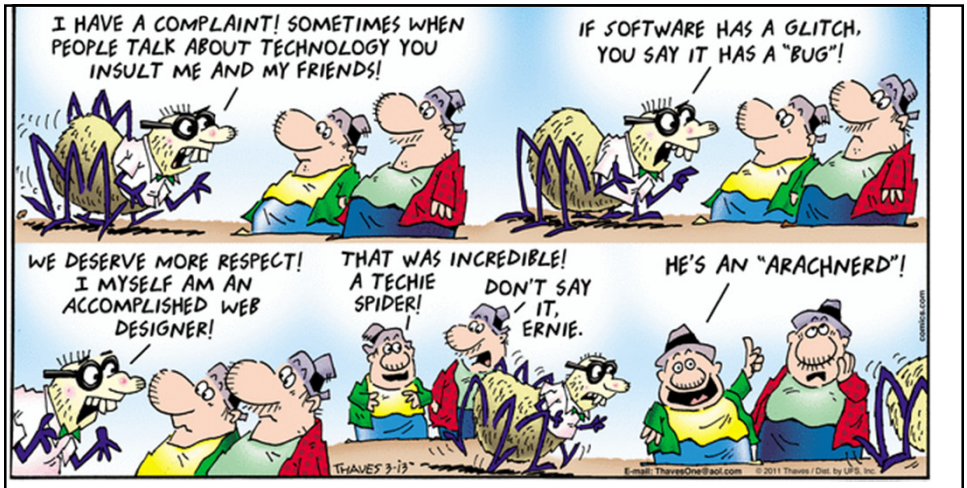


CSSE 230 Day 5

Maximum Contiguous Subsequence Sum

Questions?

Program Grading

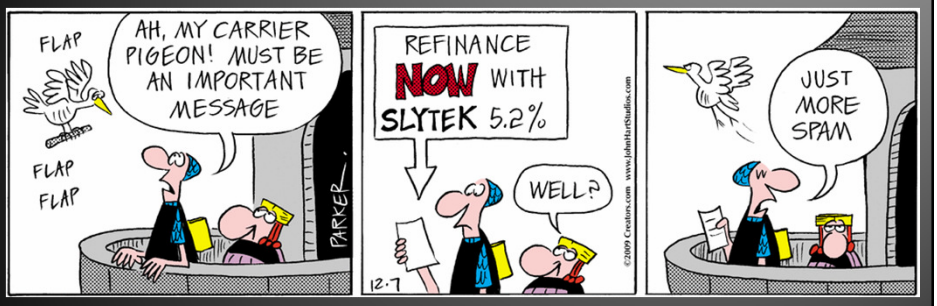
- Correctness usually graded using JUnit tests
  - Exception: when we ask you to add your own tests
- Style
  - No warnings remaining (per our preference file)
  - Reasonable documentation
  - Explanatory variable and method names
  - You should format using Ctrl-Shift-F in Eclipse
- Efficiency
  - Usually reasonable efficiency will suffice
    - (e.g., no apparently infinite loops)
  - Occasionally (like next week) we might give a minimum big-Oh efficiency for you to achieve

Between two implementations with the same big-Oh efficiency, favor the more concise solution, unless you have data showing that the difference matters.

Finish the Comparator Example

Add an anonymous Comparator to main()

Diagnostic Quiz Review



Some questions that many students missed

Expression questions

- Give a very simple Java expression that is equivalent to:

```
!(x && !x)
```

BTW:Never write something like

```
if (a.isVisible() == true)
```

- What are the values of each of the following expressions, if x==5 and y ==7 ?

```
x + ' ' + y
x + " " + y
x + y + " "
```

Simple big-Oh questions

- What is the worst-case Big-Oh running time of an unsuccessful sequential search of an unordered array that contains N elements?
- What is the worst-case Big-Oh running time of an unsuccessful binary search of an array that contains N
- What is the Big-Oh running time of merge sort of an array that contains N elements?

Q1-3

Method Selection Overloading vs. Overriding

- In Eclipse, open:
  examples.StaticParmsDemo
- From the **DiagQuizReview** project
- This is based on Figure 4.45, page 166 of Weiss.
- Section 4.9 begins:
  - "A common myth is that all methods and all parameters are bound at runtime. This is not true."
    - Methods that are static, final, or private.

