**What is Good Code**

* Readable and Scalable.
* How fast code runs cannot be measured effectively with built in functions. My computer’s CPU might be faster than someone else’s, which doesn’t speak for the code iteself.
* Big O is what is used to see how long an algorithm takes to run. Essentially, as you grow bigger with the input, how much does the algorithm slow down? Big O is machine independent. As the number of element s increase, how many operations are increased?

**Big O**

* O(n) – Linear Time. You iterate over your input once. The number of operations increases linearly as the input size increases. Generally, when you loop over an input once.
* O(1) – Input size doesn’t matter, the number of operations will remain the same.
* You can simpify Big O into one of the following:
  + O(1)
  + O(log n)
  + O(n)
  + O(n log n)
  + O(n2)
  + O(2n)
  + O(n!)

**Rules**

1. Worst Case. Worst Case is what you want to focus on. For example, if you’re searching for a value in an array of length n, say O(n) as the complexity. What you’re searching for might be the first item, but in general you want to focus on the worst case.
2. Remove Constants – Big O describes long-term growth rate of functions themselves, not the magnitudes. Multiplying by a constant affects growth rate by a constant, so linear functions still grow linearly, quadratic still quadtratically, etc.
3. Different terms for inputs. Say you have two different arrays you’re looping over in entirety, called Array1 and Array2. Also, say that Array1 has size n, and Array2 has size m. The Big O of this is going to be O(n + m).
   1. Nested loops will result in O(n2) as a general rule. Say you had the Array1 and Array2 problem from above. This will result in O(n\*m), since the loops are now nested.
4. Drop Non-Dominants. Say you have an array of size n, and in the algorithm, you loop through this input once. After this, you have a nested loop, with each loop iterating over the input size. With the first loop, you have O(n). After this, you have a nested loop, so the total complexity now is O(n + n\*n) = O(n + n2). O(n2) will grow much faster with an increasing input size, compared to O(n). Therefore, O(n2) is the dominant term in this example, so we can drop the O(n). After simplification, the total complexity of this example is then O(n2).

**Pillars of Programming**

* Scalable means speed and memory. In other words, speed and space complexity analysis.
* What criteria is used to measure which code is best?
  1. Readable
  2. Speed
  3. Space/Memory

**Space Complexity**

* Space complexity will not look at the input size itself. It measures how much space you need after receiving the input. Say you have an array of size n as the input, and you need to create another array of size n for your algorithm. The space complexity is O(n) instead of O(n + n), since we don’t count the input array in complexity analysis.