CSCI 241: Data Structures

Lecture 2
Runtime Analysis Continued
Quiz time!!!

- On review topics.
- **Not** graded - participation credit only.
- 10 minutes
Announcements

• Course webpage link sent via Canvas.

• Slides will be posted on the webpage after each lecture.

• Lab attendance policy has been refined.

• Action item: make a GitHub account by next week’s lab.
Last Time

• We care how fast things run.

• Trade-offs between operations:
  • FilingCabinet vs PilingCabinet

• Runtime analysis: counting “primitive operations”.
“Primitive” Operations

Things the computer can do in a “fixed” amount of time.

“fixed” - doesn’t depend on the input size \( n \)

A non-exhaustive list:

- **Get** or **set** the value of a variable or array location
- **Evaluate** a simple expression
- **Return** from a method
findMax(A, n):
input: an array A of n integers
output: the maximum value in A

currentMax = a[0]
for i in 1..n:
    if currentMax < A[i]:
        currentMax = A[i]
return currentMax;

\[\text{Number of Operations} = 2 + 2N\]
sillyFindMax(A, n)
input: an array A of n integers
output: the maximum value in A

for i in 0..n:
    isMax = true
    // search for an element bigger than A[i]
    for j in 0..n:
        if A[j] > A[i]:
            isMax = false
    if isMax:
        return A[i]
Input interpretation:

\[
\begin{align*}
\text{plot} & \quad \frac{2 + 2n}{1 + n + n^2} \\
& \quad n = 1 \text{ to } 1000
\end{align*}
\]
Input interpretation:

<table>
<thead>
<tr>
<th>plot</th>
<th>$2 + 2n$</th>
<th>$n = 1$ to $1000$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$1 + n + n^2$</td>
<td></td>
</tr>
</tbody>
</table>

Plot:

- findMin (fast computer)
- findMin
- sillyFindMin
Asymptotic Runtime Complexity

• As the problem size (n) gets large:
  
  the difference **between** complexity classes
dwarf the differences **within** them.

• To go from a count of operations to a big-O class:
  
  • Keep only the fastest-growing term
    
    4 is O(1)
    600 is O(1)
  
  • Drop any constants
    
    n-2 is O(n)
    n^4 + 2n + 4 is O(n^4)
    n! + n^{256} is O(n!)
Big-O, Informally

• “is O(n)” means “is in the same complexity class as n”

• Because constants get ignored, we can often use simple shortcuts:
  
  • Single loop: often O(n)
  
  • Two nested loops: often O(n^2)

  • When loop iteration variable increases as a factor of b: O(f(log_b N))
Really? *any* constant?

A practical argument:

- My MacBook Pro from 2013: 3.17 gigaFLOPs
- Fastest supercomputer as of June 2018: 200 petaFLOPs
- Supercomputer is 63,091,482 times faster.
Input interpretation:

\[
\text{plot } \frac{n^2}{63091482n} \quad n = 0 \text{ to } 1000000000
\]

\[n^2\] algorithm may be faster here!
Really? *any* constant?

A theoretical argument:

Formal definition of big-O:

A function \( f(n) \) is \( O(g(n)) \) IF there exist constants \( c \) and \( N \) such that for all \( n \) larger than \( N \), \( f(n) < c \cdot g(n) \).
O(1)

- Primitive operations:
  - Get or set the value of a variable or array location
  - Evaluate a simple expression
  - Return from a method

- Why are these all O(1)?
Common Complexities

Big-O Complexity Chart

Operations

Elements

O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)

Horrible Bad Fair Good Excellent
Big-O: Example

Alg1(n):

    sum = 0;
    for i = 0..n:
        for j = 1..100:
            sum += i*j
    return sum;

A: O(1)  B: O(n)  C: O(n^2)  D: O(n^3)