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 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.

Solve Simple Recurrence Relations

- One strategy: **find a pattern**
- Examples:
  - $T(0) = 0$, $T(N) = 2 + T(N-1)$
  - $T(0) = 1$, $T(N) = 2\,T(N-1)$
  - $T(0) = T(1) = 1$, $T(N) = T(N-2) + T(N-1)$
  - $T(0) = 1$, $T(N) = N\,T(N-1)$
  - $T(0) = 0$, $T(N) = T(N-1) + N$
  - $T(1) = 1$, $T(N) = 2\,T(N/2) + N$

    (just consider the cases where $N=2^k$)

Other solution strategies for recurrence relations

- Find patterns
- **Telescoping**
- The master theorem

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?

Another Strategy: 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":

$$
\begin{aligned}
T(N) &= 2T(\frac{N}{2}) + N \\
\Longrightarrow \frac{T(N)}{N} &= \frac{2T(\frac{N}{2})}{N} + 1 \\
\Longrightarrow \frac{T(N)}{N} &= \frac{T(\frac{N}{2})}{\frac{N}{2}} + 1
\end{aligned}
$$

Another Strategy: Master Theorem

- For Divide-and-conquer algorithms
  - Divide data into two 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

Divide and Conquer Recurrences all have the same form

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

$$\text{with } a \geq 1, b > 1$$

- Recursive part
  - a = number of parts we solve
  - b = number of parts we divide into

- Non-recursive part
  - $f(N^k)$ = overhead of dividing and combining (or, the amount of work done each recursion)

The Master Theorem is convenient, but only works for divide and conquer recurrences

- For any recurrence in the form:

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

- The solution is

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

Example: 2T(N/4) + N

Theorem 7.5 in Weiss

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 one of three strategies
  - Guess and check
  - Telescoping
  - 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 three, answering these questions. Then we will summarize

Put list on board