CSM Week 5 Review

Abstract Data Types

Abstract Data Types (ADT) are blueprints for data structures. They contain methods and types that would define a data structure (which is why ADT's are interfaces!). Here are the ADT's we went over this week:

	List	Set	Map
Ordered	Yes	No	keys: No, values: No
Repeats	Yes	No	keys: No, values: Yes
Data structures	LinkedList ArrayList	HashSet TreeSet	HashMap TreeMap

Amortized Analysis

Amoritzed runtime, also known as *average runtime*, is defined as the cost per operation, evaluated over a sequence of operations. It is also written in terms of **θ notation**. Intuitively, think of it as the cost that occurs the most often in a sequence of operations.

Example (taken from CS 61B Fall 2017 Guerrilla Section 3)

```
public class SQueue {
        private Stack inStack ;
        private Stack outStack ;

        public SQueue () {
                inStack = new Stack () ;
                outStack = new Stack () ;
        }

        public void enqueue (int item ) {
                inStack . push ( item ) ;
        }

        public int dequeue () {
                if ( outStack . isEmpty () ) {
                        while (! inStack . isEmpty () ) {
                                outStack . push ( inStack.pop() ) ;
                        }
                }
                return outStack . pop () ;
        }
}
```

What is the amortized time to dequeue an item from an SQueue containing N elements?

Best case: θ(1)

The best case occurs when outStack is not empty. We would not go into the if-statement and while loop. It would simply take θ(1) to pop an element from the stack!

Worst case: θ(N)

The worst case occurs when the outStack is empty and we must fill it with N elements. This occurs the first time we call dequeue and happens only **once**.

Amortized Runtime: θ(1)

We setup the equation below because the runtime of the sequence of calls to θ(1) is θ(N) + θ(1) + θ(1) +....., which is simplified to θ(N).

We then divide this by the number of operations, which is N + 1 (1 operation/call to dequeue to add the element to the outStack and N calls to dequeue to pop the N elements). This also simplifies to N.

$$\frac{\theta(N)}{N} = \theta(1) \text{ amortized runtime!}$$