Formal Complexity Analysis

- Formally, we define Big-O as follows:

Function f(n) is O(g(n)) iff there exist positive constants c and n0 such that f(n) <= cg(n) for all n, where n >= n0.

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.

```
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:

i	# of iterations
0	N - 1
1	N - 2
2	N - 3

and you get the picture

What is f(n)?

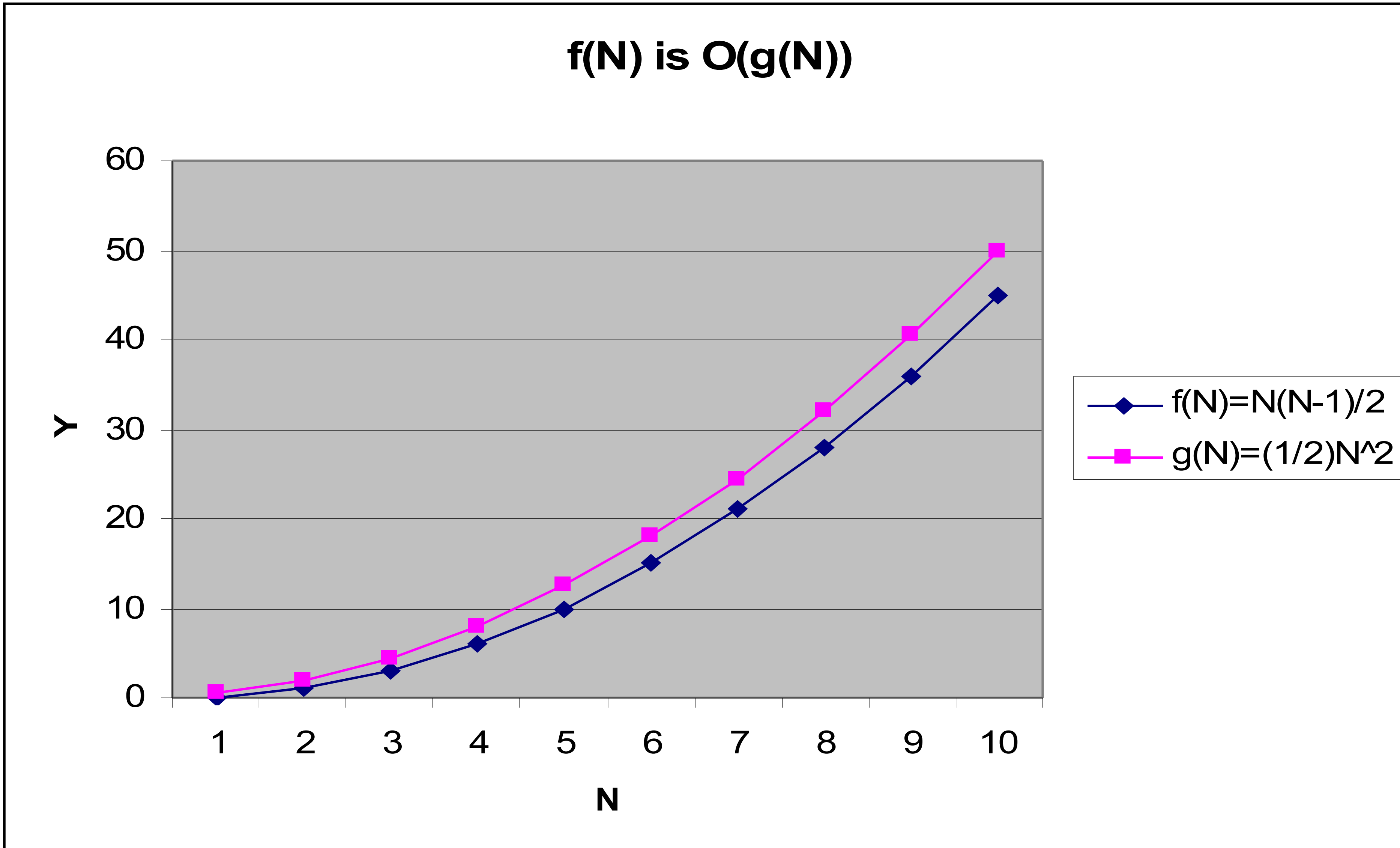
- This means that if the function f represents the number of executions of the above segment, then f(N) = (N-1) + (N-2) + (N-3) + ... + 2 + 1.
- Those who have taken a statistics class or studied summations can see that this equates to f(N) = N(N-1)/2.
- We can see that this function f can be bounded by some polynomial of N^2.

Not so obvious

- What might not be so obvious is that:

  $f(n) <= (1/2)n^2$, for $n >= 1$ and therefore, $n0 = 1$, $g(n) = n^2$, and $c = 1/2$.
- This implies that $f(n)$ is $O(n^2)$.

Graphically



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) $O(n \log_2 n)$