

Counting _{for} loops

– Why?

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

Metric: value of
total
(# of
additions)

Summation:

$$\sum_{i=1}^4 1$$

Closed Form: 4

Comparing Algorithms

Readability

Efficient

→ Run Time

→ Other Resources - memory

Portability

Reliability/Consistency

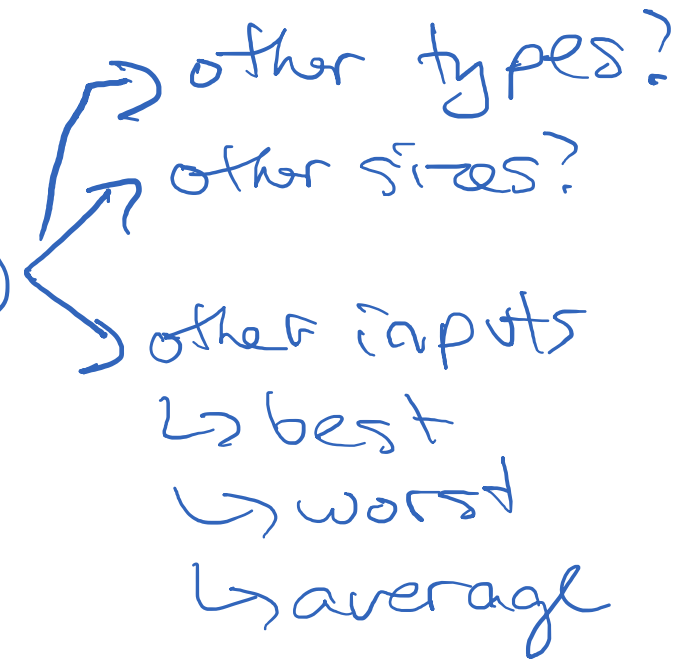
Correctness

Reuse/Maintain/Adaptability

Interpreting Empirical Data

We can measure...

- particular hardware
- particular input (size)
- particular language
- particular compiler
+ (optimization) settings
- OS
- particular environment



Empirical Data + What?

Theory!

Decidability - Can it be solved?

Complexity Class - Can it be solved in polynomial time?

Asymptotic Complexity - Big-O

Approximation - $T(N) \approx 5.5N^2$

Exact Theory - $T(N) = 5.5N^2 + 1.2N + .0042$

↑
Today

CS 670

Guidelines

These “rules” work most of the time:

- eliminate conditionals if possible
- start from inner loop + work out
- nested loops: do product
- consecutive loops: do sum
- be careful w/ loop bounds
 - may need 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

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