Outline

Last time:

- Intro: what this course is about
- Binary search and simple (back-of-the-envelope) performance analysis

Today:

- Operation Count
- FindMax, ComputeRank
Mirror, mirror on the wall, . . . .

Given two algorithms, how do you determine which is more efficient?

Runtime computation: name search

<table>
<thead>
<tr>
<th>Population (size)</th>
<th>Linear</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>EECS 281 (100)</td>
<td>10 ms</td>
<td>0.7 ms (lookup 7 names)</td>
</tr>
<tr>
<td>UM (40000)</td>
<td>4 secs</td>
<td>1.5 ms</td>
</tr>
<tr>
<td>Washtenaw</td>
<td>35 secs</td>
<td>1.8 ms</td>
</tr>
<tr>
<td>MI (9 mil)</td>
<td>15 mins</td>
<td>2.3 ms</td>
</tr>
<tr>
<td>US (290 mil)</td>
<td>8 hours</td>
<td>2.8 ms</td>
</tr>
<tr>
<td>World (6 billion)</td>
<td>7 days</td>
<td>3.3 ms</td>
</tr>
</tbody>
</table>

Assume can search 10 names/ms!
An 8-year old may take longer
Time Complexity: Operation Count

Using *wall-clock time* to measure complexity can be tricky:

- depends on cpu speed
- depends on cpu load(!)
- (but we’ll come back to this later)

Given the uncertainty of wall-clock based measurement, we use instead **operation count** as performance measure:

- pick one operation (ADD, SUB, MUL, DIV, CMP, LOAD, STORE) that is performed *most often*
- count the number of times the operation is performed
Roofing Problem

A builder can carry two shingles on his shoulder as he climbs up the ladder. He then climbs down and carries two more shingles up the ladder. Each round trip (up and down the ladder) costs $2.

Questions:

1. What is the cost to build a tree house (8 shingles)?
2. What is the cost to build a shed (128 shingles)?
3. What is the cost to build a house (2048 shingles)?

Answers:

- 8 shingles: \(2 \times \lceil\frac{8}{2}\rceil = 40\)
- 128 shingles: \(2 \times \lceil\frac{128}{2}\rceil = 128\)
- 2048 shingles: \(2 \times \lceil\frac{2048}{2}\rceil = 2048\)

What is the **ONE** operation we counted in this case?
Operation Count Example: Find Max

Given an array of $N$ elements, find the index of the largest element

Questions:

- What's the fixed cost of the algorithm?
- Which operation should we count to compute the variable cost?
- How many times is this operation executed in the worst case?
What is the Time Complexity of this Function?

void f(int N, int *p) // p an array N integers
{
    int i, j, k, l;

    i = N*2;
    j = i+1;
    k = i-3*400*j;
    l = k/2;

    if (l < 0) {
        i = N+j;
    } else {
        i = log2(k);
    }

    for (i = 0; i < N; i++) {
        j = p[i]*3;
    }

    for (i = 0; i < N; i++) {
        for (k = i; k < N; j++) {
            j = p[j]*3;
        }
    }

    return;
}
Four Operation Count Accounting Rules

Rule 1: Consecutive Statements: \( S_1; S_2; S_3; \ldots; S_N; \)

The runtime \( R \) of a sequence of statements is the runtime of the statement with the max runtime, \( \max(R(S_1), R(S_2), \ldots, R(S_N)) \).

Rule 2: Conditional Execution: if \((S_1) S_2; \) else \( S_3; \)

The runtime of a conditional execution is
\( \max(R(S_1), R(S_2), R(S_3)) \).
Rule 3: Iteration/Loop: for \((S_1; S_2; S_3)S_4;\)

The runtime of a loop with \(n\) iterations is

\[
\max(R(S_1), n \times R(S_2), n \times R(S_3), n \times R(S_4)).
\]

Rule 4: Nested Loop:

for \((S_1; S_2; S_3)\) {
    for \((S_4; S_5; S_6)\) {
        \(S_7;\)
    }
}

Count inside out. The runtime is the max of the runtime of each statement multiplied by the total number of times the statement is executed.
Aside: Sum

What is the sum of all integers from:

- 0 to 4 inclusive?
- 1 to 25 inclusive?
- 3 to 12 inclusive?
Operation Count Example: Compute Rank

Given an unsorted array \( a \) of \( N \) elements, compute an array \( r \), where \( r[i] \) is the rank of \( a[i] \) in \( a \).

The **rank** of an element is defined as the number of all elements in the array that is smaller than it, plus the number of elements *to its left* that is equal to it.

Example: \( a[4, 3, 9, 3, 7] \) gives \( r[2, 0, 4, 1, 3] \)

Time Complexity of \( \text{rank()} \): how many CMPs in total?
Aside: Pseudocode

- high-level description of an algorithm
- more structured than English prose
- less detailed than a program
- hides program design issues
- mathematical formatting allowed

**special notations:**
- ‘==’ for assignment,
- ‘===’ for equality testing

- To ease grading, **you MUST use this syntax whenever pseudocode is asked for in this course**
Operation Count vs. Runtime and Scalability

If $N$ (problem size) is small, longish runtime is tolerable, for large $N$ ($N > 10^4$, or $N \to \infty$ in general), only algorithms with low time complexity are usable.

Scalability: an algorithm/data structure is *not scalable* if its complexity grows so fast that it requires more resources than available for the expected input size

More generally: an algorithm/data structure is not scalable if its complexity grows faster than the rate of increase in input size