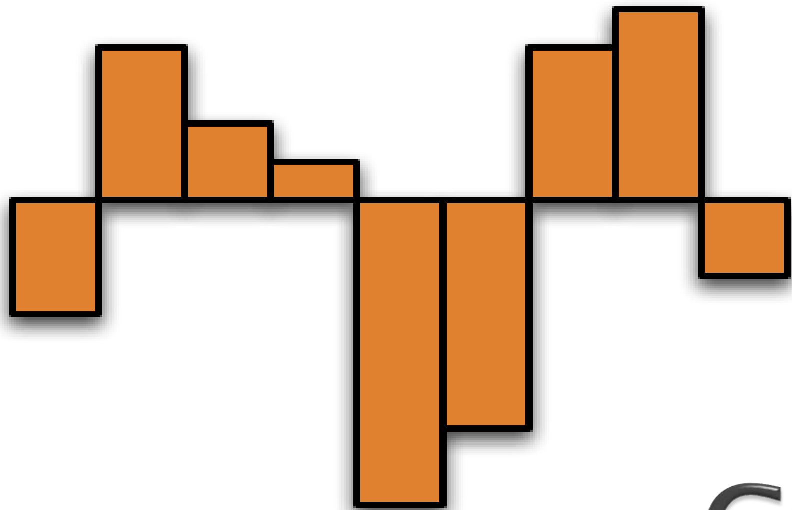
Q0



CSSE 230 Day 4

Maximum Contiguous Subsequence Sum

After today's class you will be able to:

provide an example where an insightful algorithm can be much more efficient than a naive one.

Q1

Recap: MCSS

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.

- In {−2, **11, −4, 13**, −5, 2}, MCSS is $S_{2,4} = ?$
- In {1, −3, 4, −2, −1, 6}, what is MCSS?

Recap: Eliminate the most obvious inefficiency, get $\Theta(N^2)$

```
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;
        }
    }
}
```

MCSS is $O(n^2)$

- Is MCSS $\theta(n^2)$?
  - 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

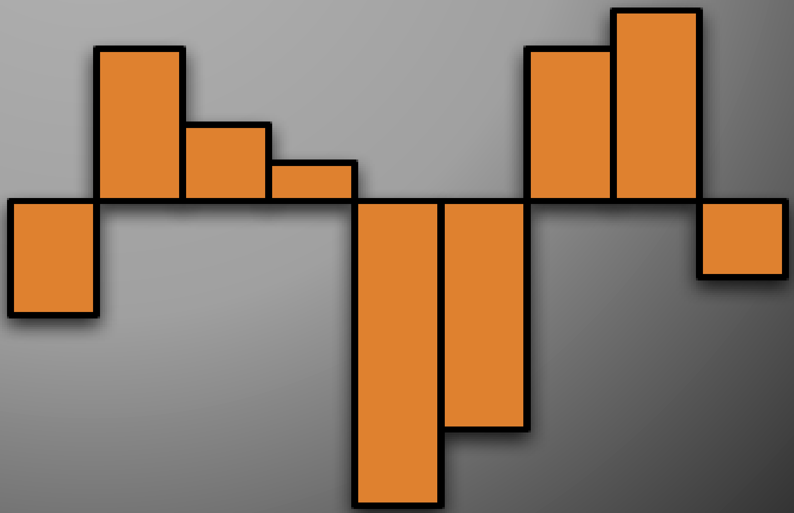
f(n) is Ω(g(n)) if $f(n) \geq cg(n)$ for all $n \geq n_0$
- So Ω gives a lower bound

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))

Maximum Contiguous Subsequence Sum

A linear algorithm.

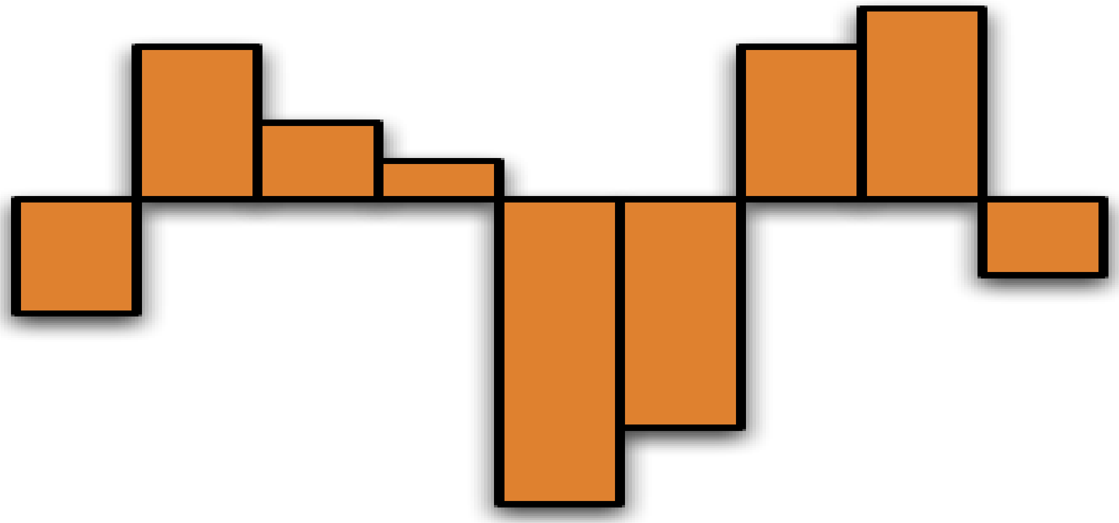
{−3, 4, 2, 1, −8, −6, 4, 5, −2}



Q2

Observations?

