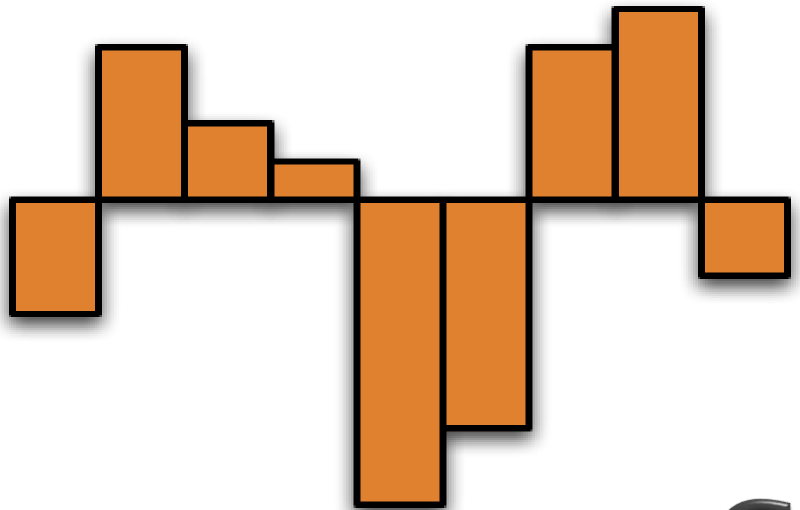


CSSE 230 Day 3

Maximum Contiguous Subsequence Sum

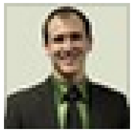
After today's class you will be able to:

- state and solve the MCSS problem on small arrays by observation
- find the exact runtimes of the naive MCSS algorithms

https://openclipart.org/image/2400px/svg_to_png/169467/bow_tie.png

Announcement, 2015

Software Engineering Professionals (SEP)
Information Session
Tuesday, September 8, 2015
6:00 –8:00 p.m.
Moench Hall F225 – CSSE LAB



Andrew Hettlinger ▸ Matt Boutell

November 6 at 12:30pm ·

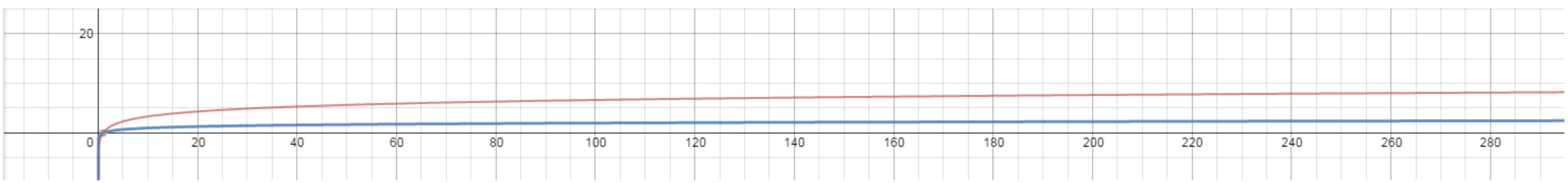
In your class, I never thought I'd actually use big O notation, but now I find myself using it in my complaints to coworkers about how a previous developer would sort a list before doing a binary search to find a single element O(nlogn) + O(logn) instead of just doing a linear search O(n). I feel really nerdy now (as if I didn't before 🙂)

Like · Comment

So why would we ever do binary search?

Homework 1

- Is it true that $\log_a(n)$ is $\theta(\log_b(n))$?
- Complete homework 1 to find out the exciting conclusion!
- Here is the graph for a=2 and b=10:



