Sorts a non-empty array of integers.

    for (int last = a.length-1; last > 0; last--) {
        // find largest, and exchange with last
        int largest = a[0];
        int largePosition = 0;

        for (int j=1; j<=last; j++)
            if (largest < a[j]) {
                largest = a[j];
                largePosition = j;
            }
        a[largePosition] = a[last];
        a[last] = largest;
    }
}

```

What's N?

Selection Sort: recursive version

```

void sort(a) { sort(a, a.length-1); }

void sort(a, last) {
    if (last == 0) return;
    find max value in a from 0 to last
    swap max to last
    sort(a, last-1)
}

```

What's N?

2-3

Telescoping

- Basic idea: 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”:

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

$$\Rightarrow \frac{T(N)}{N} = \frac{2T\left(\frac{N}{2}\right)}{N} + 1$$

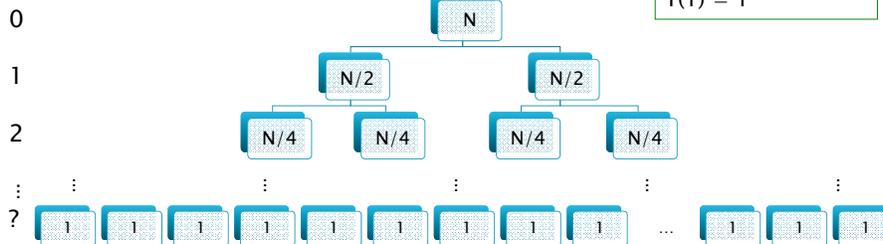
$$\Rightarrow \frac{T(N)}{N} = \frac{T\left(\frac{N}{2}\right)}{\frac{N}{2}} + 1$$



4

Recursion tree

Level



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

- How many nodes at level i ? 2^i
- How much work at level i ? $2^i (N/2^i) = N$
- Index of last level? $\log_2 N$

Total: $T(n) = \sum_{i=0}^{\log N} N = N(\log N + 1)$

5

Master Theorem

- ▶ For Divide-and-conquer algorithms
 - Divide data into one or more parts **of the same size**
 - 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

5-7

Master Theorem

- ▶ For any recurrence in the form:

$$T(N) = aT(N/b) + \theta(N^k)$$

with $a \geq 1, b > 1$

- ▶ The solution is

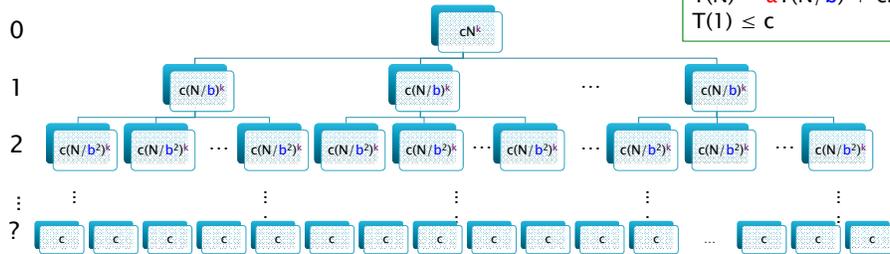
$$T(N) = \begin{cases} \theta(N^{\log_b a}) & \text{if } a > b^k \\ \theta(N^k \log N) & \text{if } a = b^k \\ \theta(N^k) & \text{if } a < b^k \end{cases}$$

Example: $2T(N/4) + N$

Theorem 7.5 in Weiss

Master Recurrence Tree

Level



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

8-10

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 three, 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:])
// UMMMMMM
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 JOBITERVIEWQUICKSORT(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.