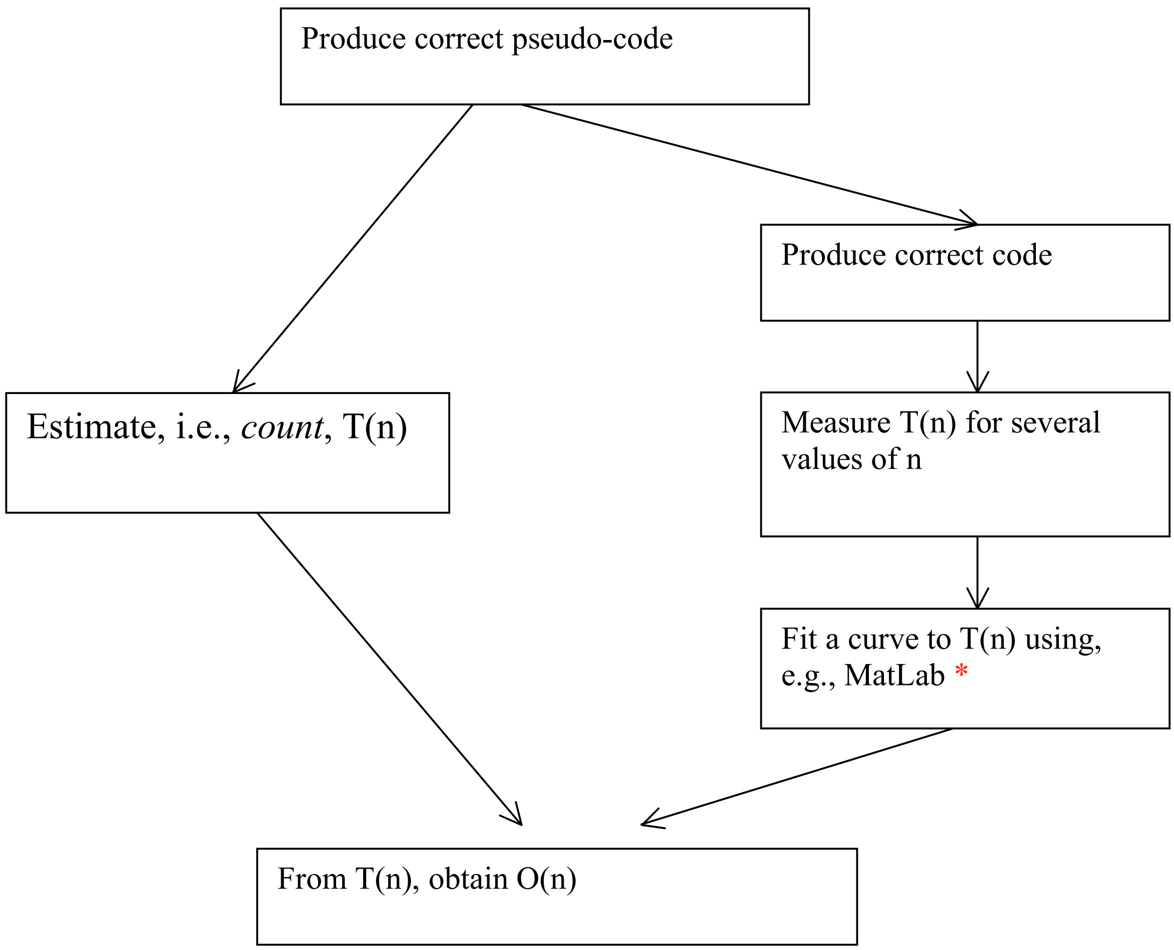
Obtaining Big Oh

Fundamental (engineering) approach: From pseudo-code, *estimate (count)* **or** *measure* running time, then simplify to obtain Big Oh.



* WARNING. Fitted curves must be interpreted with *great care*. *K* values of *n* will produce a perfectly fitted polynomial of order *n-1*. But that perfectly fitted line could be complete wrong for T(n)!

EXAMPLE

Find T(n) to compute $1^2 + 2^2 + 3^2 + \ldots + n^2$

Pseudo code

```
sum = 0;                    // (1)
for (i=1; i<=n; i++)        // (2)
     sum = sum + i^2        // (3)
return sum                  // (4)
```

Estimate each high level operation

-- assign	1	-- memory access	0
-- operation	1	-- call	time to compute parameters
-- test	1	-- return	time to computer result

(1) count is 1

(2) initialize i: 1
test i: n+1
<u>increment i: n</u>
2n + 2

(3) to execute once: 3
(one multiply, one add, one assign)

to execute n time: 3n

Total T(n) = 1 + 2n + 2 + 3n = 3 + 5n

➔ T(n) is O(n)

Rough and ready Big Oh

-- remove all terms except highest order
-- change any constant multiplier to 1

$$T(n) = \underset{\downarrow \atop 0}{\cancel{3}} + \underset{\downarrow \atop 1}{\cancel{5}}n \qquad \text{is } O(n)$$

Redo example faster with that "rough and ready" simplification in mind.

```
sum = 0;                    // (1)
for (i=1; i<=n; i++)        // (2)
     sum = sum + i²         // (3)
return sum                  // (4)
```

(1) count: 1

(2) initialize: 1
i<=n: n + 1, i.e., n
i++: n

So count for loop control is n

(3) to do body once is 3, i.e., 1
to do body n times is 1 * n

➔ T(n) is O(1) + O(n) + O(n) = O(n)

Simplification rules for Big Oh

1. Sequential statement $s_1, s_2, \ldots, s_k$

→ max $(s_1, s_2, \ldots, s_k)$

Example

a = b + c*n;
x[i] = 3 * a * a * a;

→ T(n) = 3 + 4 = 7 is O(1)

→ simplified: max(3, 4) = 4 is O(1)

2. function call, f(a), is the time to execute the function on the parameter a

T(f(a)) = time to computer parameter + time to execute function

Almost always, T(f(a)) is $T_f(n)$ where n is the size of the parameter a.

