In class notes for July 5, 2017

Data Structures and Algorithms

Admin

- Codio wont be used on the first homework
- Text files or pdf's for homework– that's it

Big-Oh Analysis

What are our metrics for measuring algorithms and runtimes? * Worst Case Scenarios * Best Case … * Average Case … ### Whats the point? We really want to abstract ourselves away from the data, the hardware, other processes running at the same time on the particular computer. * We can get away from this by counter arithmetic/logical/boolean/assignment/etc operations there are. Then we can also count loops… * This will lead to some expression that we can analyze and give a 'work-estimate'

```
int sum = 0;
for (i=0; i < a.length; i++) {
    sum += a[i];
}
```

We count the comparison i < a.length, i++, sum += a[i] where sum += is actually 2 operations and fetching a[i] is also one… but this is too close to the code we want to be abstracted more from this. We don't really care–doesn't really matter–about the coefficients or the constants added to other terms.

Because we're doing asymtotic analysis we don't care about those. Thus, our algorithm above we can say does `n` amount of work.

What if we had an snippet of code that followed the code above? Then we would have: * n + nˆ2

Now we can ignore the n, becuase we only care about nˆ2, again we're doing asymtotic analysis.

Defining Big-Oh

T(N) = O(f(N)) such that there exists some constants c and n_0 where T(n) is less than or equal to C(f(N)) for all n greater than or equal to n_0. This is an upper bound.

Big Omega

T(N) = Omega(g(N)) such that there exists some constants c and n__0 where T(N) greater than or equal to c(g(N)) for all n greater than or equal to of T(N). This is a lower bound.

Big Theta

T(N) = Theta(h(N)) if T(N) = Theta(h(N)) and T(N) = Omega(h(N)) which defines our tightest bound possible.

Note that if something is intuitively n/2 in the average case, we find that the average case is actually just n, because 1/2 is just a constant like before.

n	description
c	order of on
log__n	order of log of n
log^2__n	log squared n
n	linear algorithm
n log__n	n times log of n
n^2	polynomial
n^3	cubic
n^...	etc
2^n	exponential algorithm
n!	Factorial

Some rules for Algorithm Analysis

```
1. t_1(n) is the cost of the first and t_2(n) is the cost of a second algorithm, then t_1(n) + t_2
    * ''' ''' ''' t_1(n) * t_2(n) = O(f(n)g(n))
   * this is an example of nesting algorithms inside of another algorithm.
   * its important to think about this when calling methods that are provided in the java api. You
2. if T(n) is a polynomial of degree k, then T(n) = Theta(n^k)
3. log^k_n = O(n)
```

Example

```
for (i = 0; i < n; i++) {
    for (j=i; j<n; j++) {
        someValue++;
    }
}
```

The first pass will be n, then the second n-1 … What does this add up to? * (n(n+1))/2 which is n^2/2 + n/2 which will still end up as O(n^2)

Note

You can look at the increment in a for loop to illuminate the cost of a for loop. For example if the increment is n/2 then the work will be logirithmic.

Conditionals

```
if (conditional)
    algorithm1;
else
    algorithm2;
```

How do we describe this? Usually you'll want to use the worst case, the max(alg1, alg2)

Abstract Data Types (ADT)

A concept for describing a set of operations that we want to perform. What are the things that we should be able to do with this? ### List * This is an ADT. It is a collection of data, that has some order. We might want to get an element in location i, or set an element, remove, or add an element. These are intrintrinsic behaviors of a list. ### Interfaces * This is very close to an abstract data type in Java. For interfaces, we say that there are a set of operations * An interface can extend other interfaces, which means that whatever the super class has, the subclass will also have. * Iterator is extended by the Collection interface. This lets you iterate over the elements in the collection. Iterator comes with methods like hasNext() and next(). This essentially keeps track of what it already gave you. * You can have multiple iterators on the same list. This would be two seperate objects ### Inside the Java api

```
public interface Collection<AnyType> extends Iterable<AnyType> 2{
    int size( );
    boolean isEmpty( );
    void clear( );
    boolean contains( AnyType x );
    boolean add( AnyType x );
    boolean remove( AnyType x );
    java.util.Iterator<AnyType> iterator( );
    //this is called a factory method
 }
```

ArrayLists

- Actual concrete class in Java that implements the list interface
- This is a realization of the list interface
- What do we need?
  1. an array
  2. a length
- When we're adding items to an ArrayList, we're just filling up an array data type. But when we need to resize the initial array, we'll get hit with an order of n penalty, because we'll have to resize the array to accomodate the new element.
- Adds are always expensive for arrayLists because you'll have to insert the new element somewhere, then copy over all the old elements. This is the of the order of n for the cost of work.

LinkedLists

Singly Linked Lists

- The idea is to allocate space for only things that we need– no more, no less.
- We do this by creating Node, which is a class that defines 2 items.
  1. Data
  2. Next pointer (node)
- If I want to provide a reference, then we'll provide a reference to the first node.
- Node class is generic

  ```
  public class LLNode<AnyType>() {
  AnyType data;
  LLNode<AnyType> next;
  }
  ```
- Adding an element to the end of a list is of the order of n
- Finding an element in index i is also of the order n

Doubly Linked Lists

- Java's LinkedLists are doubly linked lists
- Doesn't just have a next field but also a previous field, so our nodes might look something like: "'java public class DoublyNode() { DoublyNode next; DoubleNode previous; Anytype data; }

- We can use sentinal nodes in order to bound our list. The first node in our list doesn't have any data– called the head– and our last node doesn't have any data– called the tail node.
- Therefore, a DoublyLink list of size 0 has 2 nodes, then a DoublyLinkedList of size 4 has 6 nodes.
- Splicing a node

  ```
  addAfter(k) = Node(30);
  t.prev = k;
  t.next = k.next;
  k.next = t;
  t.next.prev = t;
  ```

  Fridays 6-8 Tues/Thurs 7-9 Weds 1-3PM