- Consider {−3, 4, 2, 1, −8, −6, 4, 5, −2}



- Any subsequences you can safely ignore?
  - Discuss with another student (2 minutes)

Q3

Observation 1

- We noted that a max-sum sequence $A_{i,j}$ cannot begin with a negative number.
- Generalizing this, it cannot begin with a prefix $A_{i,k}$ with $k<j$ whose sum is negative.
  - **Proof by contradiction.** Suppose that $A_{i,j}$ is a max-sum sequence and that $S_{i,k}$ is negative. In that case, a larger max-sum sequence can be created by removing $A_{i,k}$ However, this violates our assumption that $A_{i,j}$ is the largest max-sum sequence.

Observation 2

- All contiguous subsequences that border the maximum contiguous subsequence must have negative or zero sums.
  - **Proof by contradiction.** Consider a contiguous subsequence that borders a maximum contiguous subsequence. Suppose it has a positive sum. We can then create a larger max-sum sequence by combining both sequences. This contradicts our assumption of having found a max-sum sequence.

Q6

Observation 3

- No max-sum sequence can start from inside a subsequences that has a negative sum and extend beyond it.
- In other words, if we find that $S_{i,j}$ is negative, we can skip all sums that begin with any of A_i, A_{i+1}, …, A_j.
- We can “skip i ahead” to be j+1.

Q4-5

Observation 3

For any i, let $j \geq i$ be the smallest number such that $S_{i,j} < 0$.

Then for any p and q such that $i \leq p \leq j$ and $p \leq q$:

- either $A_{p,q}$ is not a MCS, or
- $S_{p,q}$ is less than or equal to a sum already seen (i.e., one with subscripts less than i and j respectively).

Proof of Observation 3

Proof: Note that $S_{i,q} = S_{i,p-1} + S_{p,q}$. By assumption, $S_{i,p-1} \geq 0$, since $p-1 < j$, and $S_{i,p-1} \geq 0$ implies $S_{i,q} \geq S_{p,q}$. Consider cases:

- Suppose $q > j$, then $A_{i,j}$ is part of $A_{i,q}$ and (by Obs. 1) $A_{i,q}$ is not a MCS. But $S_{i,q} \geq S_{p,q}$, so $A_{p,q}$ is not a MCS either.
- Suppose $q \leq j$, then $S_{i,q}$ is a "sum already seen". Since $S_{p,q} \leq S_{i,q}$ the claim holds.

New, improved code!

Q7

```
public static Result mcssLinear(int[] seq) {
    Result result = new Result();
    result.sum = 0;
    int thisSum = 0;

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

        if (thisSum > result.sum) {
            result.sum = thisSum;
            result.startIndex = i;
            result.endIndex = j;
        } else if (thisSum < 0) {
            // advances start to where end
            // will be on NEXT iteration
            i = j + 1;
            thisSum = 0;
        }
    }
    return result;
}
```

$S_{i,j}$ is negative. So, skip ahead per Observation 3

Running time is is O (?)
How do we know?

Time Trials!

- From SVN, checkout `MCSSRaces`
- Study code in `MCSS.main()`
- For each algorithm, how large a sequence can you process on your machine in less than 1 second?

Q9-10

MCSS Conclusions

- The first algorithm we think of may be a lot worse than the best one for a problem
- Sometimes we need clever ideas to improve it
- Showing that the faster code is correct can require some serious thinking
- Programming is more about careful consideration than fast typing!

What have we shown?

- MCSS is $O(n)$!
- Is MCSS $\Omega(n)$ and thus $\theta(n)$?
  - Yes, intuitively: we must at least examine all n elements

- Big picture:
  - Showing that a problem is $O(g(n))$ is easy – just analyze the algorithm.
  - Showing that a problem is $\Omega(g(n))$ **in general** is much tougher. How do you prove that it is impossible to solve a problem more quickly than you already can?

Interlude

- If GM had kept up with technology like the computer industry has, we would all be driving $25 cars that got 1000 miles to the gallon.
  – Bill Gates

- If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost $100, get a million miles per gallon, and explode once a year, killing everyone inside.
  – Robert X. Cringely

Stacks and Queues

A preview of Abstract Data Types
and Java Collections

This week's major program

Stacks and Queues Part 1

An exercise in implementing your own growable circular Queue:

1. Grow it as needed (like day 1 exercise)
2. Wrap-around the array indices for more efficient dequeuing

Discuss Stacks as a warmup (push, pop), then ideas for Queues (enqueue, dequeue)

Analyze implementation choices for Queues – much more interesting than stacks!

Q8

Stacks Part 2: Evaluator

An exercise in writing cool algorithms that evaluate mathematical expressions:

Infix: 6 + 7 * 8

Postfix: 6 7 8 * +

Both using **stacks**.

Meet your partner

- Plan when you'll be working
- Review the pair programming video as needed
- Check out the code and read the specification together