Counting `for` loops

– Why?

```
int main() {
    int total = 0;
    for (int i=1; i < 5; ++i) {
        total += 1;
    }

    cout << total << endl;
}
```

Comparing Algorithms

"Good"

Today – Fast — Average Case / Worst Case

- Correct
- Elegance/Readability/Sustainability
  → no redundant code
- Use of other resources
  → Memory use
- Secure

Interpreting Empirical Data

We can measure…

- particular hardware
- particular input size
- particular input (order, type)
- particular resource(s)
- in a particular language
- particular implentation
- particular compiler
- particular optimization
- particular OS
- particular environment

?s

- readability / extensibility
- how hard to install?

Empirical Data + What?

Theory!

Empirical Data + Theory = Meaning

Decidability - can it be solved at all?

Complexity Class - can it be solved in polynomial time?

Asymptotic Complexity - "Big O"

Approximation - $T(N) \approx 3N^2$

Exact Theory - $T(N) = 3N^2 + 5N + 7.002$

(time) (input size)

today

Guidelines

These "rules" work most of the time:

- Eliminate conditionals
- Start from the inside
- Nested loops → products
- Consecutive loops → sum
- Be careful of loop bounds!
  - → if Σ bounds can't match the for loop variable, do a change of variables

Closed Forms for Common Summations

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

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Closed Forms for Common Summations

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

For example,

$$\sum_{i=1}^{log_2 n} 2^i = 2n - 2$$

Closed Forms for Common Summations

$$\sum_{i=1}^{n} \frac{1}{i} = H(n) \approx \ln n$$