1. **(20 points)** *Below we will be building a recursive formula to calculate the number of ways to build letter strings using each letter in our alphabet at least once. Let $S(n,k,r)$ represent the number of ways to build strings of k letters, drawn from an alphabet of n letters, if there are r letters which you are required to use at least once. For instance, the number of ways to build a 4-letter string using the letters A and B, if we are required to use at least one A, would be $S(2,4,1)$ (note that this is equal to 15).*

   (a) **(7 points)** *Explain why, for $k \geq r > 0$, it should be the case that*

   $$S(n,k,r) = rS(n,k-1,r-1) + (n-r)S(n,k-1,r)$$

   Let us call the letters we may use to build our length-k string $x_1, x_2, x_3, \ldots, x_n$, with the letters $x_1, \ldots, x_r$ required. The first letter of our string could be any of these n different letters, but depending on whether it is required or not, the number of letters required to complete the string may vary. If we choose any of the r letters $x_1, \ldots, x_r$, then that specific letter no longer needs to be used in the remaining $k-1$ elements of the string. In this scenario, we may select the first letter in any of r ways, and then the remaining $k-1$ letters in any of $S(n,k-1,r-1)$ ways, for a total of $rS(n,k-1,r-1)$ strings starting with any of $x_1, x_2, \ldots, x_r$.

   If, however, we start our string with any of the $n-r$ letters $x_{r+1}, x_{r+2}, \ldots, x_n$, we have not fulfilled any of our requirements, so that the remaining $k-1$ elements of the string are still required to contain at least one copy of all r required letters. Thus we may select the first letter in any of $(n-r)$ ways, with $S(n,k-1,r)$ ways to finish the string, so this scenario accounts for $(n-r)S(n,k-1,r)$ strings.

   Since every string falls into one of these two classes (starting either with $x_1, \ldots, x_r$ or with $x_{r+1}, \ldots, x_n$), the total number of possible strings counted by $S(n,k,r)$ will be the sum $rS(n,k-1,r-1) + (n-r)S(n,k-1,r)$.

   (b) **(7 points)** *Determine formulas (based on the definition of $S(n,k,r)$) for $S(n,k,0)$ and $S(n,k,k+1)$.*

   The definition of $S(n,k,0)$ is the number of possible k-element strings drawn from an n-element alphabet, with no restrictions on which letters must be used. This is a free selection, which we know to be possible in n^k ways, so $S(n,k,0) = n^k$.

   The definition of $S(n,k,k+1)$ is the number of possible k-element strings drawn from an n-element alphabet in which there are $k+1$ letters you are required to use at least once. With a string of length k, you could only reasonably use k different letters, so that this request is impossible to fulfill and thus $S(n,k,k+1) = 0$.

   (c) **(6 points)** *Use the results of the two previous sections to recursively calculate $S(3,5,3)$. Does this accord with information you've encountered elsewhere?*

   We can calculate $S(3,k,r)$ for various values of k and r; it proves easiest to do it starting from small values and working upwards.

$$\begin{aligned}
S(3,0,0) &= 3^0 = 1\\
S(3,0,1) &= 0\\
S(3,1,0) &= 3^1 = 3\\
S(3,1,1) &= 1S(3,0,0) + 2S(3,0,1) = 1\\
S(3,1,2) &= 0\\
S(3,2,0) &= 3^2 = 9\\
S(3,2,1) &= 1S(3,1,0) + 2S(3,1,1) = 5\\
S(3,2,2) &= 2S(3,1,1) + 1S(3,1,2) = 2\\
S(3,3,1) &= 1S(3,2,0) + 2S(3,2,1) = 19\\
S(3,3,2) &= 2S(3,2,1) + 1S(3,2,2) = 12\\
S(3,4,2) &= 2S(3,3,1) + 1S(3,3,2) = 50\\
S(3,5,3) &= 3S(3,4,2) + 0S(3,4,3) = 150
\end{aligned}$$

