CSE-250 Recitation

Sept 18-19 : Amortized Runtime

Amortized Runtime Analysis

```
1  public class Team {
2    private List<Player> players;
3
4    public void addPlayer(Player p) { /* ... */ }
5    public void importRoster(File f) { /* ... */ }
6    /* ... */
7  }
```

```
1  public void addPlayer(Player p) {
2    System.out.println("Welcome to the team " + p.name());
3    players.add(p);
4  }
```

What are the unqualified and amortized runtime bounds of the **`addPlayer`** method when **`List`** is a **`LinkedList`**? an **`ArrayList`**?

Amortized Runtime Analysis

```
1  public void importRoster(File f) {
2    BufferedReader br = new BufferedReader(new FileReader(f));
3    String line;
4    while (br.ready()) {
5      String line = br.readLine();
6      Player p = new Player(line);
7      players.add(p);
8    }
9  }
```

What are the unqualified and amortized runtime bounds of the **`importRoster`** method when **`List`** is a **`LinkedList`**? an **`ArrayList`**?

*(You can assume that opening the file, reading a line, and creating a **`Player`** are constant-time calls)*

Rationalization of Array Append's Amortized O(1) runtime.

After n calls to an array whose capacity is n, $\Theta(2n)$ operations will have been performed
(n constant time appends + n copies into the allocated array).
A reallocation has occurred i times only if there have been 2^i calls to append.
This describes a total of 2^i constant time appends plus

2^0 + 2^1 + 2^2 + 2^3 + … + 2^(i-1} + 2^(i) constant time copy operations.

The total sum is 2^i + sum_{a=0}^{i} { 2^a } = 2^i + 2^(i + 1) -1 = 2^i + 2*2^i - 1 = 3*2^i - 1.
So for n = 2^i calls to append, the Amortized Runtime is O(3*2^i - 1 / 2^i) = O(3*2^i / 2^i) = O(3)
This is not a fluke, nor is it magic. It is a property of summing powers of 2.
Simply, 2^i + 2^i = 2*2^i = 2^(i+1). At any point in the runtime of sequential append, at the worst case, we will be doing merely double (or some constant factor) the amount of work we had been doing previously. (Sequentially, O(n) operations, worst case O(2n))