Lecture 7: Algorithms I

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
Scenario 1: Amazon buying Adventure

- Early April 2011: A scientist at UC-Berkeley logged on to Amazon.com to buy an extra book for his lab.
- He usually pays $35-$40 per copy
- But on that day, he found 2 used copies, one priced at $1,730,045 the other at $2,198,177!!
- He thought it was just a mistake or a joke
- He re-checked the following day and the prices were $2,194,443 and $2,788,233!!
- The escalation continued for two weeks with the price peaking on April 18th at $28,698,655 (+ $3.99 shipping)!!
Scenario 2: Flash Crash (one of several)

- Early on May 6, 2010: stock market was hit by unsettling developments in Greece.... BUT
- At 2:42pm (EST) markets start dropping into a free fall
- At 2:47pm (i.e. 300 seconds later): Dow Jones was down 998.5 points (the largest single day drop in history!)
- Nearly $1 Trillion of wealth fell into the electronic ether!!!
- Some share prices crashed to one penny ($0.01) rendering billion-dollar companies worthless!
- Dow Jones recovered 500 points in less than 3 minutes!!
What Happened??

• Scenario 1:
  – Algorithms used by Amazon to price books got into price war!
  – One of the seller’s algorithms was programmed to price the book slightly higher than the competitor’s price.
  – The second algorithm, in turn, increased its price!
  – Things didn’t turn to normal until a human being stepped in and overrode the system.

• Scenario 2:
  – We don’t know till today!!
  – Explanation 1: Some of the blame was directed to a city money manager whose algorithm sold $4B worth of stock too quickly.
  – Explanation 2: Group of traders who conspired to send things down all at once through a coordinated algorithms
As we put more and more of our world under the control of algorithms, we can lose track of who – or what – is pulling the strings.

from Christopher Steiner’s book “Automate This: How Algorithms Came to Rule our World” .... (from which I got the previous two scenarios too!).
**Algorithms**

Well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.

A tool for solving a well-specified computational problem.

The statement of the problem specifies in general terms the desired input/output relationship.
Problem Statement

• Problem specifications have two parts:
  1. the set of allowed input instances,
  2. the required properties of the algorithm’s output
Questions

• What is the difference between a program and an algorithm?
• Is error handling part of the algorithm? or the HLL program?
• Does your algorithm need to produce just a correct result? or always the best result?
• If computers were infinitely fast and memory was free, would you have any reasons to study algorithms?
Can We Solve Anything With a Computer?

• **Undecidable**
  – Cannot be solved by an algorithm
  – *e.g.* Halting problem (given a program and inputs for it, decide whether it will run forever or will eventually halt.)

• **Unsolvable**
  – No finite algorithm
  – *e.g.* Goldbach’s conjecture (Every even number greater than 2 can be written as the sum of two primes.)

• **Intractable**
  – Unreasonable amount of time and resources
“Steps” of an algorithm

• Finite number
• Unambiguous
• very specific
• can be carried out in a finite amount of time in a deterministic way
• Since we can only input, store, process & output data on a computer, the instructions in our algorithms will be limited to these functions
Algorithm Properties

• It must be correct.
• It must be composed of a series of concrete steps.
• There can be no ambiguity as to which step will be performed next.
• It must be composed of a finite number of steps.
• It must terminate.
Algorithm Is Different Than A HLL Program

- In algorithms you do not need to use strict syntax
- You can present an algorithm in pseudo-code, flowchart, ...
- Pseudocode is not concerned with issues of software engineering (e.g. error handling, abstraction, modularity, ...).
Pseudocode Algorithm

• **Example:** Write an algorithm to determine a student’s final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks.
Pseudocode Algorithm

Pseudocode:

- Input a set of 4 marks
- Calculate their average by summing and dividing by 4
- if average is below 50
  - Print “FAIL”
else
  - Print “PASS”
Pseudocode Algorithm

• Detailed Algorithm

Step 1: Input M1,M2,M3,M4
Step 2: GRADE ← (M1+M2+M3+M4)/4
Step 3: if (GRADE < 50) then
      Print “FAIL”
    else
      Print “PASS”
endif
A Flowchart

- shows logic of an algorithm
- emphasizes individual steps and their interconnections
- e.g. control flow from one action to the next
## Flowchart Symbols

### Basic

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Use in Flowchart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oval</td>
<td><img src="image" alt="Oval" /></td>
<td>Denotes the beginning or end of the program</td>
</tr>
<tr>
<td>Parallelogram</td>
<td><img src="image" alt="Parallelogram" /></td>
<td>Denotes an input operation</td>
</tr>
<tr>
<td>Rectangle</td>
<td><img src="image" alt="Rectangle" /></td>
<td>Denotes a process to be carried out e.g. addition, subtraction, division etc.</td>
</tr>
<tr>
<td>Diamond</td>
<td><img src="image" alt="Diamond" /></td>
<td>Denotes a decision (or branch) to be made. The program should continue along one of two routes. (e.g. IF/THEN/ELSE)</td>
</tr>
<tr>
<td>Hybrid</td>
<td><img src="image" alt="Hybrid" /></td>
<td>Denotes an output operation</td>
</tr>
<tr>
<td>Flow line</td>
<td><img src="image" alt="Flow line" /></td>
<td>Denotes the direction of logic flow in the program</td>
</tr>
</tbody>
</table>
Example

