Big-O Practice Problems:

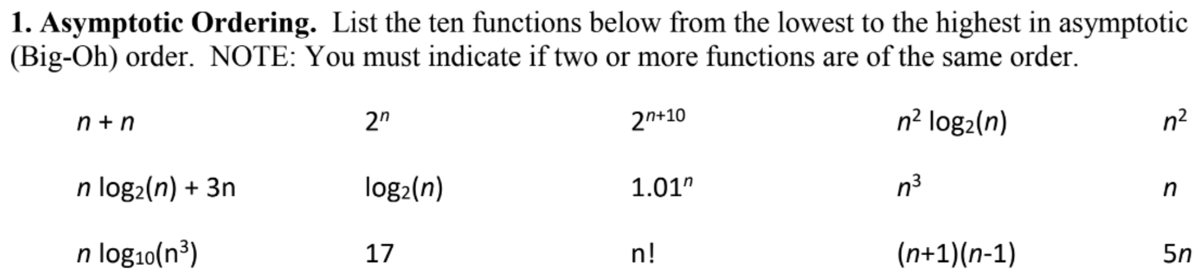
1. What does it mean if:
   $f(n) \neq O(g(n))$ and $g(n) \neq O(f(n))$???

2. Is $2^{n+1} = O(2^n)$?
   Is $2^{2n} = O(2^n)$?

3. Does $f(n) = O(f(n))$?

4. If $f(n) = O(g(n))$ and $g(n) = O(h(n))$,
   can we say $f(n) = O(h(n))$?

5.



1. Asymptotic Ordering. List the ten functions below from the lowest to the highest in asymptotic (Big-Oh) order. NOTE: You must indicate if two or more functions are of the same order.

$n + n$	2^n	2^{n+10}	$n^2 \log_2(n)$	n^2
$n \log_2(n) + 3n$	$\log_2(n)$	1.01^n	n^3	n
$n \log_{10}(n^3)$	17	$n!$	$(n+1)(n-1)$	$5n$

6. Finding one hundred. Many possible algorithms exist to determine if any pair of values in a vector add up to 100. For each of the four programs below, identify the asymptotic worst-case running time.

A: Use for loops to consider each possible pair

```
bool HasSum100_A(std::vector<int> vals) {

  for (int i = 0; i < vals.size(); i++) {
    for (int j = 0; j < i; j++) {
    if (vals[i] + vals[j] == 100) return true;
    }
  }
  return false;
}
```

B: (**C++ Challenge Problem)** Test current pair and then recurse (with a pass by reference!) for the next one until all options are considered.

```
bool HasSum100_B(std::vector<int> & vals, int i=1, int j=0) {
  if (i >= vals.size()) return false;         // Base case!
  if (vals[i] + vals[j] == 100) return true; // Found pair!
  ++j;                                       // Next pair!
  if (j == i) { ++i; j = 0; }                // (adjust i if needed)
  return HasSum100_B(vals, i, j);            // Keep testing!
}
```

C: Test current pair and then recurse (with a pass by value!) for the next one until all options are considered.

Hint: When passing a vector by value, all elements in the array are copied in Q(n) time.

```
bool HasSum100_C(std::vector<int> vals, int i=1, int j=0) {
  if (i >= vals.size()) return false;          // Base case!
  if (vals[i] + vals[j] == 100) return true;   // Found pair!
  ++j;                                          // Next pair!
  if (j == i) { ++i; j = 0; }                   // (adjust i if
needed)

  return HasSum100_C(vals, i, j);               // Keep testing!

}
```

D: Sort values first and then, for each value in the vector, search for the remaining value needed to sum to 100.

Hint: `std::sort` takes Theta(n log n) time and std::binary_search finds a value in Theta(log n) time.

```
bool HasSum100_D(std::vector<int> vals) {

  std::sort(vals.begin(), vals.end());   // Sort all values!

  for (int i = 0; i < vals.size(); i++) {

     int val_needed = 100 - vals[i];

     bool found = std::binary_search(vals.begin(), vals.end(),
val_needed);

     if (found) return true;

  }

  return false;

}
```

Testing asymptotic notation:

Use this website (https://rithmschool.github.io/function-timer-demo/) to test out how asymptotic notation works in the real world!

Choose one of the 7 functions along the top and do the following:

1. From looking at the code, try to figure out what complexity class you expect it to be in
2. Try gradually increasing the problem size and running the code to see how long it takes (be careful to increase N gently, as the website will hang if you give it too large a value)
3. Does the time complexity you observe match what you were expecting based on the code?
4. If you have time, try another function and see how the results compare!

Note: I recommend starting with a function other than logAllBinaries - it is substantially more challenging than the others.