Lec 2: Underlying Concepts and Review

Some slides borrowed from CS 240.
Thanks to Anna Lubiw and other previous CS 341 instructors.

- Problem Description
- Models of Computation
- Asymptotic Notation

Problems (terminology)

Recall from CS 240 the terminology used so we can precisely characterize what we mean by efficiency.

Problem: Given a problem instance, carry out a particular computational task. A specification of an infinite set of inputs and corresponding outputs.

Problem Instance: An *Input* for the specified problem.

Problem Solution: The *Output* (correct answer) for the specified problem instance.

Size of a problem instance: $Size(I)$ is a positive integer which is a measure of the size of the instance I.

Analysis: Measure the *Time* and *Space* used by the algorithm as a function of the input size.

Algorithms and Programs

Algorithm: An algorithm is a *step-by-step process* (e.g., described in pseudo-code) for carrying out a series of computations, given an arbitrary problem instance I.

Solving a problem: An Algorithm A *solves* a problem Π if, for every instance I of Π, A finds (computes) a valid solution for the instance I in finite time.

Program: A program is an *implementation* of an algorithm using a specified computer language.

In this course, our emphasis is on algorithms (as opposed to programs or programming).

Models of Computation

We often use **Pseudocode:** similar to a typical programming language but **intended for a human to read** - uses common programming structures and conventions but omits machine and language specific details.

In contrast, a program is a method of communicating an algorithm to a computer.

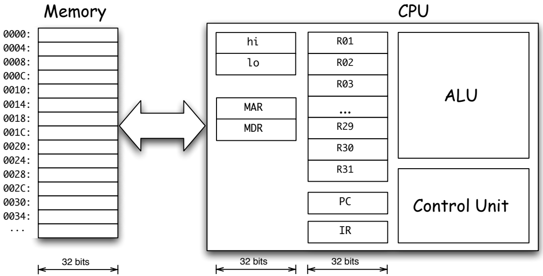
Pseudocode

- omits obvious details, e.g. variable declarations,
- has limited if any error detection,
- sometimes uses English descriptions,
- sometimes uses mathematical notation.

Size of an integer? Cost of elementary operations?

Random Access Machine

- Abstracts Assembly language
- "Random access" can access memory location i in one step



Size of a memory location? A good compromise:

- Word RAM - each memory location holds one word. Assume number of bits in word is $\Theta(\log n)$ where n is the input size.
- E.g. Given array $A[1..n]$, an index $i \in [1..n]$ fits in a word.

Other Models of Computation

Circuit Family

- Abstracts hardware circuitry

Turing Machine

- Abstract human computer working with pencil and paper.
- Has a read/write head and infinite tape of cells (move left/right 1 cell at a time) - specific cell access may not be 1 step; time to access memory location i is proportional to i.

Special purpose or **Structured** models of computing

- Comparison-based model for sorting: $\Omega(n \log n)$ lower bound

Runtime of an Algorithm

Runtime depends on input $\Rightarrow$ express runtime as a function of input size.

- Model of Computation specifies how to count input size.
- We expect the runtime to increase as input size increases.

Let $T_{\mathcal{A}}(I)$ denote the running time of an algorithm $\mathcal{A}$ on instance I.

For a given size n, there are various inputs.
How do we combine runtimes to a single number?

Runtime of an Algorithm

Runtime depends on input $\Rightarrow$ express runtime as a function of input size.

- Model of Computation specifies how to count input size.
- We expect the runtime to increase as input size increases.

Let $T_{\mathcal{A}}(I)$ denote the running time of an algorithm $\mathcal{A}$ on instance I.

For a given size n, there are various inputs.
How do we combine runtimes to a single number?

Worst-case complexity of an algorithm: take the worst I

- $T_{\mathcal{A}}(n) = \max\{T_{\mathcal{A}}(I)$ where I is an input of size $n\}$
- Often simply say $T(n)$ (or $T(I)$) if $\mathcal{A}$ is understood.

Average-case complexity of an algorithm: average over I

- Often difficult to analyze, depends on input distribution, etc

Asymptotic Analysis

- Want simple functions: e.g. $n \log n$, n^2, etc
- Machine independent so ignore coefficients (multiplicative factors) and lower order terms.

Often use Big-Oh, an upper bound.
Want the tightest bound.

- $f(n) = 7n^2 + 13n + 27 \in O(?)$
- $10^{100} n \in O(?)$
- $2^{n+1} \in O(?)$
- $(n+1)! \in O(?)$

Order Notation Summary

O-notation: $f(n) \in O(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $|f(n)| \leq c\,|g(n)|$ for all $n \geq n_0$.

Ω-notation: $f(n) \in \Omega(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $c\,|g(n)| \leq |f(n)|$ for all $n \geq n_0$.

Θ-notation: $f(n) \in \Theta(g(n))$ if there exist constants $c_1, c_2 > 0$ and $n_0 \geq 0$ such that $c_1\,|g(n)| \leq |f(n)| \leq c_2\,|g(n)|$ for all $n \geq n_0$.

o-notation: $f(n) \in o(g(n))$ if for all constants $c > 0$, there exists a constant $n_0 \geq 0$ such that $|f(n)| \leq c\,|g(n)|$ for all $n \geq n_0$.

