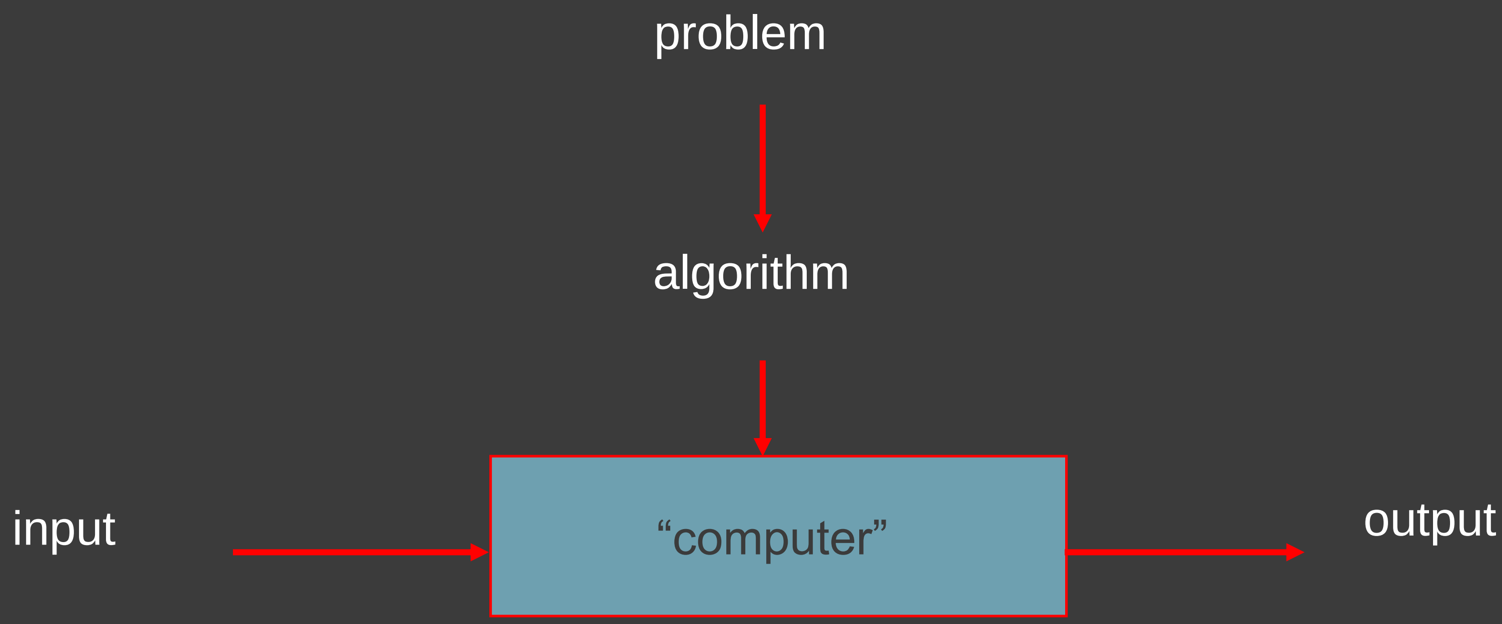
ICSI 403
Design and Analysis of Algorithms

Lecture 04b – Chapter 1, The Role of Algorithms in Computing

J Marques de Carvalho

What is an algorithm?



Algorithms

- *Algorithm*: A well-defined computational procedure that takes some set of input values and produces some set of output values.
  - i.e., the algorithm is the sequence of steps that transforms the input into the output.
- Algorithms are used to solve well-specified computational problems.
  - Problems specify, in general terms, the desired relationship between the input and the output.

The Sorting Problem

- The Sorting Problem occurs frequently in computing, and is a good place to start our discussion:
- Input: A sequence of *n* numbers $\langle a_1, a_2, \ldots, a_n \rangle$
- Output: A permutation (reordering) $\langle a_1', a_2', \ldots, a_n' \rangle$ of the input sequence such that $a_1' \le a_2' \le \ldots \le a_n'$
- Don't confuse the problem itself with an instance of the problem.
  - An instance of the sorting problem might be to transform the input sequence $\langle 31, 41, 59, 26, 41, 58 \rangle$ into the output sequence $\langle 26, 31, 41, 41, 58, 59 \rangle$

The Sorting Problem

- There may be multiple algorithms that solve a given problem.
- Which one we choose might depend on any of a variety of factors.
- In the case of the sorting problem, factors might include:
  - The number of items we need to sort.
  - How (relatively) sorted the items are believed to be.
  - Sorting medium (RAM, disk, tape).
  - Computer’s Architecture.