Step 1: Input M1, M2, M3, M4
Step 2: GRADE ← (M1 + M2 + M3 + M4) / 4
Step 3: if (GRADE < 50) then
    Print “FAIL”
else
    Print “PASS”
endif
Example

Problem: Robot Tour Optimization
Input: A set $S$ of $n$ points in the plane.
Output: What is the shortest cycle tour that visits each point in the set $S$?
Example

NearestNeighbor(P)

Pick and visit an initial point $p_0$ from $P$
$p = p_0$
i = 0
While there are still unvisited points
\[ i = i + 1 \]
Select $p_i$ to be the closest unvisited point to $p_{i-1}$
Visit $p_i$
Return to $p_0$ from $p_{n-1}$

The above algorithm is:
• Simple to understand and implement
• Makes sense

And ... WRONG! Does not produce the shortest path!
Example

NearestNeighbor($P$)

Pick and visit an initial point $p_0$ from $P$

$p = p_0$

$i = 0$

While there are still unvisited points

\[
i = i + 1
\]

Select $p_i$ to be the closest unvisited point to $p_{i-1}$

Visit $p_i$

Return to $p_0$ from $p_{n-1}$

This is what the above alg. produces

This is the optimal solution.
Example

ClosestPair(P)

Let $n$ be the number of points in set $P$.
For $i = 1$ to $n - 1$ do
  
  $d = \infty$
  
  For each pair of endpoints $(s, t)$ from distinct vertex chains
  
  if $dist(s, t) \leq d$ then $s_m = s$, $t_m = t$, and $d = dist(s, t)$

  Connect $(s_m, t_m)$ by an edge
  
  Connect the two endpoints by an edge

This one will produce the optimal solution of the previous example.
Hmmm ...

• Looks like for this problem any algorithm can produce a very bad result for some inputs 😞

• This example we just saw is a classical problem called The Traveling Salesman Problem (TSP)
Traveling Salesman Problem

The traveling salesman must travel to $n$ different towns in his area each month in order to deliver something important. Each town is a different distance away from his town and from each other town. How do you figure out a route that will minimize the distance traveled?
Brute Force?

- Enumerate all possible routes
  - For 10 towns for example there are 10! (3,628,800)
- Choose the shortest.
- This is called brute force algorithm.

OptimalTSP(P)

\[ d = \infty \]

For each of the \( n! \) permutations \( P_i \) of point set \( P \)

If \( (cost(P_i) \leq d) \) then \( d = cost(P_i) \) and \( P_{min} = P_i \)

Return \( P_{min} \)
Is Brute Force a Good Solution?

Take Home Lesson: There is a fundamental difference between *algorithms*, which always produce a correct result, and *heuristics*, which may usually do a good job but without providing any guarantee.
How Do We Judge Algorithms?

- Correctness
- Efficiency
  - Speed
  - Memory
- Algorithm analysis is predicting the resources that the algorithm requires
- Algorithms can be understood and studied in a language and machine-independent manner.
Machine Model $\rightarrow$ RAM

- Random Access Machine
- Instructions are executed one after the other.
- Basic instructions (arithmetic, logic, data movement) take fixed amount of time
- Memory is infinite
- We need a way that summarizes the behavior of an algorithm executed on RAM
Worst-/average-/best-case

- **Worst-case running time of an algorithm**
  - The longest running time for any input of size \( n \)
  - An upper bound on the running time for any input
    - ⇒ guarantee that the algorithm will never take longer
  - Example: Sort a set of numbers in increasing order; and the data is in decreasing order
  - The worst case can occur fairly often
    - E.g. in searching a database for a particular piece of information
- **Best-case running time**
  - sort a set of numbers in increasing order; and the data is already in increasing order
- **Average-case running time**
  - May be difficult to define what “average” means
The Big Oh Notation

• A way of giving an approximation of the amount of computation done by an algorithm given the input size

• ignores the difference between multiplicative constants: \( f(n) = 2n \) and \( g(n) = n \) are identical in Big Oh analysis
The Big Oh Notation

- \( f(n) = O(g(n)) \) means \( c \cdot g(n) \) is an upper bound on \( f(n) \). Thus there exists some constant \( c \) such that \( f(n) \) is always \( \leq c \cdot g(n) \), for large enough \( n \) (i.e., \( n \geq n_0 \) for some constant \( n_0 \)).

- \( f(n) = \Omega(g(n)) \) means \( c \cdot g(n) \) is a lower bound on \( f(n) \). Thus there exists some constant \( c \) such that \( f(n) \) is always \( \geq c \cdot g(n) \), for all \( n \geq n_0 \).

