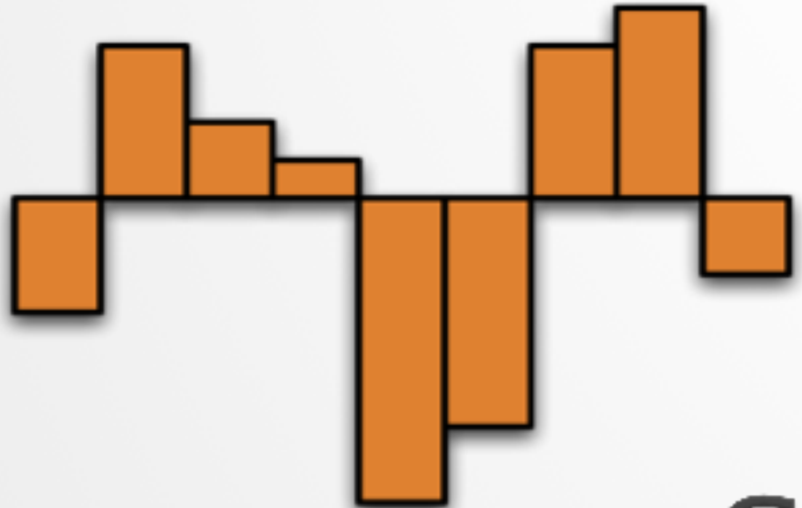
Q1–3, 4a



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

Announcements

- Homework 1 due tonight
- WarmUpAndStretching due after next class
- Reading for Day 4: Why Math?

Agenda and goals

- Asymptotic notation definitions
- Analyze algorithms for a sample problem, Maximum Contiguous Subsequence Sum (MCSS)

- After today, you'll be able to
  - explain the meaning of big-Oh, big-Omega (Ω), and big-Theta (θ)
  - apply the definition of big-Oh to asymptotically analyze functions, and running time of algorithms

Asymptotics: The "Big" Three

Big-Oh
Big-Omega
Big-Theta

Big-Oh, Big-Omega, Big-Theta

O( ) Ω( ) Θ( )

- f(n) is O(g(n)) if there exist c, n_0 such that:
  $$f(n) \leq cg(n) \text{ for all } n \geq n_0$$
  - So big-Oh (O) gives an upper bound

- f(n) is **Ω**(g(n)) if there exist c, n_0 such that:
  $$f(n) \geq cg(n) \text{ for all } n \geq n_0$$
  - So big-omega (Ω) gives a lower bound

- f(n) is Θ(g(n)) if it is both O(g(n)) and **Ω**(g(n))
  Or equivalently:
- f(n) is Θ(g(n)) if there exist c_1, c_2, n_0 such that:
  $$c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0$$
  - So big-theta (θ) gives a tight bound

Uses of O, Ω, Θ

- By definition, applied to *functions.*

  “$f(n) = n^2/2 + n/2 - 1$ is $\Theta(n^2)$”

- Can also be applied to an *algorithm*, referencing its **running time**: e.g., when f(n) describes the number of executions of the most-executed line of code.

  “selection sort is $\Theta(n^2)$”

- Finally, can be applied to a *problem*, referencing its **complexity:** the running time of the best algorithm that solves it.

  “The sorting problem is $O(n^2)$”

Big-Oh Style

- Give tightest bound you can
  - Saying $3n+2$ is $O(n^3)$ is true, but not as useful as saying it's $O(n)$
  - On a test, we'll ask for Θ to be clear.
- Simplify:
  - You could also say: $3n+2$ is $O(5n-3\log(n) + 17)$
  - And it would be technically correct…
  - It would also be poor taste … and your grade will reflect that.

Q6

Efficiency in context

- There are times when one might choose a higher-order algorithm over a lower-order one.

- Brainstorm some ideas to share with the class

C.A.R. Hoare, inventor of quicksort, wrote:
Premature optimization is the root of all evil.

Thoughts on Teaming

Next week's programming assignment is with a partner

Two Key Rules

- No prima donnas
  - Working way ahead, finishing on your own, or changing the team's work without discussion:
    - harms the education of your teammates
- No laggards
  - Coasting by on your team's work:
    - harms your education
- Both extremes
    - are selfish
    - may result in a failing grade for you on the project

Grading of Team Projects

- We'll assign an overall grade to the project
- Grades of individuals will be adjusted up or down based on team members' assessments

- At the end of the project each of you will:
  - Rate each member of the team, including yourself
  - Write a short Performance Evaluation of each team member with evidence that backs up the rating
    - Positives
    - Key negatives

Ratings

Excellent—Consistently did what he/she was supposed to do, very well prepared and cooperative, actively helped teammate to carry fair share of the load