Algorithm Correctness

- An algorithm is *correct* if, for every input instance, it completes with the correct output.
- A correct algorithm *solves* the problem.
- Incorrect algorithms might not complete, might complete, but with the wrong output, or might complete with the correct output only for some input instances.

Examples of Practical Algorithms

- Selection Sort
- Binary Search

Data Structures

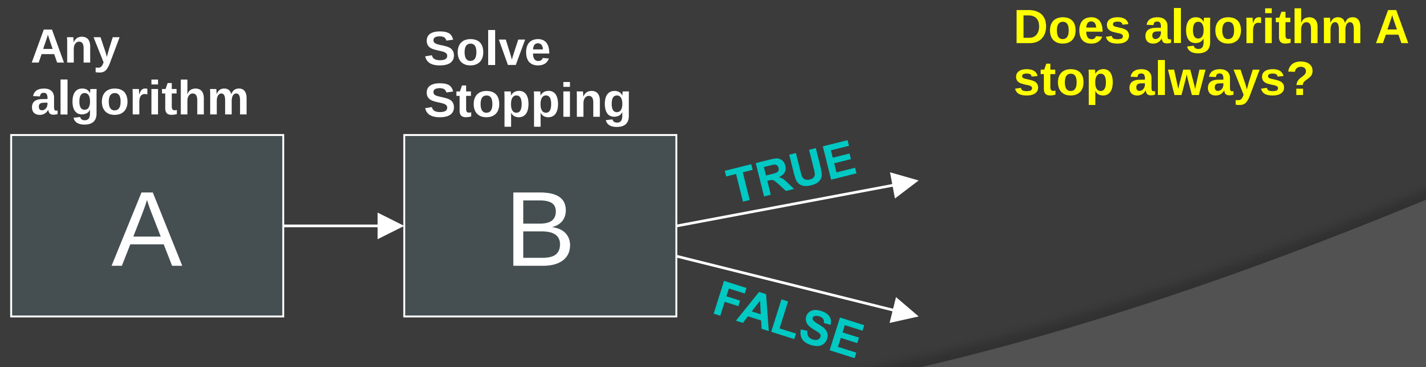
- This class is the successor to ICSI 213 (Data Structures).
- As you learned in ICSI 213, stacks, queues, and general lists all have different strengths and limitations.
  - Which one you use depends on the problem.
- This semester will expose you to trees and graphs, and the algorithms to manipulate them.

Algorithms

- We will be examining problems with well-known algorithms
- Developing new algorithms is its own branch of theoretical computer science
- The algorithms presented in the Cormen text can be used as “tools in your toolbelt”, or may inspire you to develop new tools (algorithms) in the future

