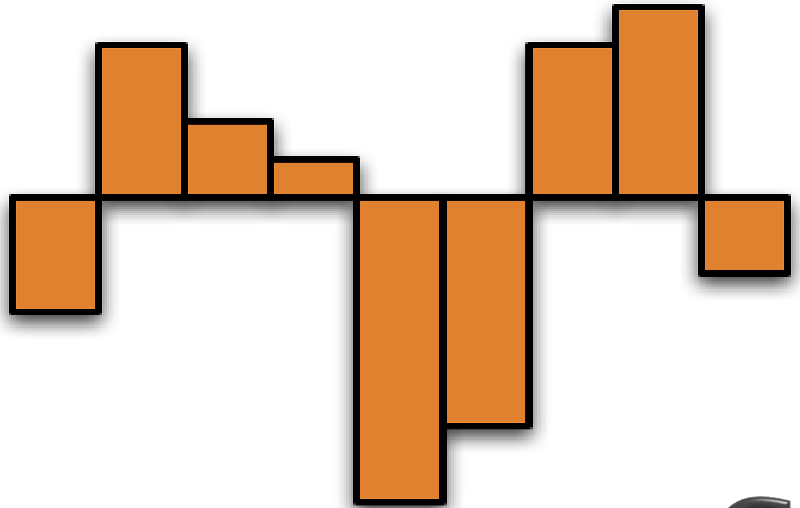
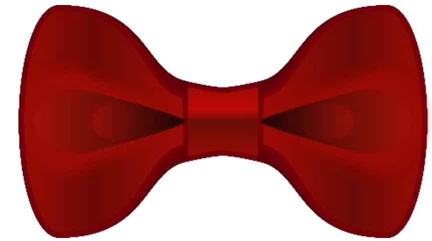


Tie Day #1. Log your bonus points in Moodle



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

Limits and Asymptotics

- Consider the limit

$$\lim_{n \to \infty} \frac{f(n)}{g(n)}$$

- What does it say about asymptotic relationship between f and g if this limit is…
  - 0?
  - finite and non-zero?
  - infinite?

Apply this limit property to the following pairs of functions

1. n and n^2
2. $\log n$ and n (on these questions and solutions ONLY, let $\log n$ mean natural log)
3. $n \log n$ and n^2
4. $\log_a n$ and $\log_b n$ $(a < b)$
5. n^a and a^n $(a > 1)$
6. a^n and b^n $(a < b)$

Recall l'Hôpital's rule:
under appropriate conditions,

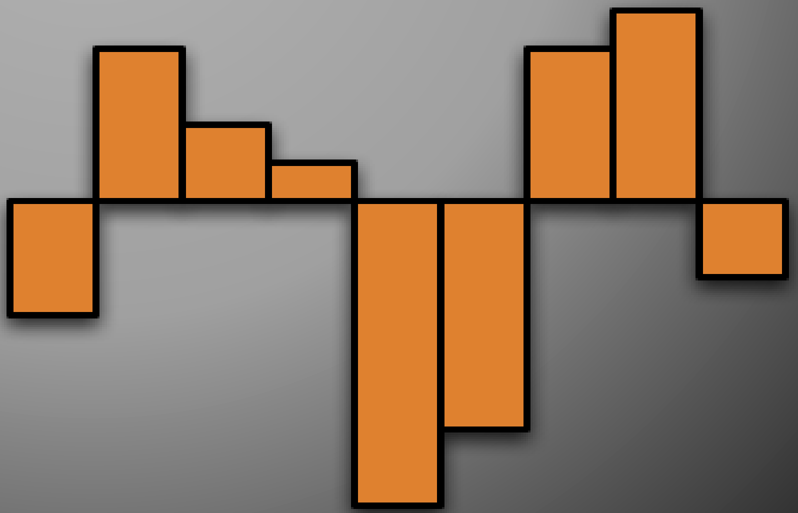
$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

Q13-15

Maximum Contiguous Subsequence Sum

A deceptively deep problem
with a surprising solution.

{−3, 4, 2, 1, −8, −6, 4, 5, −2}

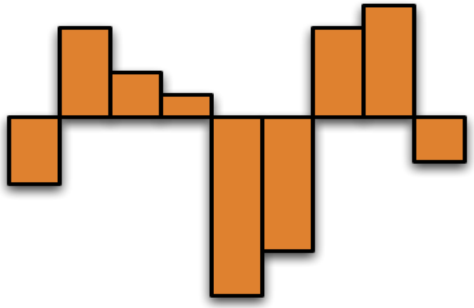


Why do we look at this problem?

