Formal Complexity Analysis

- Formally, we define Big-O as follows:

Function $f(n)$ is $O(g(n))$ iff there exist positive constants $c$ and $n_0$ such that $f(n) \leq cg(n)$ for all $n$, where $n \geq n_0$. 
What is happening?

for (i = 0; i < howmany; ++i)
{
    for (j = i + 1; j < howmany; ++j)
    {
        if(nums[i] < nums[j])
        {
            temp = nums[i];
            nums[i] = nums[j];
            nums[j] = temp;
        }
    }
}
What is the Computing Complexity?

In this case, the N we are talking about is the variable howmany. What we need to figure out is how many times the segment below is executed.

```cpp
if(nums[i] < nums[j])
{
    temp = nums[i];
    nums[i] = nums[j];
    nums[j] = temp;
}
```
Number of Iterations

For various values of i, let's take a look:

\[
\begin{array}{ll}
  i & \text{# of iterations} \\
  0 & N - 1 \\
  1 & N - 2 \\
  2 & N - 3 \\
\end{array}
\]

and you get the picture
What is \( f(n) \)?

- This means that if the function \( f \) represents the number of executions of the above segment, then \( f(N) = (N-1) + (N-2) + (N-3) + \ldots + 2 + 1 \).

- Those who have taken a statistics class or studied summations can see that this equates to \( f(N) = N(N-1)/2 \).

- We can see that this function \( f \) can be bounded by some polynomial of \( N^2 \).
Not so obvious

- What might not be so obvious is that:
  \[ f(n) \leq (1/2)n^2, \text{ for } n \geq 1 \text{ and therefore, } n_0 = 1, \ g(n) = n^2, \text{ and } c = 1/2. \]
- This implies that \( f(n) \) is \( O(n^2) \).
Graphically

f(N) is O(g(N))

\[ f(N) = \frac{N(N-1)}{2} \]

\[ g(N) = \frac{1}{2}N^2 \]
Other Computing Complexities

**Problem:** Give an algorithm that works in each of the following times:

1) $O(1)$
2) $O(n)$
3) $O(\log_2 n)$
4) $O(n^2)$
5) $O(n \log_2 n)$