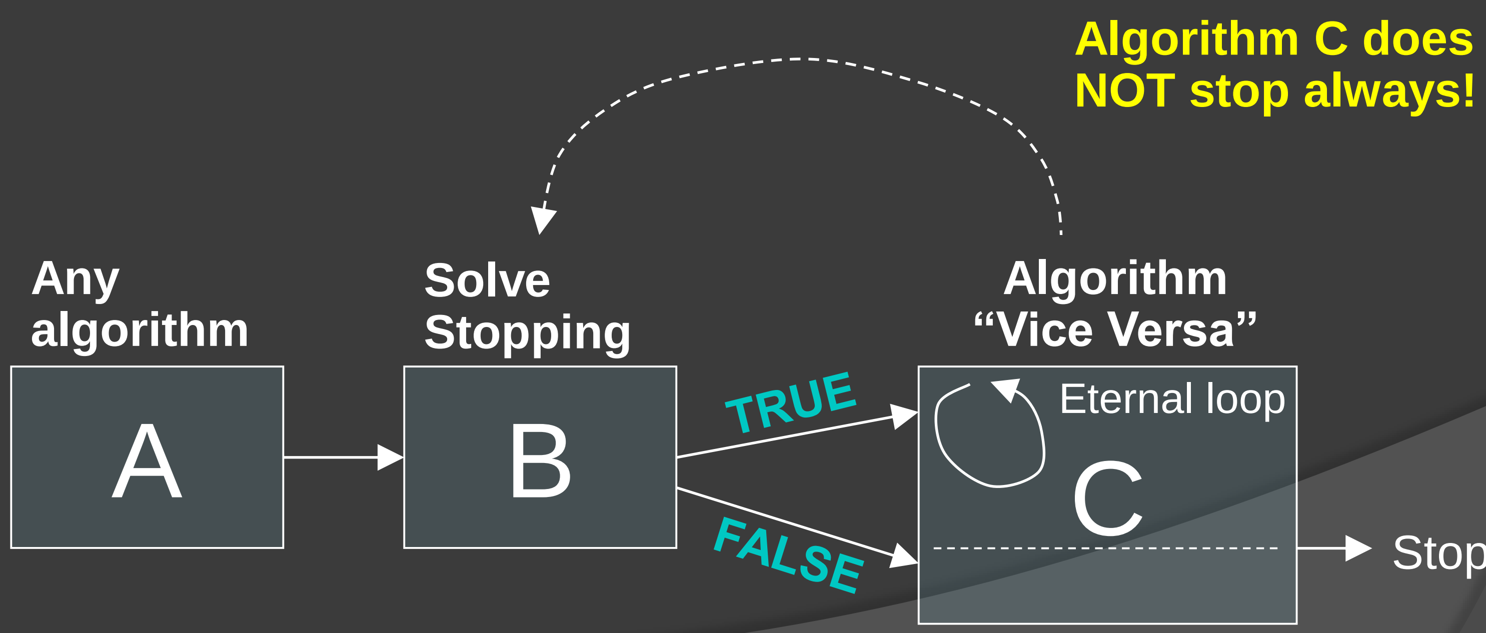
Non-solvable problems

- Give list of all rational numbers in [0..1].
- Longest sequence of π without the 0 digit.
- Stopping problem.
  - Can we always say a program will ever stop?



Non-solvable problems

- Give list of all rational numbers in [0..1]
- Longest sequence of π without the 0 digit
- Stopping problem



Algorithm principles

- Sequence
  - One command at a time
  - Parallel and distributed computing
- Condition
  - IF
  - CASE
- Loops
  - FOR
  - WHILE
  - REPEAT

Complexity

Time complexity:

