Asymptotics is Fun!

Here is a video walkthrough for all parts of the problem.

(a) Using the function `g` defined below, what is the runtime of the following function calls? Write each answer in terms of `N`.

```
1 void g(int N, int x) {
2     if (N == 0) {
3         return;
4     }
5     for (int i = 1; i <= x; i++) {
6         g(N - 1, i);
7     }
8 }
```

`g(N, 1)`: $\Theta(\quad)$

`g(N, 2)`: $\Theta(\quad)$

Solution:

`g(N, 1)`: $\Theta(N)$

Explanation: When `x` is 1, the loop gets executed once and makes a single recursive call to `g(N - 1)`. The recursion goes `g(N)`, `g(N - 1)`, `g(N - 2)`, and so on. This is a total of N recursive calls, each doing constant work.

`g(N, 2)`: $\Theta(N^2)$

Explanation: When `x` is 2, the loop gets executed twice. This means a call to `g(N)` makes 2 recursive calls to `g(N - 1, 1)` and `g(N - 1, 2)`.
From the first part, we know `g(..., 1)` does linear work. Thus, this is a recursion tree with N levels, and the total work is $(N-1)+(N-2)+...+1 = \Theta(N^2)$ work.

(b) Suppose we change line 6 to `g(N - 1, x)` and change the stopping condition in the for loop to `i <= f(x)` where `f` returns a random number between 1 and `x`, inclusive. For the following function calls, find the tightest Ω and big O bounds.

```
1 void g(int N, int x) {
2     if (N == 0) {
3         return;
4     }
5     for (int i = 1; i <= f(x); i++) {
6         g(N - 1, x);
7     }
8 }
```

`g(N, 2)`: $\Omega(\quad)$, $O(\quad)$

`g(N, N)`: $\Omega(\quad)$, $O(\quad)$

Solution:
`g(N, 2)`: $\Omega(N)$, O(2^N)
`g(N, N)`: $\Omega(N)$, O(N^N)
Explanation: Suppose `f(x)` always returns 1. Then, this is the same as case 1 from (a), resulting in a linear runtime.
On the other hand, suppose `f(x)` always returns x. Then `g(N, x)` makes `x` recursive calls to `g(N - 1, x)`, each of which makes `x` recursive calls to `g(N - 2, x)`, and so on, so the recursion tree has 1, x, x^2 ... nodes per level. Outside of the recursion, the function `g` does `x` work per node. Thus, the overall work is $x * 1 + x * x + x * x^2 + ... + x * x^{N-1} = x(1 + x + x^2 + ... + x^{N-1})$.
Plug in `x = 2` to get $2(1 + 2 + 2^2 + ... + 2^{N-1}) = O(2^N)$ for our first upper bound. Plug in `x = N` to get $N(1 + N + N^2 + ... + N^{N-1}) = O(N^N)$ (ignoring lower-order terms).