ω-notation: $f(n) \in \omega(g(n))$ if for all constants $c > 0$, there exists a constant $n_0 \geq 0$ such that $c\,|g(n)| \leq |f(n)|$ for all $n \geq n_0$.

Algebra of Order Notations

Identity rule: $f(n) \in \Theta(f(n))$

Transitivity:

- If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$ then $f(n) \in O(h(n))$.
- If $f(n) \in \Omega(g(n))$ and $g(n) \in \Omega(h(n))$ then $f(n) \in \Omega(h(n))$.

Maximum rules: Suppose that $f(n) > 0$ and $g(n) > 0$ for all $n \geq n_0$. Then:

- $f(n) + g(n) \in O(\max\{f(n), g(n)\})$
- $f(n) + g(n) \in \Omega(\max\{f(n), g(n)\})$

Proof: $\max\{f(n), g(n)\} \leq f(n) + g(n) \leq 2\max\{f(n), g(n)\}$

Runtime Examples

- $O(1)$ - counting a finite number of things
- $O(\log n)$ - binary search
- $O(n)$ - find max
- $O(n \log n)$ - sorting
- $O(n^2)$ - insertion sort
- $O(n^3)$ - multiplying two $n \times n$ matrices
- $O(2^n)$ - try all subsets
- $O(n!)$ - try all orderings of a set; e.g. Travelling Salesman
- Also, $\sqrt{n}, \log \log n, (\log n)^2$, etc

Multiple Variables

Output sensitive: Jarvis March $O(n * h)$ where h is the number of edges in convex hull; i.e. h is a measure of the output.

Multiple variables: Graph $G = (V, E)$ where $|V| = n$ and $|E| = m$. $m \in O(n^2)$ but in some instances $(n + m)$ may have a tighter Θ-bound.

***O*-notation:** $f(n, m) \in O(g(n, m))$ if there exist constants $c > 0$ and $n_0, m_0 \geq 0$ such that $|f(n, m)| \leq c\,|g(n, m)|$ for all $n \geq n_0, m \geq m_0$.

Reductions

- Simply put: Using a known algorithm to solve a new problem.

Problem

2-SUM

Instance: *Array $A[1 \ldots n]$ of numbers and taget number m*

Find: *i, j s.t. $A[i] + A[j] = m$ (if they exist)*

Note: Its sometimes simpler to index arrays from 1 to n.

```
Algorithm1(A, n, m)
    for i ← 1 to n do
        for j ← 1 to n do
            if A[i] + A[j] = m then
                return FOUND
    return FAIL
```

Reductions continued

Algorithm 2: Use algorithms Sort and BinarySearch.

```
Algorithm2(A, n, m)
    Sort A
    for i ← 1 to n do
        j ← BinarySearch(A, m − A[i])
        if A[i] + A[j] = m then
            return FOUND
    return FAIL
```

Runtime: $O(n \log n) + O(n \log n) \in O(n \log n)$

Reductions continued

Algorithm 3: Improve the 2nd phase.

```
Algorithm3(A, n, m)
    Sort A
    i, j ← 1, n
    while i ≤ j do
        sum ← A[i] + A[j]
        if sum > m then
            j ← j − 1
        elseif sum < m then
            i ← i + 1
        else
            return FOUND
    return FAIL
```

Runtime: $O(n \log n) + O(n) \in O(n \log n)$ but $O(n)$ after sorting.

Correctness Invariant: if a solution exists: $i^* \leq j^*$ then $i^* \geq i, j^* \leq j$

Reductions continued

Problem

3-SUM

Instance: *Array $A[1 \ldots n]$ of numbers and taget number m*

Find: i, j, k *s.t.* $A[i] + A[j] + A[k] = m$ *(if they exist)*

Reduce 3-SUM to 2-SUM.

Reductions continued

Problem

3-SUM

Instance: *Array $A[1 \ldots n]$ of numbers and taget number m*

Find: *i, j, k s.t. $A[i] + A[j] + A[k] = m$ (if they exist)*

Reduce 3-SUM to 2-SUM.

Note: $A[i] + A[j] + A[k] = m$ so $A[i] + A[j] = m - A[k]$

Algorithm

- Run 2-SUM with target $m - A[k]$ for $k = 1, \ldots, n$.

Runtime: $O(n \cdot n \log n) \in O(n^2 \log n)$?

Reductions continued

Problem

3-SUM

Instance: *Array $A[1 \ldots n]$ of numbers and taget number m*

Find: *i, j, k s.t. $A[i] + A[j] + A[k] = m$ (if they exist)*

Reduce 3-SUM to 2-SUM.
Note: $A[i] + A[j] + A[k] = m$ so $A[i] + A[j] = m - A[k]$

Algorithm

- Run 2-SUM with target $m - A[k]$ for $k = 1, \ldots, n$.

Runtime: $O(n \cdot n \log n) \in O(n^2 \log n)$?

Don't need to sort over and over - use Alg 3 but only sort once.
Runtime: $O(n \log n) + O(n^2) \in O(n^2)$
Faster Algorithms?