- 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?

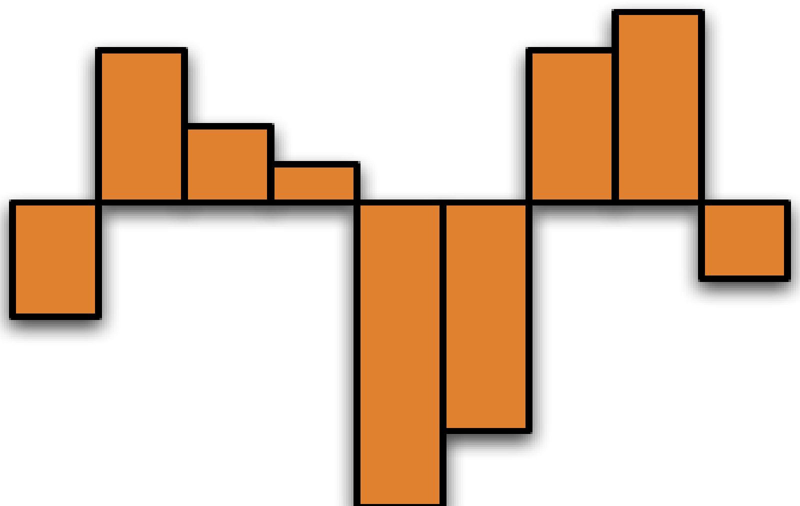
1-based indexing. We'll use when analyzing b/c easier

Write a simple correct algorithm now

Q5

- Must be easy to explain
- Efficiency doesn't matter.
- 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?

Analysis of this Algorithm

- What statement is executed the most often?
- How many times?
- How many triples, **(i,j,k)** with **1≤i≤k≤j≤n** ?

```
        //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 ];
```

Outer numbers could be 0 and n – 1, and we'd still get the same answer.

How to find the exact sum

- By hand
- Using Maple

Q6, Q7

Counting is (surprisingly) hard!

- How many triples, **(i,j,k)** with **1≤i≤k≤j≤n** ?
- What is that as a summation?
  - Can also just get from code

$$\sum_{i=1}^{n}\left(\sum_{j=i}^{n}\left(\sum_{k=i}^{j}1\right)\right)$$

- Let's solve it by hand to practice with sums

Simplify the sum

$$\sum_{i=1}^{n}\left(\sum_{j=i}^{n}\left(\sum_{k=i}^{j}1\right)\right)$$

- When it gets down to "just Algebra", Maple is our friend

Help from Maple, part 1

Simplifying the last step of the monster sum

```
> simplify((n^2+3*n+2)/2*n
  -(n+3/2)*n*(n+1)/2+1/2*n*(n+1)*(2*n+1)/6);
```

$$\frac{1}{6}n^3+\frac{1}{2}n^2+\frac{1}{3}n$$

```
> factor(%);
```

$$\frac{1}{6}(n+2)\,n\,(n+1)$$

Help from Maple, part 2

Letting Maple do the whole thing for us:

```
sum(sum(sum(1, k=i..j), j=i..n), i=1..n);
```

$$\frac{1}{2}(n+1)n^2+2(n+1)n+\frac{1}{3}n+\frac{5}{6}-\frac{1}{2}n(n+1)^2-(n+1)^2$$
$$+\frac{1}{6}(n+1)^3-\frac{1}{2}n^2$$

```
> factor(simplify(%));
```

$$\frac{1}{6}(n+2)n(n+1)$$

We get same answer if we sum from 0 to n-1, instead of 1 to n

```
factor(simplify(sum(sum(sum(1,k=i..j), j=i..n),
i=1..n)));
```

$$\frac{n(n+2)(n+1)}{6}$$

```
factor(simplify(sum(sum(sum(1,k=i..j),j=i..n-1),
i=0..n-1)));
```

$$\frac{n(n+2)(n+1)}{6}$$

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

Fun tangent

Observe that $\frac{n(n+1)(n+2)}{6} = \binom{n+2}{3}$,

from basic counting/probability

- The textbook makes use of this in a curious way to find the sum more easily. Fun, but not required for class.

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 $f(n) \le cg(n)$ 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))

Can we do even better?

Tune in next time for the exciting conclusion!

http://www.etsu.edu/math/gardner/batman