It is also viable, although perhaps easier to make a mistake with this approach, to start from the top and work down:

$$\begin{aligned}
S(3,5,3) &= 3S(3,4,2) + 0S(3,4,3)\\
&= 3S(3,4,2)\\
&= 3\left[2S(3,3,1) + 1S(3,3,2)\right]\\
&= 6S(3,3,1) + 3S(3,3,2)\\
&= 6\left[1S(3,2,0) + 2S(3,2,1)\right] + 3\left[2S(3,2,1) + 1S(3,2,2)\right]\\
&= 6S(3,2,0) + 18S(3,2,1) + 3S(3,2,2)\\
&= 6\cdot 9 + 18\left[1S(3,1,0) + 2S(3,1,1)\right] + 3\left[2S(3,1,1) + 1S(3,1,2)\right]\\
&= 54 + 18S(3,1,0) + 42S(3,1,1) + 3S(3,1,2)\\
&= 54 + 18\cdot 1 + 42\left[1S(3,0,0) + 2S(3,0,1)\right] + 3\cdot 0\\
&= 54 + 54 + 42S(3,0,0) + 84S(3,0,1)\\
&= 54 + 54 + 42\cdot 1 + 84\cdot 0 = 150
\end{aligned}$$

In either case we get an answer of 150; note that this accords with a number you were given early in class, and one that appeared in question 1(b) of the last problem set for the same enumeration.

2. **(10 points)** *I have selected a set of 30 different integers between 1 and 1000 (inclusive). Prove, without knowledge of which 30 integers I chose, that you can find seven different (but possibly overlapping) four-element subsets of my set whose elements all add up to the same number.*

   Whatever set I chose, you could select $\binom{30}{4} = 27405$ different four-element subsets of it. Let us treat those subsets as "pigeons" and classify them into "pigeonholes" according to what the sums of their elements are. THe smallest possible such sum is $1+2+3+4=10$, and the largest is $997+998+999+1000=3994$, so there are a total of 3985 different possible total values, i.e. 3985 pigeonholes. Noting that $\frac{27405}{3985} > 6$, the pigeonhole principle guarantees that seven of our pigeons (i.e. four-element sets) nest in the same pigeonhole (i.e. sum of their elements).

3. **(10 points)** *If you have a list of n numbers, how many operations it take to determine whether there are numbers x, y, and z in the list such that $x+y=z$? Put your answer in "big-O" form and briefly explain your algorithm.*

   An easy algorithm is to brute-force your way through all the elements of the list, according to the following pseudocode procedure:

```
• For each value of $i$ from 1 to $n$, do the following:
    – For each value of $j$ from 1 to $n$, do the following:
        ∗ For each value of $k$ from 1 to $n$, do the following:
            · If the $k$th element of the list is equal to the sum of the $i$th and $j$th
              elements of the list, report success and stop.
• If we reach the end without reporting success, report failure and stop.
```

   Because this procedure laboriously looks at each triple (i, j, k) with coordinates between 1 and n, it may have to check n^3 distinct cases. The check itself is a single step of simple arithmetic, but doing it up to n^3 times will take $O(n^3)$ time; note that this is a worst-case estimate when the algorithm fails to find a triple. For very cooperative data, the algorithm may end very quickly!

   This brute-force approach can be improved on. If we sort the list in advance (which can be done in $O(n \log n)$ time), then instead of scanning through the entire list looking at each x, y, and z, we can scan through looking at just two variables (x and y are the most obvious), and then perform a *binary search* algorithm to see if the list contains the desired result $x + y$ somewhere. Binary search itself is a logarithmic-time procedure; iterated over every pair of possible values for x and y, we have to perform this logarithmic-time procedure n^2 times for a final computational complexity of $O(n^2 \log n)$ (which, notably, subsumes the cost of sorting completely, so we don't need to add any additional term to represent that extra step).

שתי אבנים בונות שני בתים: שלש אבנים בונות ששה בתים: ארבע אבנים ב ונות ארבעה ועשרים בתים: חמש אבנים בונות מאה ועשרים בתים: שש אבנים בונות ש בע מאות ועשרים בתים: שבע אבנים בונות חמשת אלפים וארבעים בתים: מכאן ו אילך צא וחשוב מה שאין הפה יכול לדבר ואין האוזן יכולה לשמוע

—ספר היצירה פרק ד׳ משנה ט״ז

[Two stones (or letters) build two houses (or words), three stones build six houses, four stones build twenty-four houses, five stones build one hundred twenty houses, six stones build seven and twenty houses, seven stones build five thousand forty houses; thenceforth are numbers which the mouth can not speak and the ear can not hear.]

—Sefer Yetzira, Chapter 4, Verse 16