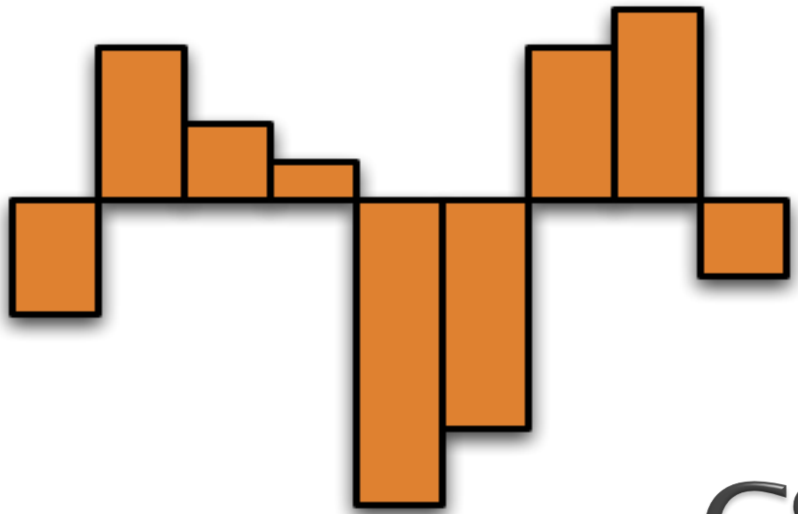
Q0, 1



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.

Announcements

- Sit with your StacksAndQueues partner now
- Day 2 quizzes returned
- Why Math?

Andrew Hettinger ▸ **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 sort first to do binary search?

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.

Reminder: we use 0-based indexing.

Recap: Eliminate the most obvious inefficiency, get Θ(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;
        }
    }
}
```

- Exhaustive search: find every $S_{i,j}$

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?
    - If so, it can't use exhaustive search. (Why?)

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

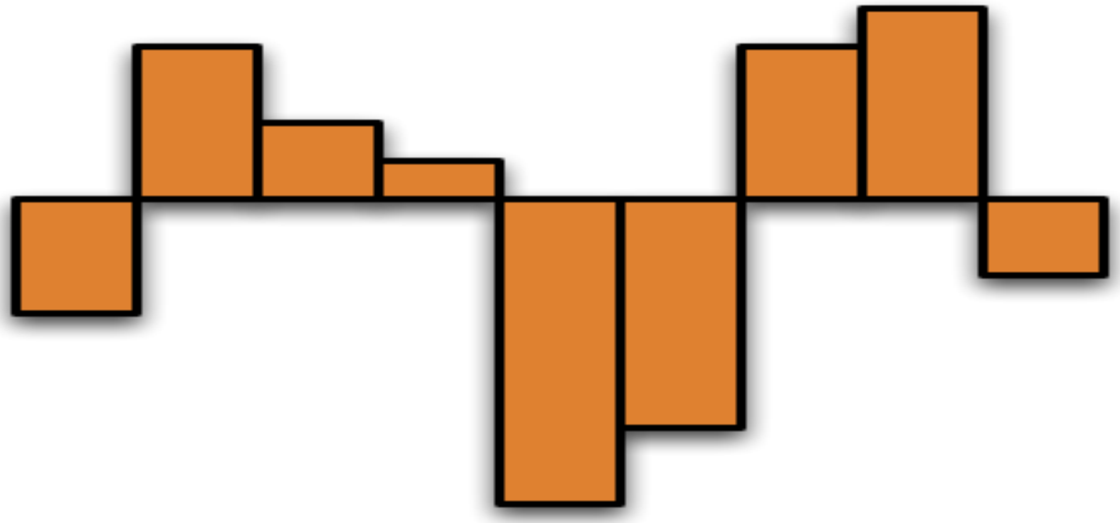
f(n) is Ω(g(n)) if $f(n) \geq cg(n)$ for all $n \geq n_0$
- So Ω gives a lower bound

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))

Q2

Observations?

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



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

Hidden

```
/**
* Linear-time maximum contiguous subsequence sum algorithm.
* seqStart and seqEnd represent the actual best sequence.
*/
public static int maxSubSum3( int [ ] a )  {
    int maxSum = 0;
    int thisSum = 0;

