1. **Asymptotic comparisons.** Order the following functions from slowest-growing to fastest-growing. Some of them may be equal (i.e., may belong to the same $\Theta$ class).

\[
\begin{align*}
2n^2 & \quad 65n^{4321} & \quad \lg n & \quad 1700n & \quad 1.5^n & \quad n^3 + 4n^2 + 9000 & \quad n & \quad \frac{1}{1000}n^3
\end{align*}
\]

**Solution:**

1. \(\lg n\)
2. \(n\) and \(1700n\)
3. \(2n^2\)
4. \(\frac{1}{1000}n^3\) and \(n^3 + 4n^2 + 9000\)
5. \(65n^{4321}\)
6. \(1.5^n\)

2. **Analyzing algorithms.** Suppose I wrote an expression, \(T(n)\), for the runtime of my algorithm, and then I compared \(T(n)\) to some other functions as follows:

Which of the following claims are true about the worst-case performance of my algorithm? (More than one claim may be true).

I. It’s \(\Theta(f(n))\).
II. It’s \(\Omega(h(n))\).
III. It’s \(O(g(n))\), provided \(T(n)\) describes the runtime of the worst possible input.
IV. It’s \(O(g(n))\).
V. It’s $\Theta(f(n))$, provided $T(n)$ describes the runtime of the worst possible input.

VI. It's $\Omega(f(n))$.

**Solution:** II, III, V, and VI are all true.

Some more detail: unless we’re told that $T(n)$ describes the runtime of the worst possible input, we don’t know what input it is talking about. It could be the runtime for the worst-case input, but it also could be the runtime for the best-case input, or some other input in between.

In case I, we can see from looking at the graph that $T(n) \in \Theta(f(n))$, but there could be some other, worse input that produces a curve that grows much faster. So we can’t conclude that the worst-case runtime is $\Theta(f(n))$.

In case II, we can see from the graph that $T(n) \in \Omega(h(n))$. $T(n)$ might not describe the worst-case input, but the worst-case input certainly is at least as bad as $T(n)$, so the worst case must also be $\Omega(h(n))$.

In case III, we are told that $T(n)$ is the worst case, and we can see from the graph that $T(n) \in O(g(n))$.

In case IV, as in case I, we don’t know whether $T(n)$ is the worst-case input, so there could be some other worse input that grows faster than $g(n)$.

In case V, we are told that $T(n)$ is the worst case, and since $T(n)$ falls between $c_2 f(n)$ and $c_2 f(n)$ we know that $T(n) \in \Theta(f(n))$.

In case VI, as in case II, we know that $T(n)$ is lower bounded by $c_3 f(n)$ so $T(n) \in \Omega(f(n))$. There could be a worse input that grows faster than $T(n)$, and that input would also be lower bounded by $c_3 f(n)$.
3. Writing recurrences. Here’s an algorithm that you may (or may not) have seen before:

```python
BinarySearch(A, x, i, j) // A is an array of length n
    if j < i return false
    mid = (i+j)/2
    if A[mid] == x return true
    if A[mid] > x return BinarySearch(A, x, i, mid-1)
    else return BinarySearch(A, x, mid+1, j)
```

Write a recurrence for $T(n)$, the runtime of BinarySearch. You do not need to solve your recurrence (but feel free to if you want the practice!)

**Solution:**

$$T(n) = T(n/2) + O(1)$$

Explanation: Let $n$ be the size of the array on any given iteration of the recursive function. Then each time through the function, you will perform a comparison and (if you haven’t found $x$) search an array of size $n/2$.

This recurrence turns out to be $T(n) = O(\log(n))$. 