Problem 1 (Sportsball)
Consider the problem of filling basketball teams. A single basketball team must have five people. Give combinatorial descriptions for each of the required values.

(a) Basketball teams have five basic positions—center, power forward, small forward, shooting guard, and point guard. If there are \( s \) students to choose from, how many ways can I fill a single basketball team where each student is assigned to one of these positions?

(b) In a pickup game of basketball, the teams typically don’t make a distinction between positions. In this context (where players are simply “on” a team without being assigned a position), how many ways are there to fill a team when there are \( s \) students to choose from?

(c) Now suppose you are a high school P.E. coach and you are filling a team of mixed grades. Such a team has at least 1 player that is in each of 9th, 10th, 11th, and 12th grades. The last slot may be drawn from any of these players. Let the number of students from each grade be:

- 9th graders: \( a \)
- 10th graders: \( b \)
- 11th graders: \( c \)
- 12th graders: \( d \)

How many ways are there to fill a single team?

(d) At the college level, we only differentiate between three positions—center, forward, and guard. A team is normally composed of a center, two forwards, and two guards. Suppose there are \( s \) students to choose from. How many ways can we fill a single team where each student is assigned to one of these positions? (Hint: You want to rule out repetition where, e.g., student \( s_1 \) is first chosen to be a guard, then \( s_2 \) and the case where \( s_2 \) is first chosen, then \( s_1 \).)

(e) Suppose you are filling teams based on a rating system where:

- There are \( x \) 2 point players.
- There are \( y \) 3 point players.
- There are \( z \) 5 point players.

As a coach you may spend at most 15 points on forming a team of 5 players (but you may choose to spend less). How many different teams can you form in this system? (Hint: Systematically explore all the possible valid point combinations, determine how many teams you can form from each, and then combine everything together. Make sure you give an unsimplified formula for this problem so your derivation is clear.)

(f) Finally, suppose the pool of players consist of

- \( v \) point guards,
- \( w \) shooting guards,
- \( x \) small forwards,
- \( y \) power forwards, and
- \( z \) centers.

In a sentence or two, give a description of the quantity described by the following combinatorial description. Be explicit in your description when ordering matters.

\[
\binom{v}{1} \cdot \binom{x}{2} \cdot \binom{y}{1} \cdot \binom{z}{1}
\]
Problem 2 (Relationship Counting)

Consider the problem of counting binary relations over a domain of set $S$ and range of set $T$.

(a) How many possible \textit{functions} can we form with domain $S$ and range $T$? Justify your combinatorial description using a few sentences explaining how you constructed your formula.

(b) How many possible \textit{injective functions} can we form with domain $S$ and range $T$? Justify your combinatorial description using a few sentences explaining how you constructed your formula.

(c) Under what conditions (i.e., constraints on the sizes of $S$ and $T$) can we \textit{not} form any \textit{injective functions} between these sets? Prove this claim using the principles of counting.

(d) Similarly, under what conditions (i.e., constraints on the sizes of $S$ and $T$) can we \textit{not} form any \textit{surjective functions} between these sets? Prove this claim using the principles of counting.
Problem 3 (Down for the Count)

Consider the following Racket code:

\[
\text{define } f \text{ (lambda (n)} \\
\text{ (begin)} \\
\text{ (for-each (lambda (i)} \\
\text{ (display "!"))} \\
\text{ (range (quotient n 2)))} \\
\text{ (for-each (lambda (i)} \\
\text{ (for-each (lambda (j)} \\
\text{ (begin)} \\
\text{ (display "!”))} \\
\text{ (range n))})} \\
\text{ (range n)))})
\]

\[
\text{define } g \text{ (lambda (n)} \\
\text{ (if (<= n 1) (void) (begin (display "!”)) (display "!”) (display "!”) (g (- n 2)))})
\]

Or the equivalent C code:

\[
\text{void f(int n) \{} \\
\text{ for (int i = 0; i < n/2; i++) \{} \\
\text{ printf("!");} \\
\text{ \}} \\
\text{ for (int i = 0; i < n; i++) \{} \\
\text{ for (int j = 0; j < n; j++) \{} \\
\text{ printf("!");} \\
\text{ printf("!");} \\
\text{ \}} \\
\text{ \}}
\]

\[
\text{void g(int n) \{} \\
\text{ if (n <= 1) \{} \\
\text{ \} else \{} \\
\text{ printf("!");} \\
\text{ printf("!");} \\
\text{ printf("!");} \\
\text{ g(n - 2);} \\
\text{ \}} \\
\text{ \}}
\]

(a) Give two combinatorial descriptions for the number of !s printed by \( f \), the first using summation notation, and the second, an explicit formula without the use of summation notation. Your description should be a function of the input \( n \).
(b) Give a recurrence relation describing the number of ! s printed by g.
   (Hint: Your recurrence relation’s case should account for the fact that the recurrence goes down by two!)
(c) Derive a closed-form expression for this recurrence relation and prove that the closed-form expression is equivalent to the recurrence.
   (Hint: To navigate difficulties surrounding n^2 not necessarily being a natural number, use the floor function \( \lfloor \cdot \rfloor \).)
   (Hint: When proving your formula correct, note that because the recurrence ticks down by two, a standard induction hypothesis may not suffice.)
Problem 4 (Parallel Hell)
Consider the problem of analyzing programs operating in parallel over shared state. For example, here are two example program snippets written in C over a shared global variable `glob` with initial value 3.

```
// Program A
/* 1a */ int x = glob;
/* 2a */ glob = 5;
/* 3a */ glob = x * 2 + glob;

// Program B
/* 1b */ int y = glob;
/* 2b */ glob = glob - 2;
/* 3b */ glob = y;
```

If program A and B operated in serial, then execution of one program follows the other, so the value of `glob` will be 11. However, in a world where A and B operate in parallel, it is not clear which program executes first. On top of that, the instructions of A and B may be interleaved, leading to a variety of possible outcomes.

One way we can think of the possible interleavings of parallel programs is to serialize their executions, imagining how the two programs interact with each other through `glob` For example, one such interleaving might be:

```
1a, 2a, 1b, 3a, 2b, 3b
```

where program A executes its first two instructions, then program B executes its first instruction, then program A finishes, and then program B finishes after that. This results in a different value for `glob`, 5! (Trace through this execution to make sure you understand why the resulting value is 5).

In general, a serialized execution is an interleaving of the instructions of the parallel-executing programs involved. The only constraint on this execution is that the order of the instructions within a program is preserved. Thus, the following interleaving:

```
1a, 1b, 2a, 3b, 3a, 2b
```

is not a valid interleaving because `3b` comes before `2b`.

(a) Come up with a serialized execution above where `glob` has the value 3 at the end of the execution.
   (Hint: There’s not really a systematic way to approach this problem in general. You might find it more productive to simply enumerate possible executions until you stumble on one that produces the desired result.)

(b) The state explosion problem when performing static analysis over parallel programs come from the number of interleavings of instructions we must consider. Suppose that we are analyzing a pair of parallel programs that each have m and n instructions, respectively. Derive a combinatorial description of the number of possible serialized executions of these two programs and give a justification of why this description is accurate.
   (Hint: There is an elegant description of this situation! Think careful about the requirements of a particular interleaving. What must be true about the order of instructions? How does this constrain the possible ways we can layout instructions if we, for example, layout one set of instructions first?)
   (Hint: Draw diagrams to help you spatially understand these relationships!)

(c) What does the state explosion problem say about the problem of trying to analyze parallel programs? For example, is it feasible to perform brute-force search of interleavings to find those that exhibit bugs (such as race conditions)? Why?