Asymptotics

Correction

- Said A/B was basic operation last class
- $A/B = A \cap B^C$
- Could further expand $A \oplus B = (A \cup B) \cap (A \cap B)^C$

Big O notation

- Formal:

$$\forall f(x), g(x) > 0, f(x) = O(g(x)) \text{ iff}$$
$$\exists c_1, n_0 \in \mathbb{R} \; s.t. \; \forall n \geq n_0, f(n) \leq c_1 \cdot g(n)$$

- English: f(x) is O(g(x)) if **eventually** f(x) is always less than g(x)

Big Omega

$$\forall f(x), g(x) > 0, f(x) = \Omega(g(x)) \text{ iff}$$
$$\exists c_1, n_0 \in \mathbb{R} \; s.t. \; \forall n \geq n_0, f(n) \geq c_1 \cdot g(n)$$

Big Theta

$$\forall f(x), g(x) > 0, f(x) = \Theta(g(x)) \text{ iff}$$
$$\exists c_1, c_2, n_0 \in \mathbb{R} \; s.t. \; \forall n \geq n_0,$$
$$c_1 \cdot g(n)) \leq f(n) \leq c_2 \cdot g(n)$$

Summary

- Big O - Takes less than that much time
- Big Omega - Takes more than that much time
- Big Theta - Takes “exactly” that much time

Getting the formulas

- Only look at the highest order term
- Drop all constants
- If multiple variables, highest order for each variable
- $x^3y + xy + 2 + xy^3$

Variations

- Little o notation

$$\forall f(x), g(x) > 0, f(x) = o(g(x)) \text{ iff}$$
$$\forall c_1 \in \mathbb{R}\ \exists n_0 \in \mathbb{R}\ s.t.\ \forall n \geq n_0 f(n) \leq c_2 \cdot g(n)$$

- Much stronger claim than big O

- Must work for all constants, not just some

Proving little o

- $f(x) = x$ is $o(x^2)$

Fancier and rarer types

$$\forall f(x), g(x) > 0, f(x) = \tilde{O}(g(x)) \text{ iff}$$
$$\exists c_1, n_0 \in \mathbb{R} \; s.t. \; \forall n \geq n_0$$
$$f(n) \leq c_2 \cdot g(n) \cdot \log^k g(n)$$

Heirarchy

- 1 - Constant
- α(n) - Inverse Ackermann
- $\log^*(n)$ - Iterated logarithmic (log-star)
- log(n) - Logarithmic
- $\log^k(n)$ - Polylogarithmic
- n - Linear
- n log n - Linearithmic
- n^c - Polynomial
- 2^n - Exponential
- n! - Factorial

A grain of salt

- Theorists love having a bunch of these extra classes
- But in reality...
- Log^*(number of atoms in the entire universe) = 4
- Inverse Ackermann grows even slower than that

Examples

```
int arr1[n][m], arr2[m][n], arr3[n][n];
for(int i = 0; i < n; i++)
    for(int j = 0; j < n;  j++)
        for(int k = 0; k < m; k++)
            arr3[i][j] += arr1[i][k]*arr2[k][j]
```

```
temp = 0
for(int i = 0; i < n; i*=2)
    for(int j = 0; j < n; j++)
        temp++;
```

The importance of solving as a sequence

```
s = 0; c = n; p = log(n);
for (h = 1; h < p; h++) {
   c = c/2;
   for (j = 1; j < c; j++) {
       for (k = 1; k <= h; k++)
          s++;
   }
}
```

Proving a big-O is “tight”

- Easy to find a big O for something
  - Can technically just say O(n!) and be right for just about anything
- We want the smallest big O that is correct
- Need a “witness”
- Provide an example of an input that does cause the worst case scenario that happens
- When this happens you can say that the big O result is “Tight”

Amortization

- Averaging large numbers of costs

```
int size=0;int capacity = 10;
int *arr = new int[capacity];
void insert(int n){
     if(size > capacity){
          capacity *= 2
          int *temp = new int[capacity];
          for(int i = 0; i < capacity/2; i++) temp[i] = arr[i];
          delete arr;
          arr = temp;
     }
     arr[filled++] = n;
}
```

Space Complexity

- Works exactly the same

- Count memory used instead of time

Spacetime complexity

- O(space)*O(time)
- Also known as Area-Time Complexity
- Seems useful, but...
- Amortization causes problems

Fairly recent: Cumulative complexity

Formally defined using something called a pebbling game

Not required to know, but look up if interested

Idea: “integrate” the space complexity over time

Uses in cryptocurrency and hash functions, but more on that later

Exercise: Cumulative complexity calling resize

```
int size=0;int capacity = 10;
int *arr = new int[capacity];
void insert(int n){
     if(size > capacity){
          capacity *= 2
          int *temp = new int[capacity];
          for(int i = 0; i < capacity/2; i++) temp[i] = arr[i];
          delete arr;
          arr = temp;
     }
     arr[filled++] = n;
}
```

Questions?

Stacks and Queues

- Basic Abstract Data Type (ADT)
- Queues used for First In First Out (FIFO)
- Stacks are Last In First Out (LIFO)

Queues

- Works like a line, implements the following functions at least
- Insert(item i) - adds i to the end of the queue
- remove() - removes an item from the front of the queue and returns it

Stacks

- Last item put in is the first item out
- push(item i) - puts item i on the top of the stack
- pop() - returns and removes the top item of the stack

Linked Lists

- One way of implementing stacks and queues
- On board...

Arrays

- Store elements in an array
- Keep pointers to head or tail. Allows wraparound behavior on queues
- Allows random O(1) access

Array doubling

- Wait until full, then double array and copy
- Amortized analysis - prove running time
- O(1) insert and delete
- Shrinking also allowed - do at ¼ capacity

Which is more popular?

- What do standard libraries use?
- Java: Array
- C++: Array
- C: “Stop being lazy and just make/find one yourself”

Deques

- Mashup between stacks and queues
- Allows enqueues from either end instead of one
- For stacks you just use enqueue_front() and dequeue_front() (or back. Doesn’t matter)
- Many languages like Python or C++ will make you use these instead of a specific stack or queue

Possible event of interest

June 14, 2017

2:00 p.m.

HAAS Hall Rm. 111

This talk presents numerous results from the area of quantitative investigations in logic and computability. We present a quantitative analysis of random formulas or random lambda terms or random combinatory logic terms. Our main goal is to investigate likelihood of semantic properties of random computational objects. For the given logical calculus (or type theory or combinatory logic term) we investigate the proportion of the number of distinguished formulas (or types or terms) of a certain length n to the number of all formulas of such length. We are especially interested in asymptotic behavior of this fraction when n tends to innity. For the given set A of objects the limit $\mu(A)$ if exists, is an asymptotic probability of nding formula from the class A among all formulas or may also be interpreted as the asymptotic density of the set A. Results obtained are having some philosophical flavor like for example:

1. How big is the fraction of one logic being sub-logic of the bigger one based on the same language,
2. Estimate chances that random formula is true or how big is the fraction of tautologies?
3. What are chances that random program terminates?

About events

- CS Dept often hosts interesting speakers
- Email list you can sign up for to receive notices about talks
- Also for those interested: Dedicated cryptography group meets during the year, weekly talks about current research
- Email me if you are interested in either of these