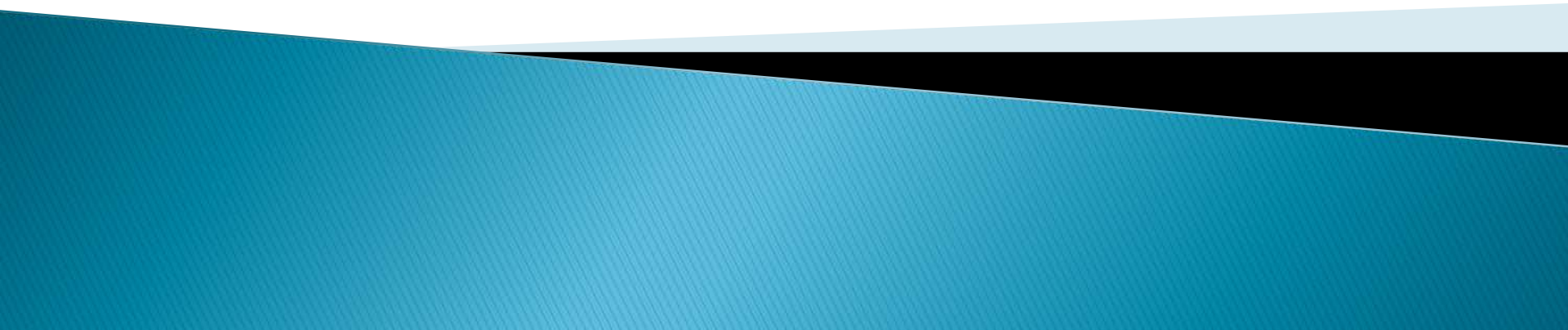


# CSSE 230 Day 6

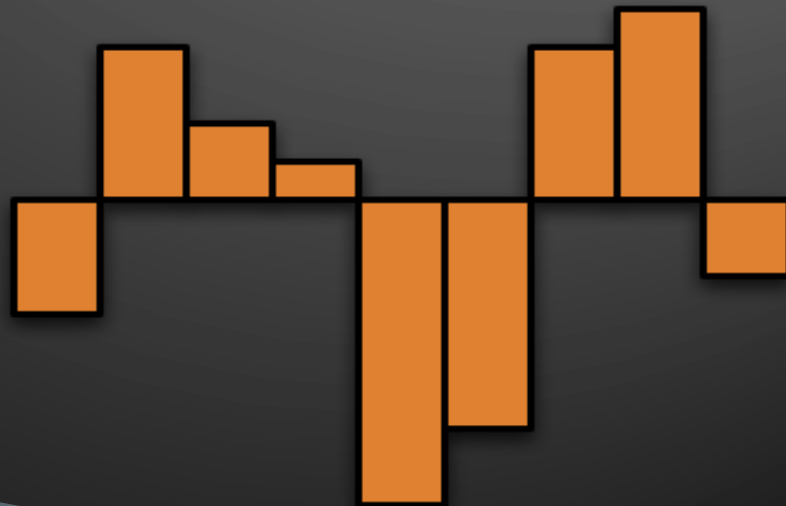
Maximum Contiguous Subsequence Sum



# Maximum Contiguous Subsequence Sum

»» A linear algorithm.

$\{-3, 4, 2, 1, -8, -6, 4, 5, -2\}$



# Recap: MCSS

*Problem definition:* Given a non-empty sequence of  $n$  (possibly negative) integers  $A_1, A_2, \dots, A_n$ , 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?

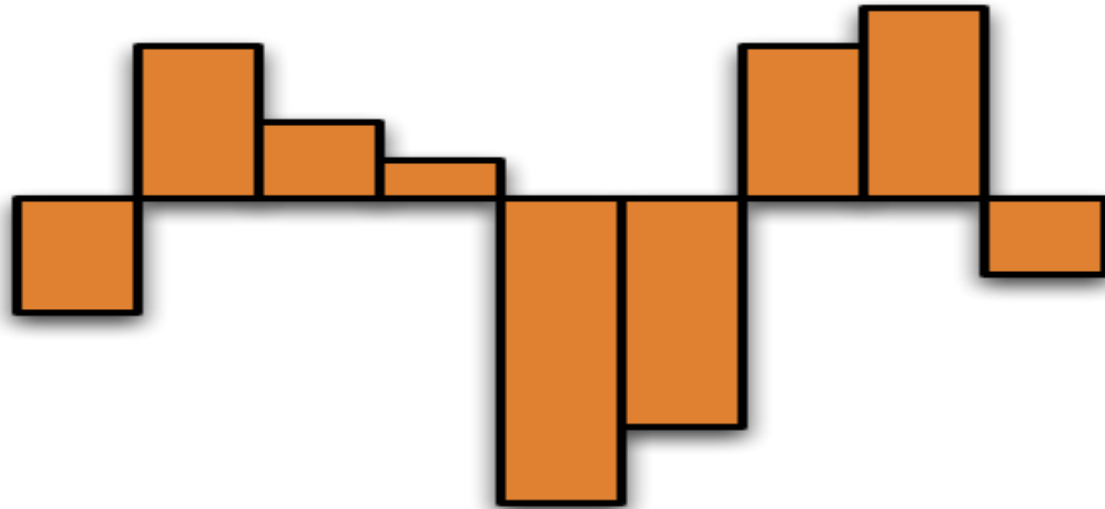
# Recap: Eliminate the most obvious inefficiency, get $\Theta(N^2)$

```
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;  
        }  
    }  
}
```

We can do  
even better  
than this!

