CSSE 230 Day 20

Recurrence Relations

Reminders/Announcements

- Don't forget the Scrabble team member preference survey on ANGEL. Due Wednesday at 5:00 PM.
- **Thursday's class time**: Work on EditorTrees with your team. Get help from assistants as needed.
  - Missing lecture time is one thing …
- Additional help: Will Anderson will be available
  - Thursday in class, hours 2 and 3.
  - Thursday and Friday hours 9-10 in F-217

Evening lab hours this week (F-217, 7-9 PM)
Tuesday, Thursday, Sunday Brian Lackey
Wednesday: Doug Mann

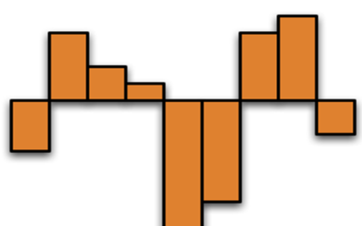
- Today's Agenda:
  - MCSS revisited (recursive version)
  - Analyzing recursive (and other) algorithms by using **recurrence relations**
- Your Questions (and my answers!)

Introduction to Recurrence Relations

- A technique for analyzing recursive algorithms

Recap: Maximum Contiguous Subsequence Sum problem

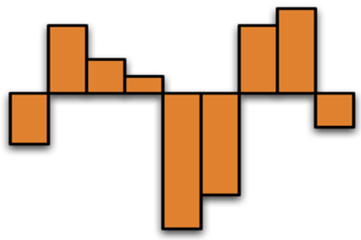
Problem definition: Given a non-empty sequence of n (possibly negative) integers $A_1, A_2, \ldots, A_n$, find the maximum consecutive subsequence $S_{i,j} = \sum_{k=i}^{j} A_k$, and the corresponding values of i and j.



1

Divide and Conquer Approach

- Split the sequence in half
- Where can the maximum subsequence appear?
- Three possibilities :
  - entirely in the first half,
  - entirely in the second half, or
  - **begins** in the first half and **ends** in the second half



Overview of algorithm

1. Using recursion, find the maximum sum of **first** half of sequence
2. Using recursion, find the maximum sum of **second** half of sequence
3. Compute the max of all sums that begin in the first half and end in the second half
   - (Use a couple of loops for this)
4. Choose the largest of these three numbers

2-3

```
private static int maxSumRec( int [ ] a, int left, int right )
{
    int maxLeftBorderSum = 0, maxRightBorderSum = 0;
    int leftBorderSum = 0, rightBorderSum = 0;
    int center = ( left + right ) / 2;

    if( left == right )  // Base case
        return a[ left ] > 0 ? a[ left ] : 0;

    int maxLeftSum  = maxSumRec( a, left, center );
    int maxRightSum = maxSumRec( a, center + 1, right );

    for( int i = center; i >= left; i-- )
    {
        leftBorderSum += a[ i ];
        if( leftBorderSum > maxLeftBorderSum )
            maxLeftBorderSum = leftBorderSum;
    }

    for( int i = center + 1; i <= right; i++ )
    {
        rightBorderSum += a[ i ];
        if( rightBorderSum > maxRightBorderSum )
            maxRightBorderSum = rightBorderSum;
    }

    return max3( maxLeftSum, maxRightSum,
                 maxLeftBorderSum + maxRightBorderSum );
```

So, what's the run-time?

4

Analysis?

- Use a **Recurrence Relation**
  - A function of N, typically written T(N)
  - Gives the run-time as a function of N
  - Two (or more) part definition:
    - Base case, like T(1) = c
    - Recursive case, like T(N) = T(N/2)

So, what's the recurrence relation for the recursive MCSS algorithm?

5-6

```
private static int maxSumRec( int [ ] a, int left, int right )
{
    int maxLeftBorderSum = 0, maxRightBorderSum = 0;
    int leftBorderSum = 0, rightBorderSum = 0;
    int center = ( left + right ) / 2;

    if( left == right )  // Base case
        return a[ left ] > 0 ? a[ left ] : 0;

    int maxLeftSum  = maxSumRec( a, left, center );
    int maxRightSum = maxSumRec( a, center + 1, right );

    for( int i = center; i >= left; i-- )
    {
        leftBorderSum += a[ i ];
        if( leftBorderSum > maxLeftBorderSum )
            maxLeftBorderSum = leftBorderSum;
    }

    for( int i = center + 1; i <= right; i++ )
    {
        rightBorderSum += a[ i ];
        if( rightBorderSum > maxRightBorderSum )
            maxRightBorderSum = rightBorderSum;
    }

    return max3( maxLeftSum, maxRightSum,
                 maxLeftBorderSum + maxRightBorderSum );
```

What's N in the base case?

Recurrence Relation, Formally

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

- Similar to differential equation, but discrete instead of continuous
- Some solution techniques are similar to diff. eq. solution techniques

7-10

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 $N=2^k$)

10

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!

11

Solution Strategies for Recurrence Relations

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

12-13

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?

14

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":

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

15a

A Fourth Strategy: Master Theorem

- For Divide-and-conquer algorithms
  - Divide data into two or more parts
  - 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

15b

Divide and Conquer Recurrence

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

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

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

16

Master Theorem

- For any recurrence relation:

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

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

- 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