A Simple Example

$$\sum_{i=1}^{n} i^2$$

Time Units to Compute

- 1 for the assignment.
- 1 assignment, $n+1$ tests, and n increments.
- n loops of 3 units for an assignment, an addition, and two multiplications.
- 1 for the return statement.

Total: $1+(1+n+1+n)+3n+1 = 5n+4 = O(n)$

```
int sum (int n)
{
   int partial_sum = 0;
   int i;
   for (i = 1; i <= n; i++)
      partial_sum = partial_sum + (i * i );
   return partial_sum;
}
```

```
   i = 1;
   if (i <= n) {
      partial_sum = partial_sum + (i * i );
      i++; /* i = i + 1 */
   }
```

Analysis too complex

General Rules

- Loops
  - The running time of a "for" loop is at most the running time of the statements inside the "for" loop (including tests) times the number of iterations.

```
for (i = 1; i <= n; i++) {
     sum = sum + i;
}
```

- The above example is *O(n)*.

- Nested loops
  - The total running time of a statement inside a group of nested loops is the running time of the statement multiplied by the product of the sizes of all the loops.

```
for (i = 1; i <= n; i++) {
    for (j = 1; j <= m; j++) {
          sum = sum + i + j;
    }
}
```

$3mn = O(mn)$

- The above example is *O(mn)*.

- Consecutive statements
  - These just add, and the maximum is the one that counts.

```
for (i = 1; i <= n; i++) {
    sum = sum + i;
}
for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
        sum = sum + i + j;
    }
}
```

← O(n)

← $O(n^2)$

- The above example is $O(n^2+n) = O(n^2)$.

A Question:

```
for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
        sum = sum + i + j;
    }
}
sum = sum / n;
for (i = 1; i <= n; i++) {
    sum = sum + i;
}
for (j = 1;  j<= n; j++) {
    sum = sum + j*j;
}
```

- $n^2+1+n+n=O(n^2+2n+1) = O(n^2)$.

- If (test) s1 else s2
  - The running time is never more than the running time of the test plus the larger of the running times of s1 and s2.

```
if (test == 1) {
    for (i = 1; i <= n; i++) {
        sum = sum + i;
    }
}
else for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            sum = sum + i + j;
        }
    }
```

- The running time = 1 + $max(n,n^2) = O(n^2)$.

- Recursion:
  - Analyze from the inside (or deepest part) first and work outwards. If there are function calls, these must be analyzed first. This even works for recursive functions:

```
long factorial (int n) {
   if (n <= 1)
      return 1;
   else
      return n * factorial(n-1);
}
```

Time Units to Compute
---
1 for the test.
1 for the multiplication statement.

What about the function call?

- The running time of *factorial(n) = T(n) = 2+T(n-1) = 4+T(n-2) = 6+T(n-3) = … = 2n = O(n).*

Another Recursive Example

```
long fib (int n) {
   if (n <= 1)
      return 1;
   else
      return fib(n-1) + fib(n-2);
}
```

Time Units to Compute
---
1 for the test.

1 for the addition.
What about the function calls?

- The running time of *fib(n) = T(n) = T(n-1) + T(n-2) + 2*. Can we estimate *T(n)* from this?

Fibonacci Analysis

Let $F(n)$ be the nth Fibonacci number.

We can prove that (1) $T(n) \geq F(n)$ and

(2) $F(n) \geq (3/2)^n$. Thus $T(n) \geq (3/2)^n$,

which means the running time grows

exponentially. This is quite bad.