# Observations?

- ▶ Consider  $\{-3, 4, 2, 1, -8, -6, 4, 5, -2\}$



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

# 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: If  $S_{i,k}$  is negative, then  $S_{k+1,j} > S_{i,j}$ , so  $A_{i,j}$  would not be a sequence that produces the maximum sum.

# Observation 2

- ▶ All contiguous subsequences that border the maximum contiguous subsequence must have negative (or zero) sums.
  - **Proof:** If one of them had a positive sum, we could simply append (or “prepend”) it to get a sum that is larger than the maximum. Impossible!

# 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.

# So What!?

- ▶ If we find that  $S_{i,j}$  is negative, we can skip all sums that begin with any of  $A_i, A_{i+1}, \dots, A_j$ .
- ▶ There is no new MCS that starts anywhere between  $A_i$  and  $A_j$ .
- ▶ So we can “skip  $i$  ahead” to be  $j+1$ .

## Observation 3 again:

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).

# New, improved code!

```
public static Result mcsslLinear(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  $\Theta(?)$   
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?

# 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!
- 

# Pair programming

»» A cheezy, helpful video

[http://www.youtube.com/watch?v=rG\\_U12uqRhE&feature=plcp](http://www.youtube.com/watch?v=rG_U12uqRhE&feature=plcp)

# Finite State Machines

» Also known as  
Deterministic Finite Automata

# A Finite State Machine (FSM)

- ▶ A finite set of **states**,
  - One is the **start state**
  - Some are **final**, a.k.a **accepting**, states
- ▶ A finite **alphabet** (input symbols)
- ▶ A **transition function**
- ▶ How it works:
  - Begin in start state
  - Read an input symbol
  - Go to the next state according to transition function
  - More input?
    - Yes, then repeat
    - No, then if in accept state, return true, else return false.



# Example

- ▶ Draw a FSM to determine whether a lowercase sequence of characters contains each of the 5 regular vowels once in order
  - Example: **facetious**
- ▶ In some versions of FSMs, each transition generates output.

# Another FSM Example



A

B

C

D

# Draw state diagrams for these FSMs

- ▶ Indicate the Start State and final (accepting) states
- ▶ FSM1:
  - Input alphabet {0, 1}
  - Accepts (ends in an accepting state) all input strings that do NOT contain **010** as a substring
- ▶ FSM2: (only if you get the first one done quickly)
  - Input alphabet {0, 1}
  - Accepts (ends in an accepting state) all input strings that are binary representations of numbers that are divisible by 3

Hints: Use 4 states, a start state plus 1 state each for  $x\%3==0$ ,  $x\%3==1$ , and  $x\%3==2$ .

What does the arrival of a 0 do to the current value? (doubles it) What about a 1?

x	binary	x	binary
0	0	7	111
1	1	8	1000
2	10	9	1001
3	11	10	1010
4	100	11	1011
5	101	12	1100
6	110	13	1101

# Colorize

- ▶ A pair programming assignment.
- ▶ Due (along with Hardy, Part 2) on Class Day 10.

# Colorize program assignment

- ▶ Input: legal Java source code
- ▶ Output: colored HTML
  - Keywords in blue, strings in red, comments in green, everything else in black
  - Layout just like original Java input file

```
// Opening comment. Note that a "string" is ignored here.
class /* Bad name */ Stupid {
    int x;
    String t = "A string with a /* in it";
    String p = "A string with a \" in it";
    boolean b = t.compareTo(p) < 0;

    public static void main(String [] args) {
        System.out.println("" + t + " " + p);
        System.out.println("Can you think of other interesting cases that your ")
    }
    /* Notice that comments /* do not "nest" in Java // */
}
```

We can use an FSM for this!

# More About Colorize

» FSM representations

# Possible Representations of the Finite State Machine

Diagrams  
on the  
whiteboard

- ▶ 2-Dimensional array:
  - Rows indexed by state, Columns by input character.
  - Each array entry is a pair object (as in DS Section 3.7):
    - [next state, what to print]
- ▶ Monolithic controller with nested switch statements
- ▶ The first choice may be more efficient and have shorter code
- ▶ The second choice is probably easier to write and modify
  - Can be made more modular by having a method for each state