Example

b[] = shift(a[]); a is array size n

T(n):

assign to b: 1

call shift(): $T_{shift}(n)$

→ $1 + T_{shift}(n)$

3. if/else

```
if (condition)
        s1
else
        s2
```

T(n) = time to calculate condition + max (time for s1, time for s2)

4. loops

count to do body one time * number of times through loop

4-1/2. Nested loops
Computer from inside out.

Example A.

```
for (i=0; i<n; i++)                             //(1)
     for (j=0; j<n; j++)                        //(2)
          if (b[j] <= 0)                        //(3)
               a[j] = 1;                        //(4)
          else {
               a[j] = 3 * b[j];                 //(5)
               b[j] = b[j] - 1;                 //(6)
          }
```

innermost body (5 & 6): max(2, 2) = 2

j loop control (2): $\underbrace{n}_{\#\text{ times}} * \underbrace{2}_{\text{body}} = 2n$

I loop (1): $\underbrace{n}_{\#\text{ times}} * \underbrace{2n}_{\text{body}} = 2n^2$

$T(n) \approx 2n^2$ is $O(n^2)$

5. recursion

count using above rules!

Example

```
fun (x)
     if (x==0) return

     return x * fun (floor(x/2));
```

```
T(n) =    1                    for n = 0
          1 + T(floor(n/2))    for other n
```

T(floor(x)) = 1 (probably, because floor is a simple arithmetic built-in function)

NOTE! Warning!! You cannot say $T_{fun}(n) = 1+1 = 2$ for n!=0 because you are not considering how many times fun() is invoked!

Example B.

```
fun(n)
     if (n <= 0)                    //(1)
          return 1;            //(2)
     else
          return n* fun(n-1);       //(3)
```

$T(n) = 1$ for n=0
$T(n-1) + 1$ for n>=0

Note, using if/else rule, we could say

$T(n) \approx 1 + \max(0, 1+T(n-1))$
$1+ T(n-1)$

But the former method of writing makes it slightly easier to solve the recurrence.

On to more examples!

Example C.

```
sum = 0                                    //(1)
for (i=1; i<=n; i++)                       //(2)
     for (j=1; j<=n^2; j = j*2)            //(3)
          sum++;                           //(4)
```

inner (j) loop:

(4) body: 1

(3) loop control: $\log_2 (n^2-1)$ times

$\log_2(n^2-1) * 1 \approx \log_2(n^2)$

STOP HERE and THINK about this.

$n=1$		$n=2$		$n=4$		$n=8$		$n=16$	
$n^2=1$		$n^2=4$		$n^2=16$		$n^2=64$		$n^2=256$	
J=	1	j=	1	j=	2	j=	1	j=	1
			2		2		2		2
			4		4		4		4
					8		8		8
					16		16		16
							32		32
							64		64
									128
									256
count	1		3		5		7		9
			log(2)=1		log(16)=4		log(64)=6		log(256)=8

→ count ≈ $\log(n^2)+1$

outer (i) loop: # times executed * body

n $\log(n^2)$

→ $T(n) \approx n \log(n^2)$

Example D.

```
sum = 0                               //(1)
for (i=1; i<=n; i++)                  //(2)
     for (j=1; j<=i; j++)             //(3)
          sum++;                      //(4)
```

inner (j) loop: # times loop is executed * time to execute body once.

PROBLEM: The # times j loop is executed depends on the outer (i) loop.

To handle this, we use Sigma notation to represent sequences.

$\sum_{i=s}^{f} (term_i) = term_s + term_{s+1} + term_{s+2} + \ldots + term_{f-1} + term_f$

E.g., $\sum_{i=1}^{n} (i^2) = 1^2 + 2^2 + 3^2 + \ldots + n^2$

$\sum_{i=0}^{n-1} (2^i) = 2^0 + 2^1 + 2^2 + \ldots 2^{n-1}$

NOTE: You must be careful with loop control limits because S notation requires the index increase by 1.

E.g., for (j=0; j<=i; j=j+2)
sum++:

produces j stepping through j=0, 2, 4, 6, … i

In order to accommodate this, do a change of variable: k = j/2:

j = 0, 2, 4, 6, …, i
k = 0, 1, 2, 3, …, i/2

Back to the exercise

Inner j loop: $\sum_{j=0}^{i-1} (1)$

Loop control body

Outer loop: $\sum_{i=0}^{n-1} \left(\sum_{j=0}^{i-1} (1) \right)$

STOP and solve this.

$$\sum_{i=0}^{n-1} \left(\sum_{j=0}^{i-1} (1) \right)$$

$$= \sum_{i=0}^{n-1} (\underbrace{1+1+1+ \ldots +1}_{\text{1 appears i times}})$$

$$= \sum_{i=0}^{n-1} (i)$$

$$= (0 + 1 + 2 + \ldots + n\text{-}1)$$

From high school

$\sum_{i=1}^{n} (i) = 1 + 2 + 3 + \ldots + n = n (n\text{-}1) / 2$

$$\sum_{i=0}^{n-1} (i) = 0 + 1 + 2 + \ldots + n\text{-}1$$

$$= 0 + \sum_{i=1}^{n-1} (i)$$

$$= \sum_{i=1}^{n-1} (i)$$

$$= (n\text{-}1) (n\text{-}2) /2$$

So, $T(n)$ is $O(n^2)$

STOP and do this problem

```
sum = 0;
for (i=0; i<n; i++)
     for (j=0; j < i * i; j++)
          for (k=0; k < j; k++)
               sum++
```