Give an analysis for the towers of hanoi function. Write a recurrence equation and then find a tight big-oh.

```
void moveDisks (int n, char fromPeg, char toPeg, char auxPeg) {
   if (n == 1) {
      cout << setw(6) << n
           << "            " << fromPeg << "            " << toPeg << endl;
   }
   else {
      moveDisks(n-1, fromPeg, auxPeg, toPeg);
      cout << setw(6) << n
           << "            "  << fromPeg << "            " << toPeg << endl;
      moveDisks(n-1, auxPeg, toPeg, fromPeg);
   }
}
```

Recurrence equation:

$$T(n) = \begin{cases} O(1) & \text{if } n = 1 \\ O(1) + T(n-1) + T(n-1) & \text{if } n > 1 \end{cases}$$

Consider n>1:

$$T(n) = 1 + \underbrace{T(n-1)}_{1 + \underbrace{T(n-2)}_{1+T(n-3)+T(n-3)} + \underbrace{T(n-2)}_{1+T(n-3)+T(n-3)}} + \underbrace{T(n-1)}_{1 + \underbrace{T(n-2)}_{1+T(n-3)+T(n-3)} + \underbrace{T(n-2)}_{1+T(n-3)+T(n-3)}}$$

...

Each time, at each level of replacement, we get a one (really O(1)) appearing a power of 2 times. At the end, T(1) appears 2^(n-1) times.

$$= 1*(2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^{n-2}) + 2^{n-1}$$

$$= \sum_{i=0}^{n-1} 2^i = 2^n - 1 \qquad \text{So } T(n) = O(2^n).$$