# Correctness and Efficiency

Debswapna Bhattacharya, PhD
Assistant Professor
Computer Science and Software Engineering
Auburn University

# Correctness

- An algorithm is said to be correct only if for **each and every** problem instance, it halts with the correct output.
- A correct algorithm solves (all instances of) the given computational problem.
- An incorrect algorithm might not halt at all on some problem instances, or it might halt with something other than the correct answer.

# Why is correctness important?

In August 2012 a trading algorithm lost money at the rate if 10+ million dollars per minute and humans did not notice until more than half an hour had passed.

In 2015, users found the photo-sharing site Flickr's new image-recognition technology was labeling dark-skinned people as "apes" and auto-tagging photos of Nazi concentration camps as "jungle gym" and "sport."


A Boeing 787 has 6.5 million lines behind its avionics and online support systems. What is at stake if the algorithms upon which this software is based have flaws?

# Efficiency

- **Work:** How many basic computations (operations) an algorithm has to execute to solve a specific problem instance, i.e., when it is given an input of a certain size. Work increases as input size increases.

- **Time** : How long the algorithm takes to solve a specific problem instance, i.e., when it is given an input of a certain size. Time increases as input size increases.

- **Space** : How many memory locations are used by the algorithm to solve a specific problem instance, i.e., when it is given an input of a certain size. Space needed increases as input size increases.

- **Other**: I/O bottleneck, energy consumption, thermal efficiency, etc.

# Efficiency

- Time and other efficiencies are directly correlated with work efficiency. Why?
- So we will primarily look at work efficiency in this course, and also discuss space efficiency when relevant.
- Since time and work are so closely related, we will use the term *time efficiency* for the number of operations an algorithm executes to solve an input problem instance of a particular size.
- Time efficiency is denoted by the function T(n) which provides the number of operations an algorithm executes to solve an input problem instance of size n.

# Why is efficiency important?

- Inefficient algorithms take too long (or too much memory) to compute solutions to large problem instances, so they are impractical.

Let us look at an example

# Mirror…mirror on the wall, where're the biggest trades of'em all?

Suppose you are hired by Goldman Sachs for a well-paid wall street job.

Suppose on your first day your boss shows up at your office and tells you: "**write a program to find out the 10 largest stock trades that occurred in the NY Stock Exchange each day**."

Goldman Sachs is a tough place to work…lots of competitive folks with Ivy League degrees…if you don't deliver, you can kiss your job goodbye!

# The Problem

How many trades occur in NYSE each day?

2800 companies trading 1.46 billion shares each day. Trades bundle a bunch of shares. Let us say on average 146 shares per trade. So you are looking at a volume of about 10 million trades.

All you have to do is to find the values of the ten largest trades. Assume that no trade has exactly the same dollar and cents value as another, i.e., no duplicates…

# The Problem…contd

Suppose you can download the trade value data to your computer and say, store it in an array.

How will you solve the problem if you had to deal with 10 million data items one day? What is your program going to do?

How quickly will your program produce the answer?

Discussion of various computational strategies to solve this problem

# The Problem…contd

Suppose you decide to sort the array using Bubble sort.

How many steps will your algorithm have to execute?

Bubblesort is a quadratic ($O(n^2)$) algorithm. What does this mean?

It will carry out approximately $n^2 = 10^{14}$ operations to sort the array

Then in ten operations the program can print out the top ten trades.

# The Problem…contd

Say you have a really fast machine running at 4 GHz, i.e., $4 * 10^9$ clock cycles per second.

A machine typically requires about 200 clock cycles to execute one computation step.

So your computer can execute $2 * 10^7$ steps in each second.

How long will it take to do just the sorting (execute $10^{14}$ steps)?

Approximately $5 * 10^6$ seconds.

# The Problem…contd

How long is $5 * 10^6$ seconds?

There are $8.6 * 10^4$ seconds in a day

**58 days…**so you'll have to tell your boss: "come back after two months!"

You better head over to the unemployment benefits office!

# The Problem…contd

What if you used a different sorting algorithm – Merge Sort which has O(nlogn) complexity?

How many steps are needed to sort 10 million items using Merge Sort?

$24 * 10^7$ steps

How long will it take to execute $24 * 10^7$ steps?

About 12 seconds!

# Other strategies

Create an array of 10 trade values and read in the first ten trades, sort them, then process each of the remaining $10^7$-10 trades as follows: If the next trade's values is less than the smallest trade already in the array, discard it; otherwise swap it with the smallest trade in the array and re-sort it.

How long will this take?

# Other strategies

Yet another strategy is to iterate through the array of trades 10 times, each time finding the largest number and moving it to one end of the array.

How long will this take?

# Selection Problem

This example is actually an instance of a general problem called the Selection Problem:

Given n items and a way to compare any two items (to decide if one is >, = or < the other), find the k-th largest or smallest item, $1 \leq k \leq n$

How fast do you think you can solve this problem?

# Selection Problem

Fastest known algorithm (called a median-of-median algorithm) is O(n).

At most how many steps are needed to find the $10^{th}$ largest number in 10 million numbers using this algorithm? $10^7$ steps

How long will it take to execute $10^7$ steps?

Recall that your computer can execute $2 * 10^7$ steps in each second half-a-second!

You can repeat 9 more times to find the $9^{th}$...$1^{st}$ largest numbers spending 0.5*10=5 seconds in total or after finding the $10^{th}$ largest number scan through the entire array once again to locate all numbers that are smaller, which will give you the answer in 1 second!

# So, your choices are…

Sort data using an $n^2$ sorting algorithm and ask your boss to come back after two months…lose your job

Sort data using the most efficient strategy and corresponding algorithm and find the answer in a second or less.

This is why efficiency is such a critical measure of algorithms.

# Other strategies

If we maintained an array of 10 trade values on an ongoing basis starting at the beginning of the day, and processed each trade as it came in instead of waiting till all trades were in, even though the total processing time will still be about 15 seconds, the program will need to run continuously throughout the day. On the other hand, when the last trade is in at 5:00 pm, the program will finish a few microseconds later.

Another interesting characteristic of this approach is that at any time during the day, the algorithm will tell you the top ten trades from among the trades it has seen until then. Algorithms that process their inputs one at a time and can give you the correct answer at any time based on the data they have seen so far, are called **online** or **anytime** algorithms.

# So…

- **Time efficiency** is usually the most important criterion. But there may be other criteria to consider as well…
  - **Online or anytime** algorithms: if an answer is needed at any time based on partial input
  - **In-place** algorithms: if memory usage is a concern
  - **Generality**: if the algorithm needs to work correctly for a more general problem specification