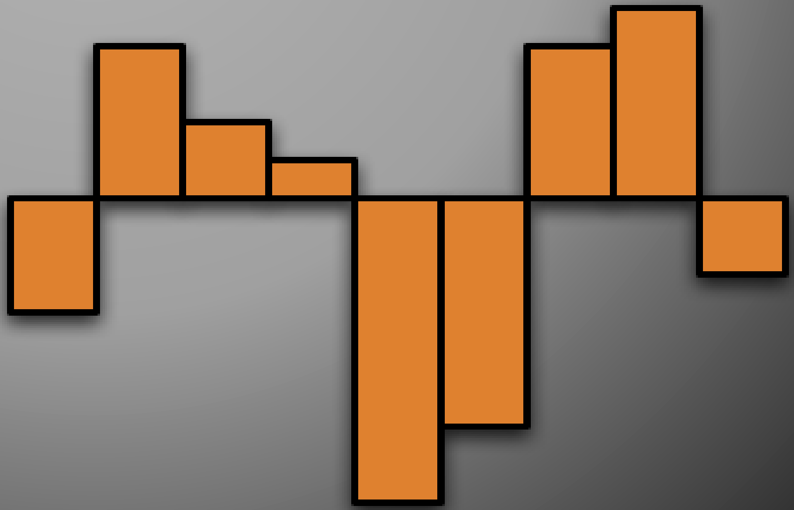
- Is it true that 3^n is $\theta(2^n)$?

Q1

Maximum Contiguous Subsequence Sum

A deceptively deep problem with a surprising solution.

{−3, 4, 2, 1, −8, −6, 4, 5, −2}

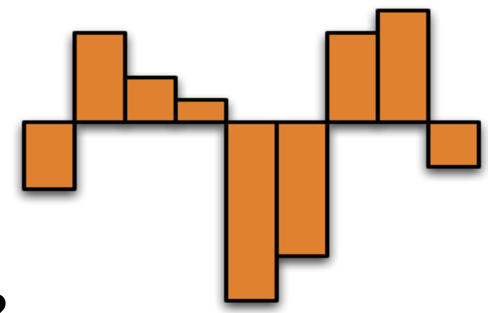


Why do we look at this problem?

- It’s interesting
- Analyzing the obvious solution is instructive
- We can make the program more efficient

A Nice Algorithm Analysis Example

- **Problem**: Given a sequence of numbers, find the maximum sum of a contiguous subsequence.



- **Consider**:
  - What if all the numbers were positive?
  - What if they all were negative?
  - What if we left out “contiguous”?

Q2-4

Formal Definition: Maximum Contiguous Subsequence Sum

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.

- Quiz questions:
  - In {-2, **11, -4, 13**, -5, 2}, $S_{2,4}$ = ?
  - In {1, -3, 4, -2, -1, 6}, what is MCSS?
  - If every element is negative, what's the MCSS?

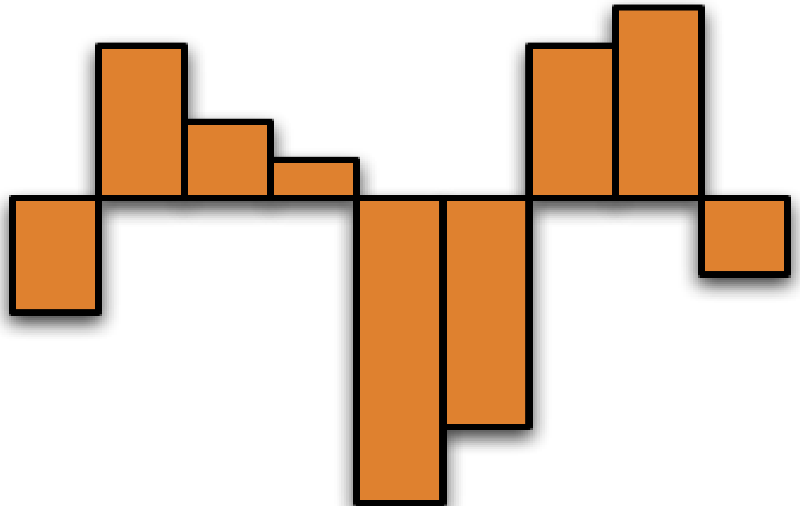
1-based indexing. We'll use when analyzing b/c easier

Write a simple correct algorithm now

Q5

- Must be easy to explain
- Correctness is KING. Efficiency doesn't matter yet.
- 3 minutes

- Examples to consider:
  - {–3, 4, 2, 1, –8, –6, 4, 5, –2}
  - {5, 6, –3, 2, 8, 4, –12, 7, 2}



