Additional Concepts from Module 2:

- **Gates** - Please review what are OR, AND, and NOT gates. Some people are better at using the logic tables to understand gates. If you do not like the logic tables in the book then you can think of gates like switches. For example, if a light bulb is connected to 2 switches:
    - OR - If either one of the switches or both switches are in the "on" position then the light bulb is on.
    - AND - Only if both switches are in the "on" position then the light bulb is on.
    - NOT - A light bulb is connected to 1 switch. If the switch is in the "off" position then the light bulb is on. If the switch is in the "on" position then the light bulb is "off".
    - **BONUS (extra credit in midterm):**
        - XOR - Also called exclusive OR gate. The light bulb is on only if 1 of the switches is in the "on" position. This means if both switches are in the "on" position then the light bulb is off. Also if both switches are in the "off" position then the light bulb is off.
        - NAND - Also called NOT AND gate. The light bulb is on when either switch is in the "on" position or both switches are in the "off" position. The light bulb is off when both switches are in the "on" position.
- **ASCII** - Characters (letters, punctuation, etc.) are stored as numbers, for the midterm I will give you a table of ASCII values and a set of numbers and you should be able to write out the characters and vice versa.

Concepts from Module 4 and 5:

- Search algorithms:
    - Search can mean: is the element in an array, is there an element larger, is there an element smaller, etc.
    - **Linear search** is when we compare element by element, example:
        - We want to know if there is 3 in the array [1, 4, 3, 6, 7]
        - We determine if 1 is equal to 4 (no), we move on to the next element, we determine if 4 is equal to 3 (no), we move on to the next element, we determine if 3 is equal to 3 (yes), we output something like True or we output an index. If we used a different number, let's say 10, we would not find it in the array so we would go through all 5 of the elements and output False or a stand-in index such as -1.
        - Pros: linear search is simple to implement, it has very low memory usage as we only need to compare the current value to the desired value
        - Cons: very slow, in the worst-case scenario we will require N-steps where N is the number of elements in the array so if N is very large like a billion we will need to perform a billion calculations. In algorithm analysis this is called O(n), pronounced Big-O of N.
    - **Binary search** is when we have a **sorted** array (from smallest to largest, sometimes but less common, largest to smallest) and start by going half way into the array and compare that value. If the desired value is greater than the value

at the halfway point then we jump to the halfway point between the first halfway point and the end of the array and compare. If the desired value is less than the value at the halfway point then we jump to the halfway point between the beginning of the array and the first halfway point and compare the values. We repeat this until we find the desired value and similar to linear search output we have found the value or we exhaust all comparisons in the array and output that we have not found the value.
        - Pros: much faster than linear search, it is O(log2(n)), so when we have very large arrays we can get our result much faster.
        - Cons: more complex meaning we could possibly mess up the algorithm when we write it, more memory complexity as we need to also have an extra halfway point variable (realistically when it comes to modern computers this is pretty much negligible).
    - The most important thing to consider is that an algorithm is correct first and foremost. Then we can consider how to make or choose a faster algorithm. Additionally, if the number of values you need to search is small, then a simpler algorithm might be better than a faster more complex algorithm. If the number of values is very large then a faster more complex algorithm may be necessary.
- Know what is ascending order and descending order, important for sorting:
    - **Ascending**: start at the lowest (smallest) value and each successive value is equal or greater than the previous value
    - **Descending**: start at the highest (greatest) value and each successive value is equal or less than the previous value