Note that all of the code from the Weiss book is available on the course web site. You can run it, modify it, and experiment.

Interlude

- Computer Science is no more about computers than astronomy is about telescopes.

Donald Knuth

Interlude

- Computer Science is no more about computers than astronomy is about _____.

Donald Knuth

Q4

Aliasing

- How many objects are created in this code?

```
MyNumber a = new MyNumber();
a.setNum(5);
MyNumber b = new MyNumber();
b.setNum(6);
MyNumber c = a;
System.out.println(c);
```

- What is “aliasing”?

Q5

Default Constructors

- What does Java do if no constructor is declared for a class?
  - How can we instantiate the class?
  - What values do the fields get?

```
class Jambalaya {
  int beans;
  double rice;
  Insect crayfish;

  public String toString() {
    return beans + " " + rice + " " + crayfish;
  }
}
```

Parameter Passing

- this code is available In Eclipse, open **examples.WhatIsX**

```
public static void main(String[] args) {
    int x = 0;
    f(x);
    System.out.println(x);
}

private static void f(int x) {
    /*
     * TODO: Without adding printing, can you change the body of
     * this method to get this program to print:
     *    3?
     *    3.5?
     *    Anything else?
     */
```

Q6-7

More Big-Oh Practice

```
for (int i = 0; i < n; i++)
   for (int j = 0; j < i; j++)
     sum++;
```

34% of students answered N log N.
Where could the log come from?

```
for (int i = 0; i < n; i++)
   for (int j = 0; j < n * n; j++)
     for (int k = 0; k < j; k++)
       sum++;
```

```
for (int i = 1; i < n; i = i * 2)
   sum++;
```

One more distinction

- `throw` versus `throws`
  - Part of exception handling
  - Signal an error with: `throw new ExceptionType()`
  - Abdicate responsibility with:

```
void myMethod() throws ExceptionType {
  …
}
```

Finite State Machines

- Also known as Deterministic Finite Automata

A Finite State Machine (FSM)

Q8-9

- 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.

Q10

Example

- Draw a FSM to determine whether a lowercase sequence of characters contains each of the 5 regular vowels once in order
  - Example: f**a**c**e**t**i**o**u**s
- 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, Coming Soon

- A pair programming assignment.
- Due (along with Hardy, Part 2) on Class Day 12.

Colorize program assignment

- Input: legal Java source code
- Output: colorized 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!

Next time

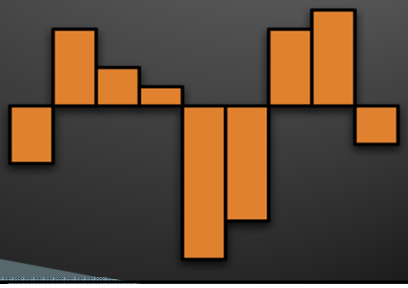
- Maximum Contiguous Subsequence Sum problem from Weiss Chapter 5.
- Perhaps we will start it today.

Q1

Maximum Contiguous Subsequence Sum

- A deceptively deep problem with a surprising solution.

{–3, 4, 2, 1, –8, –6, 4, 5, –2}

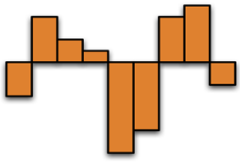


Why do we look at this problem?

- 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.

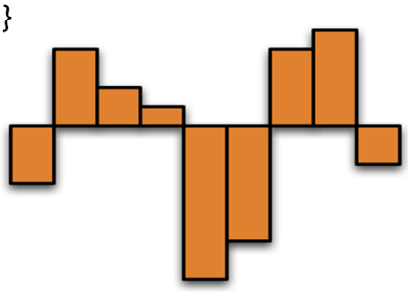
- In {-2, **11, -4, 13**, -5, 2}, $S_{2,4}$ = ?
- In {1, -3, 4, -2, -1, 6}, what is MCSS?
- If every element is negative, what's the MCSS?

1-based indexing

Q5

A quick-and-dirty algorithm

- Design one right now.
  - Efficiency doesn't matter.
  - It has to be easy to understand.
  - 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 $1 \le i \le k \le j \le n$?

```
//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.

Three ways to find the sum

- By hand
- Using Maple
- Magic! (not really, but a preview of Disco)