CSSE 230

Recurrence Relations
Sorting overview

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

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: **guess and check**

- 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 $N=2^k$)

Another Strategy

- **Substitution**
- $T(1) = 1$, $T(N) = 2\ T(N/2) + N$
  (just consider $N=2^k$)
- Suppose we substitute $N/2$ for N in the recursive equation?
  - We can plug the result into the original equation!

Solution Strategies for Recurrence Relations

- Guess and check
- Substitution
- Telescoping and iteration
- The “master” method

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?

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":

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

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

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

A Fourth 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 Recurrence

$$T(N) = aT(\frac{N}{b}) + f(N)$$

$$a \geq 1, b > 1, \text{ and } f(N) = O(N^k)$$

- Recursive:
  - b = number of parts we divide into
  - a = number of parts we solve
- Non-recursive:
  - f(N) = overhead of dividing and combining
- Examples:
  - Binary Search: a =__ , b =___ , k = ___.
  - Merge sort: a =__ , b =___ , k = ___.

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

- For any recurrence relation ***in the form***:

$$T(N) = aT(\frac{N}{b}) + f(N)$$

with $a \geq 1, b > 1,$ and $f(N) = O(N^k)$

- The solution is:

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

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 four strategies
  - Guess and check
  - Substitution (a.k.a. iteration)
  - 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

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.