- \( f(n) = \Theta(g(n)) \) means \( c_1 \cdot g(n) \) is an upper bound on \( f(n) \) and \( c_2 \cdot g(n) \) is a lower bound on \( f(n) \), for all \( n \geq n_0 \). Thus there exist constants \( c_1 \) and \( c_2 \) such that \( f(n) \leq c_1 \cdot g(n) \) and \( f(n) \geq c_2 \cdot g(n) \).
The Big Oh Notation

Problem: Is $2^{n+1} = \Theta(2^n)$?
Example

- \( f(n) = 2n + 5 \)
  \( g(n) = n \)
- Consider the condition
  \( 2n + 5 \leq n \)
will this condition ever hold? No!
- How about if we multiply a constant by \( n \)?
  \( 2n + 5 \leq 3n \)
the condition holds for values of \( n \) greater than or equal to 5
- This means we can select \( c = 3 \) and \( n_0 = 5 \) and
  \( f(n) \rightarrow O(n) \)
Example (cont'd)

2n+5 is $O(n)$
Is It Wise to Ignore Constants?

• If two algorithms one is \(O(n^2)\) and the other \(O(\log n)\)
  – one is \(C_1n^2\) and the other \(C_2\log n\)
  – What if \(C_2\) is much bigger than \(C_1\)?

<table>
<thead>
<tr>
<th>(n)</th>
<th>(f(n))</th>
<th>(\log n)</th>
<th>(n)</th>
<th>(n\log n)</th>
<th>(n^2)</th>
<th>(2^n)</th>
<th>(n!)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.003 μs</td>
<td>0.01 μs</td>
<td>0.033 μs</td>
<td>0.1 μs</td>
<td>1 μs</td>
<td>3.63 ms</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.004 μs</td>
<td>0.02 μs</td>
<td>0.08 μs</td>
<td>0.4 μs</td>
<td>1 ms</td>
<td>77.1 years</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0.005 μs</td>
<td>0.03 μs</td>
<td>0.147 μs</td>
<td>0.9 μs</td>
<td>1 sec</td>
<td>8.4 (\times 10^{15}) yrs</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>0.005 μs</td>
<td>0.04 μs</td>
<td>0.213 μs</td>
<td>1.6 μs</td>
<td>18.3 min</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>0.006 μs</td>
<td>0.05 μs</td>
<td>0.282 μs</td>
<td>2.5 μs</td>
<td>13 days</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(n)</th>
<th>(f(n))</th>
<th>(\log n)</th>
<th>(n)</th>
<th>(n\log n)</th>
<th>(n^2)</th>
<th>(2^n)</th>
<th>(n!)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.007 μs</td>
<td>0.1 μs</td>
<td>0.644 μs</td>
<td>10 μs</td>
<td>4 (\times 10^{13}) yrs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000</td>
<td>0.010 μs</td>
<td>1.00 μs</td>
<td>9.966 μs</td>
<td>1 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td>0.013 μs</td>
<td>10 μs</td>
<td>130 μs</td>
<td>100 ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100,000</td>
<td>0.017 μs</td>
<td>0.10 ms</td>
<td>1.67 ms</td>
<td>10 sec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.020 μs</td>
<td>1 ms</td>
<td>19.93 ms</td>
<td>16.7 min</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10,000,000</td>
<td>0.023 μs</td>
<td>0.01 sec</td>
<td>0.23 sec</td>
<td>1.16 days</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100,000,000</td>
<td>0.027 μs</td>
<td>0.10 sec</td>
<td>2.66 sec</td>
<td>115.7 days</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1,000,000,000</td>
<td>0.030 μs</td>
<td>1 sec</td>
<td>29.90 sec</td>
<td>31.7 years</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Big Oh examples

- \( N^2 / 2 - 3N = O(N^2) \)
- \( 1 + 4N = O(N) \)
- \( 7N^2 + 10N + 3 = O(N^2) = O(N^3) \)
- \( \log_{10} N = \log_2 N / \log_2 10 = O(\log_2 N) = O(\log N) \)
- \( \sin N = O(1); \; 10 = O(1), \; 10^{10} = O(1) \)
- \( \log N + N = O(N) \)
- \( N = O(2^N), \; \text{but} \; 2^N \text{ is not } O(N) \)
Example

- Calculate \(\sum_{i=1}^{N} i^3\)

```c
int sum(int n)
{
    int partialSum;
    partialSum=0;
    for (int i=1;i<=n;i++)
    {
        partialSum += i*i*i;
    }
    return partialSum;
}
```

- Lines 1 and 4 count for one unit each
- Line 3: executed N times, each time four units
- Line 2: (1 for initialization, N+1 for all the tests, N for all the increments) total 2N + 2
- total cost: \(6N + 4 \Rightarrow O(N)\)
Good Book
Good Book: For Fun!
Conclusions

• In this lecture we defined what an algorithm is in simple terms.
• Big Oh notation is a convenient way to compare algorithms.
• Sometimes the best solution may not be needed and a good-enough solution is just fine.