Last Algorithms and touching on Chapter 3

Spring 2017 - Aaron S. Crandall, PhD

Today’s Outline

- Announcements
- Thing of the Day
- Last algorithms
  - Space complexity
- Touching on Chapter 3

Announcements

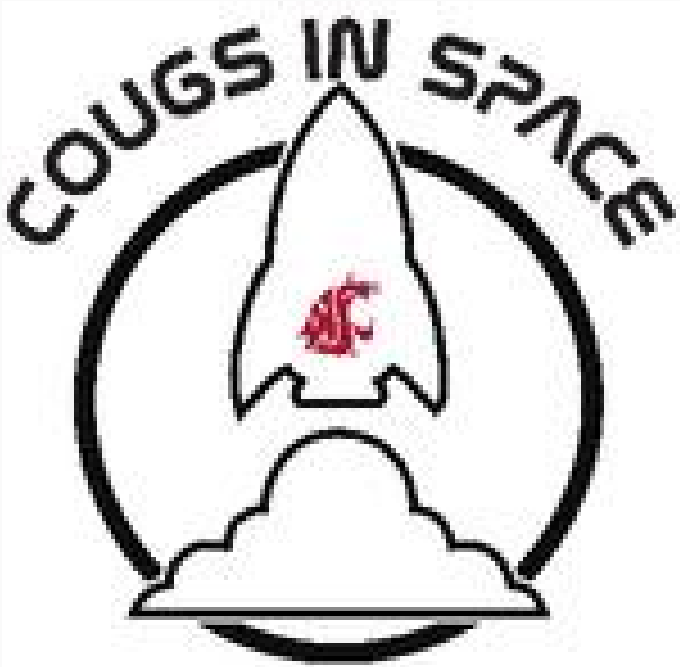
- Google’s schedule of events for next week is updating, but they’re going to have lots of opportunities to meet with people.
- Women’ tennis vs Play With a Cougar @ 4:00 PM today
- Women’s Soccer vs CSUN @ 7:00 PM PT today
- Football vs Boise State is Saturday @ 7:30pm. Parking shut down, etc
- Women’s tennis vs ITF Las Vegas - All day sunday

Cougs in Space: Recruiting Embedded Project Team for CougSat I

CougSat I needs more embedded engineers -

* Combination of code & hardware
* Starting with Arduino-based systems and RPi's
* Linux experience is valuable (more gained too!)
* Especially computer engineering, but CS is great

cougsinspace.slack.com



Thing of the Day: The $10,500 robot arm that can build its coworkers

- Franka Emika is a robot arm for $10,500 that’s incredibly versatile.
- In fact, their signature move is to show one building another one. https://qz.com/874689/the-10500-robot-arm-that-can-build-its-coworkers/

A quiz! Yes, it’s time.

bit.ly/2xRWKvT

Last class recap

- We discussed:
  - Using Git for the class and who that's architected
    - Yes, there's a few people who dropped me notes, we'll get your repos sorted out
  - Also talked about using Linux
  - Then the BigFive at high velocity
- Didn't get to:
  - We ran through more examples of Big-O

Now, back to some algorithm analysis review & new materials

What to analyze in an algorithm?

- Options include:
  - T_ave(N)
  - T_worst(N)
  - T_optimal(N)
- T_optimal(N) <= T_ave(N) <= T_worst(N)
- Do implementation details matter for algorithms analysis?
  - Copying big arrays vs. pass by reference example
  - TL;DR: No, implementation isn't about algorithm analysis
  - That said: it matters in the real world when you code

The IF statement rule

```
if( condition )
        S1
Else
        S2
```

* Use the larger of S1 or S2
* Yes, if you know the ratio of the two you could do a deeper analysis for a tighter bound, but the default is to just take the larger branch cost

The IF statement rule at work

```
void smartsort( int & vec ) {
        if( vec.size() <= 10 ) {
                insertionSort( vec ); // O(n^2)
        else:
                quickSort( vec );    // O(n log n)
}
```

Overall, how do we calculate the time complexity of this function?

What's the space complexity?

Is there a way to get a better average time estimate?

A different example (and why)

```
function sample( k ) {
  if k < 2
     return 0
  return 1 + sample(k/2)
}
```

- What is the time complexity of this algorithm?
- What is the space complexity? … hidden costs
- What if it was sample(k/3)?

When worst case analysis breaks down

- Big-O is often much bigger than average running time
- This can be an opportunity to empirically derive running behavior to update your O(N) calculations. For example:
  - Determining the ratio of S1:S2 in IF statements
  - Getting a distribution of orderliness in input data (pre-sorted or not)
  - This is not always possible since you're now working in distributions, so you get stuck with worst case analysis

What is space (or memory) complexity?

- Algorithms take both time and space to execute
- So far we've mostly only talked about time
- But we also need to understand space use
- In our general model, we aren't using a formal Turing machine model with the three tapes:
  - Read-only input, write-only output, read-write work memory
  - Dr. Dang will definitely be doing formal models, but we don't have to yet

Space complexity is based upon work memory size

- Size of memory used to do the work during the algorithm
  - Only includes memory consumed by function calls individually, but…!
  - Can be summed up if there's a recursive call stack
- Difference between memory complexity and RAM use!
  - RAM use is more than memory complexity since you need O(log n) bits per variable
  - We'll be working in memory complexity space, not RAM use space

Basic function space complexity

```
int sum(int x, int y, int z) {
  int r = x + y + z;
  return r;
}
```

- 3 units for the parameters (x, y, z), 1 for the local variable (r), and nothing changes
- Result: O(1)

Basic loop space complexity

```
int sum(int a[], int n) {
  int r = 0;
  for (int i = 0; i < n; ++i) {
    r += a[i];
  }
  return r;
}
```

- N units for a
- Plus space for n, r, and i
- But that’s O(N) + n + r + i
- Result is O(N)

Basic loop space complexity II

```
int sum(int & a[], int n) {
  int r = 0;
  for (int i = 0; i < n; ++i) {
    r += a[i];
  }
  return r;
}
```

- ? units for a
- Plus space for n, r, and i
- But that’s O(?) + n + r + i
- Result is O(?)

Recursive space complexity

```
function sample(k)
  if k < 2
     return 0
  return 1 + sample(k/2)
```

- Input needs O(1) per call, but how many calls?
- Result is?

One bound on space complexity

- Time complexity >= space complexity
- Why is this?
- Think about what it takes to read N elements

Once more with the magic of compilers

- What is tail recursion?
  - It's worth knowing, but we're not going to assume it exists in our world
- It turns a function call into a GOTO statement
  - What is a GOTO statement?
  - This means it jumps to the start of the next iteration without pushing on the stack!
  - Ever wonder how some languages can infinitely recurse?
- What does this do to memory complexity of our log(N) function?
- g++ -S -O0 vs. g++ -S -O2
  - -O0 -> no tail recursion optimizations
  - -O2 -> does tail recursion optimizations

Recursive space complexity

```
function sample(k)
  if k < 2
     return 0
  return 1 + sample(k)
```

- Input needs O(1) per call, but how many calls end up happening?
- If there's tail recursion how many calls?
- Result is?

Monday’s material: Chapter 3!

- So, if you haven’t finished Chapter 2 get on it.
- Today was a day of exclamation points!
  - For this, I’ll apologize, but I’m not sure I’m sorry.*
- On Monday we’ll do our single day on Chapter 3
  - Look at stacks, queues, lists from a computer science perspective

* I’m never sorry for a well placed Interrobang. Why should I be‽