Very good—Consistently did what he/she was supposed to do, very well prepared and cooperative

Satisfactory—Usually did what he/she was supposed to do, acceptably prepared and cooperative

Ordinary—Often did what he/she was supposed to do, minimally prepared and cooperative

Marginal—Sometimes failed to show up or complete tasks, rarely prepared

Deficient—Often failed to show up or complete tasks, rarely prepared

Unsatisfactory—Consistently failed to show up or complete tasks, unprepared

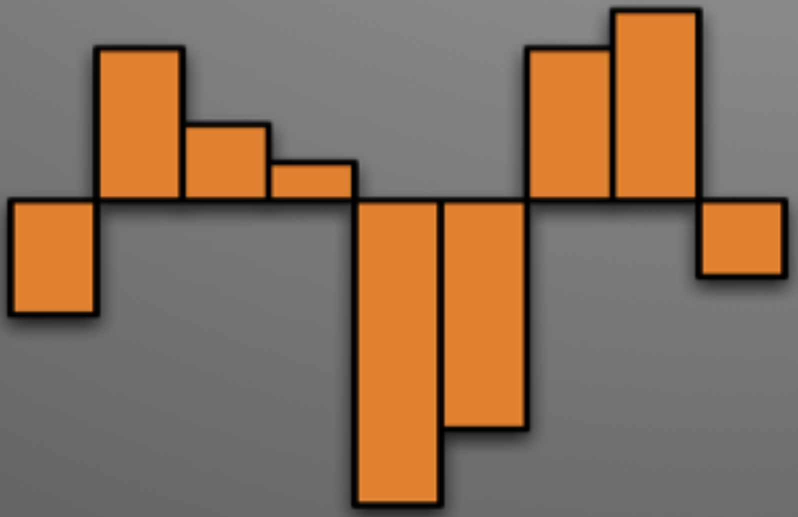
Superficial—Practically no participation

No show—No participation at all

Maximum Contiguous Subsequence Sum

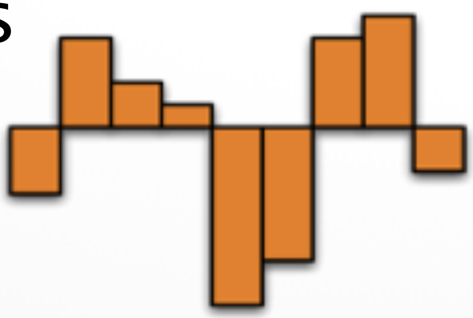
A deceptively deep problem with a surprising solution.

{−3, 4, 2, 1, −8, −6, 4, 5, −2}



A Nice Algorithm Analysis Example

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



- Why study?
- Positives and negatives make it interesting. Consider:
  - What if all the numbers were positive?
  - What if they all were negative?
  - What if we left out “contiguous”?
- Analysis of obvious solution is neat
- We can make it more efficient later.

Q7-10

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.

1-based indexing. But we'll use 0-based indexing

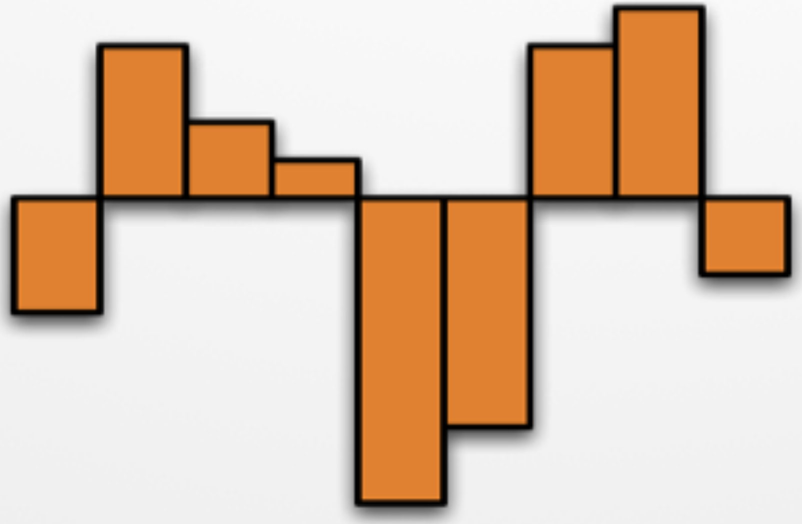
- Quiz questions:
  - In {-2, **11, -4, 13**, -5, 2}, $S_{1,3}$ = ?
  - In {1, -3, 4, -2, -1, 6}, what is MCSS?
  - If every element is negative, what's the MCSS?

Write a simple correct algorithm now

Q11

- 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?

Q12

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 ];
```

Q13

Solution

```
//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 telescopes.

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))

Can we do even better?

Tune in next time for the exciting conclusion!