First Algorithm

Find the sums of *all* subsequences

```
public final class MaxSubTest {
    private static int seqStart = 0;
    private static int seqEnd = 0;

    /* First maximum contiguous subsequence sum algorithm.
     * seqStart and seqEnd represent the actual best sequence.
     */
    public static int maxSubSum1( int [ ] a ) {
        int maxSum = 0;
          //In the analysis we use "n" as a shorthand for "a.length "
        for( int i = 0; i < a.length; i++ )
            for( int j = i; j < a.length; j++ ) {
                int thisSum = 0;

                for( int k = i; k <= j; k++ )
                    thisSum += a[ k ];

                if( thisSum > maxSum ) {
                    maxSum   = thisSum;
                    seqStart = i;
                    seqEnd   = j;
                }
            }
        return maxSum;
    }
```

i: beginning of subsequence

j: end of subsequence

k: steps through each element of subsequence

Where will this algorithm spend the most time?

How many times (exactly, as a function of N = a.length) will that statement execute?

Q6, Q7

Analysis of this Algorithm

- What statement is executed the most often?
- How many times?

```
        //In the analysis we use "n" as a shorthand for "a.length "
for( int i = 0; i < a.length; i++ )
    for( int j = i; j < a.length; j++ ) {
        int thisSum = 0;

        for( int k = i; k <= j; k++ )
            thisSum += a[ k ];
```

Interlude

- Computer Science is no more about computers than astronomy is about ________.

Donald Knuth

Interlude

- Computer Science is no more about computers than astronomy is about <u>telescopes</u>.

Donald Knuth

Where do we stand?

- We showed MCSS is $O(n^3)$.
  - Showing that a **problem** is O(g(n)) is relatively easy – just analyze a known algorithm.
- Is MCSS $\Omega(n^3)$?
  - Showing that a **problem** is Ω (g(n)) is much tougher. How do you prove that it is impossible to solve a problem more quickly than you already can?
  - Or maybe we can find a faster algorithm?

f(n) is O(g(n)) if $f(n) \le cg(n)$ for all $n \ge n_0$
- So O gives an upper bound

f(n) is Ω(g(n)) if $f(n) \ge cg(n)$ for all $n \ge n_0$
- So Ω gives a lower bound

f(n) is θ(g(n)) if $c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0$
- So θ gives a tight bound
- f(n) is θ(g(n)) if it is both O(g(n)) **and** Ω(g(n))

What is the main source of the simple algorithm's inefficiency?

```
//In the analysis we use "n" as a shorthand for "a.length "
for( int i = 0; i < a.length; i++ )
    for( int j = i; j < a.length; j++ ) {
        int thisSum = 0;

        for( int k = i; k <= j; k++ )
            thisSum += a[ k ];
```

- The performance is bad!

Eliminate the most obvious inefficiency...

```
for( int i = 0; i < a.length; i++ ) {
    int thisSum = 0;
    for( int j = i; j < a.length; j++ ) {
        thisSum += a[ j ];

        if( thisSum > maxSum ) {
            maxSum = thisSum;
            seqStart = i;
            seqEnd   = j;
        }
    }
}
```

This is Θ(?)

MCSS is $O(n^2)$

- Is MCSS $\Omega(n^2)$?
  - Showing that a problem is Ω (g(n)) is much tougher. How do you prove that it is impossible to solve a problem more quickly than you already can?

  - Can we find a yet faster algorithm?

f(n) is O(g(n)) if $f(n) \leq cg(n)$ for all $n \geq n_0$
- So O gives an upper bound

f(n) is Ω(g(n)) if $f(n) \geq cg(n)$ for all $n \geq n_0$
- So Ω gives a lower bound

f(n) is θ(g(n)) if $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$
- So θ gives a tight bound
- f(n) is θ(g(n)) if it is both O(g(n)) **and** Ω(g(n))

Q8, Q9

Can we do even better?

Tune in next time for the exciting conclusion!

Think about the 7,2 on the other side of the −12:

{5, 6, −3, 2, 8, 4, −12, 7, 2}