Combinatorics Review

We'll talk a lot in this course about **brute force** algorithms: this refers to solving a problem in the most straightforward way, without trying to be clever. Often, it means generating all possible solutions and testing all of them until we find a solution that works. Sometimes, this approach will work for us (and when it does, that's great: brute force algorithms tend to be easy to implement precisely because they aren't clever). But sometimes the number of solutions to check is so large that brute force isn't practical.

Basic combinatorics can be useful in determining, without needing to implement and test a brute force algorithm, whether brute force will be a feasible approach for the problem we're trying to solve. "Combinatorics" is really a fancy word for "counting." We can use some tools from combinatorics to count how many possible solutions exist to a particular problem, which we can use to derive an asymptotic bound on the runtime of a brute force approach to the problem.

This isn't intended to be a comprehensive overview of combinatorics. Rather, we just want to provide enough information that you can easily determine running times of some algorithms that you're likely to encounter in this course and in the future.

A few useful formulas

- Number of ways to sample from k items n times, with replacement: k^n
- Number of ways to order n distinct elements: $n!$
- Number of distinct ways to order items in m distinct groups $g_1, g_2, \ldots g_m$, each consisting of n_i identical objects: $\frac{(\sum_{i=1}^{m} n_i)!}{\prod_{i=1}^{m}(n_i!)}$
- Number of combinations of size k taken from n objects: $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$
- Number of permutations of size k taken from n objects: $\frac{n!}{(n-k)!}$

Sampling with replacement

1. Suppose you have an urn that contains a red ball, a green ball, and a blue ball. You pick a ball out of the urn, put it back in the urn, and pick another ball. If you do this three times, you would observe some sequence of the three colours – for example, you could pick the red ball, the blue ball, and the blue ball again. We'll denote the sequence {red, blue, blue} by RBB.

   If you pick a ball n times, how many possible colour sequences could you observe?

2. Consider the Boolean satisfiability (SAT) problem: given n TRUE or FALSE variables $x_1, \ldots, x_n$, and a Boolean expression consisting of those variables joined by AND, OR, NOT, and parentheses, is the formula *satisfiable*? That is, can we assign TRUE or FALSE values to each of the n variables in some way that the entire Boolean expression evaluates to TRUE?

   We want to consider a brute force approach to this problem. How many different possible solutions will we have to check?

Permutations

3. In how many different ways can you rearrange the letters in the word "dermatoglyphics"[1]?

[1] The scientific study of fingerprints. It's also the longest word in the English language with no duplicated letters – along with "uncopyrightable."

4. Consider the Traveling Salesperson Problem: given a set of n cities, what is the shortest tour that visits all the cities (i.e., what ordering of cities results in the least possible distance traveled)?

   We want to consider a brute force approach to this problem. How many different possible solutions will we have to check?

Permutations with duplicates

5. In how many different ways can you rearrange the letters in the word "Mississippi"?

6. How many different possible solutions do we need to consider in a brute force algorithm for the Resident Hospital Problem (RHP) with n residents and m hospitals each containing n_i slots? (We did also answer this in class; but if you were at all confused by that explanation, this is a good time to think about it a bit more and/or ask for some clarification if you're still having trouble.)