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

        if( thisSum > maxSum ) {
            maxSum = thisSum;
            seqStart = i;
            seqEnd   = j;
        } else if( thisSum < 0 ) {
            i = j + 1;
            thisSum = 0;
        }
    }

    return maxSum;
}
```

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.

Q4

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 an MCSS. 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.

Observation 3

- Imagine we are growing subsequences from a fixed left index i. That is, we compute the sums $S_{i,j}$ for increasing j.

- Claim: For such $S_{i,j}$ that "just became negative" (for the first time, with the inclusion of the j^{th} term), any subsequence starting in between $i + 1$ and j cannot be a MaxCSS (unless its sum equals an already-found MaxCSS)!

- In other words, as soon as we find that $S_{i,j}$ is negative, we can skip all sums that begin with any of $A_{i+1}, \ldots, A_j$.

- We can "skip i ahead" to be $j + 1$.

Proof of Observation 3

- Proof by Contradiction. Suppose there is such a MaxCSS, namely $S_{p,q}$.

i $S_{i,j}$ just became negative! j

- Key point. What must be true of the following sums?

$S_{i,p-1} \geq 0$ | $S_{p,j} < 0$

Case 1. $q > j$

p MaxCSS q

Starts with a negative prefix. Violates Obs. 1!

Case 2. $q \leq j$

p MaxCSS q

Borders a subsequence with nonnegative sum.
Violates Obs. 2, or there is a previous MaxCSS with the same sum.

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.

Proof of Observation 3

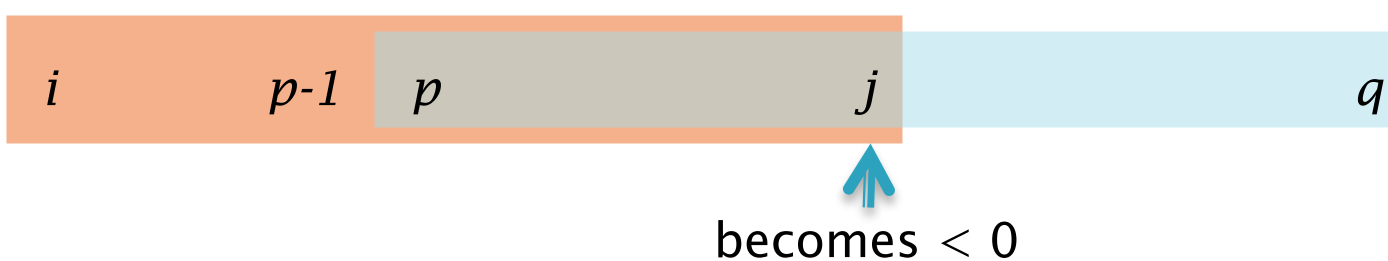
Proof: Note that $S_{i,q} = S_{i,p-1} + S_{p,q}$. By assumption, $S_{i,p-1} \geq 0$, since $p-1 < j$, and $S_{i,p-1} \geq 0$ implies $S_{i,q} \geq S_{p,q}$. Consider cases:

- 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.

Proof of Observation 3

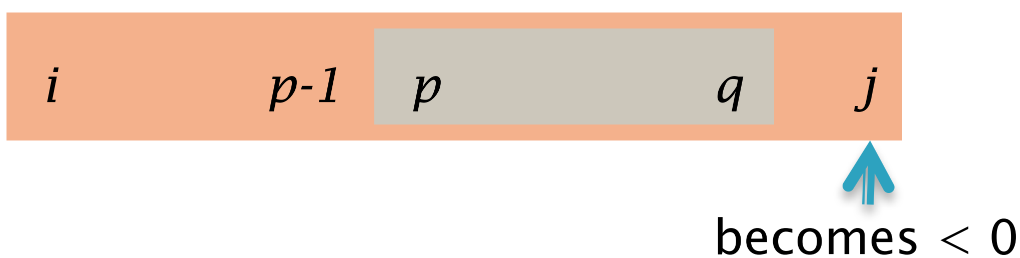
Proof: Note that $S_{i,q} = S_{i,p-1} + S_{p,q}$. By assumption, $S_{i,p-1} \geq 0$, since $p-1 < j$, and $S_{i,p-1} \geq 0$ implies $S_{i,q} \geq S_{p,q}$. Consider cases:

- 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.



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 $q \leq j$, then $S_{i,q}$ is a "sum already seen". Since $S_{p,q} \leq S_{i,q}$ the claim holds.

New, improved code!

Q5, Q6

```
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?

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

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?

Q10-11

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!

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

Q9, 7-8

Stacks and Queues assignment

Intro: Ideas for how to implement stacks and queues using arrays and linked lists

How to write your own growable circular queue:

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

Stacks and Queues implementation

Analyze implementation choices for Queues – much more interesting than stacks! (See HW)

Application: An exercise in writing cool algorithms that evaluate mathematical expressions:

Evaluate Postfix: 6 7 8 * +
(62. How?)
Convert Infix to Postfix: 6 + 7 * 8
(6 7 8 * +   You'll figure out how)

Both using **stacks**.
Read assignment for hints on *how*.

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