Cracking the Coding Internview - Notes

# Chapter 1

# IX – Interview Questions

## 1. Arrays and Strings

* Array questions and string questions are often interchangeable
  + So questions asked here for arrays may be asked as string questions and vice versa

### Hash Tables

* A hash table is a data structure that maps keys to values for efficient look up
* … know this stuff …
* If collisions are high, worst case is , where is the number of keys
  + Usually you assume good implementation that keeps collisions to a minimum and have lookup time of
* Apparently, *you can also implement a Hash Table using a binary search tree*
  + This gives lookup time
  + The advantage being that you may use less space since you don’t have to allocate a large array
  + You can also iterate through keys in order

### ArrayList & Resizable Arrays

* In some languages arrays are automatically resizable: the array/list grows as you add items
* In Java, arrays are a fixed length defined during creation
* Dynamically resized array is an ArrayList
  + Provides access
* When full, doubles in size, taking time
  + But this is rare so it is **amortized for insertion**
* Why is it amortized ?
  + Say you’re resizing an array of size *N*, you can work backwards and see how many elements you copied each time the capacity increased
  + When we increase it to *K* elements, it was previously half that size, so you needed to copy elements
  + Before that, you copied over elements, and before that , etc. until you started with 1 (well I guess usually you’d start at more than that but for the sake of understanding)
* **Resizing factor:** Size of new array relative to current **(2 in Java)**

### StringBuilder

* Looking at the code below, what would be it’s running time if each string you’re adding has length *x* and there are *n* strings?

1. String joinWords(String[] words) {
2. Sting sentence = "";
3. **for** (String w : words) {
4. sentence = sentence + w;
5. }
6. **return** sentence;
7. }

* Each time you concatenate the string, you create a new copy of it
  + I think what they mean by that is that you’re copying every character from *sentence* and every character from *w*
  + The first iteration copies *x* characters, the second copies *2x* charaters, then *3x*, etc.
  + So the total runtime is
* For a proof that , check: <http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/runsums/triNbProof.html>
* *StringBuilder* avoids this problem by creating a resizable array of all the strings
  + It copies them only when necessary

1. String joinWords(String[] words) {
2. StringBuilder sentence = **new** StringBuilder();
3. **for** (String w : words) {
4. sentence.append(w);
5. }
6. **return** sentence.toString();
7. }

* She recommends you try to implement