CSCI 136:
Data Structures
and
Advanced Programming

Lecture 10

Asymptotic analysis, part 1

Instructor: Dan Barowy

Williams

Announcements

- PRE-LAB: Partner preference form
- Quiz wording (ambiguous)

Quiz

Outline

Rec. solution to coin problem

Asymptotic analysis

Last time

Prove: n cents can be obtained by using only 3-cent and 8-cent coins, for all n ≥ 15.

Proof sketch

a = 15; **P(15)**: is 5 x 3 cents. **True**.

P(k) ⇒ P(k+1) **True**.

Assume **P(k)** is **true**.

Case 1: **P(k)** has a at least one 8-cent coin.
Then we can produce the value k+1 by replacing an 8-cent coin with 3 x 3 cent coins.

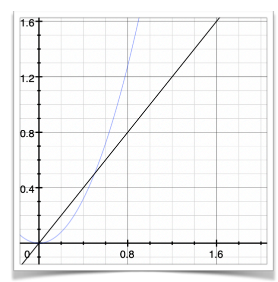
Case 2: **P(k)** has no 8-cent coin.
Then we can produce the value k+1 by replacing 5 x 3 cents coins with 2 x 8 cent coins. This is OK because k > 15.

Therefore we can find change for all n ≥ 15. **True**.

Activity

Now write a program that gives you the correct change for all n ≥ 15.

Asymptotic analysis



How do we know if an algorithm is faster than another?

Why can't we just measure "wall time"?

Why can't we just measure "wall time"?

- Other things are happening at the same time
- Total running time usually varies by input
- Different computers may produce different results!

Let's just count instructions, then

- What do we count?
  - Count all computational steps?
  - What is a "step"?
  - What about steps inside loops?

Stepping back…

- How accurate do we need to be?
  - If one algorithm takes 64 steps and another 128 steps, do we need to know the precise number?

We what do

Instead of precisely counting steps, we usually develop an approximation of a program's time or space complexity.

This approximation ignores tiny details and focuses on the big picture: how do time or space requirements grow as a function of the size of the input?

Cases: best, average, worst

We can do this analysis for the best, average, and worst cases. We often focus on the worst case.

Example

```
// Pre: array length n > 0
public static int findPosOfMax(int[] arr) {
        int maxPos = 0
        for(int i = 1; i < arr.length; i++)
                if (arr[maxPos] < arr[i]) maxPos = i;
        return maxPos;
}
```

- Can we count steps exactly?
  - `if` complicates counting
- Idea: **overcount**: assume `if` block always runs
  - in the worst case, it does
- Overcounting gives **upper bound** on run time
- Can also **undercount** for **lower bound**

Overcounting Example

```
// Pre: array length n > 0
public static int findPosOfMax(int[] arr) {
        int maxPos = 0
        for(int i = 1; i < arr.length; i++)
                if (arr[maxPos] < arr[i]) {
                    maxPos = i;
                }
        return maxPos;
}
```

// line 1 cost: c_1
// line 2 cost: nc_2
// line 3 cost: nc_3
// line 4 cost: nc_4
// line 6 cost: c_5

Total cost: $c_1 + nc_2 + nc_3 + nc_4 + c_5$

$= c_1 + n(c_2 + c_3 + c_4) + c_5$

$= n(c_2 + c_3 + c_4) + c_1 + c_5$

$\approx O(n)$

(as you shall see)

Recap & Next Class

Today we learned:

Intro to asymptotic analysis

Next class:

Big-O notation