- How much time it takes to compute
- Measured by a function $T(N)$

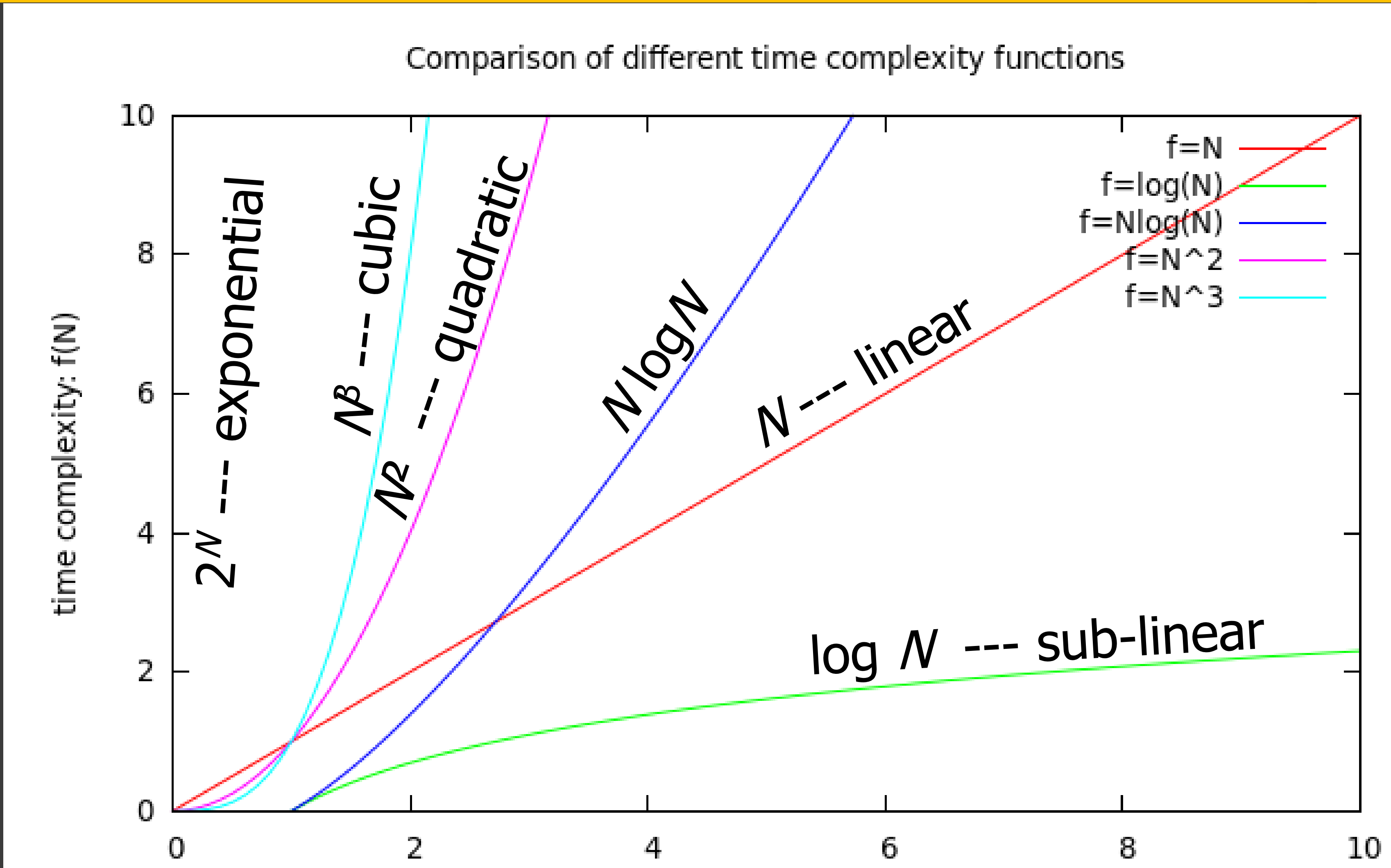
Space complexity:

- How much memory it takes to compute
- Measured by a function $S(N)$

Time complexity

- N = Size of the input
- $T(N)$ = Time complexity function
- Order of magnitude:
  - How rapidly $T(N)$ grows when N grows
  - For example: $O(N)$ $O(\log N)$ $O(N^2)$ $O(2^N)$

Asymptotic analysis



Examples: Bubble sort (N^2), Quicksort ($N \cdot \log N$)

Complexity Classes

- P (Polynomial Time) Class:
  - Can be solved by deterministic machines (our computers) in polynomial time.
    - Ex.: Selection sort.
- NP (Non-deterministic Polynomial Time) Class:
  - Can be solved by non-deterministic machines) in polynomial time. Its solution might be hard to find.
  - A solution can be guessed out of polynomially many options.
    - Ex.: Graph coloring. Coloring vertices of a graph in such a way that no two adjacent vertices have the same color.

Complexity Classes

- NP-Hard Class
  - It takes a long time to check them.
  - If a solution for an NP-Hard problem is given it will take a long time to check if it is correct or not.
  - Ex.: Halting problem.
- NP-Complete Class
  - If one could solve an NP-complete problem in polynomial time, then one could also solve any NP problem in polynomial time.
  - Ex.: Hamiltonian Cycle: A cycle in an undirected graph G=(V,E) that traverses every vertex exactly once.

Hard Problems

- One of the topics we may examine this semester is the class of NP-Complete problems.
- Some problems are known to have efficient solutions (in terms of the time required to solve a problem of a given size).
- For some problems, it has been proven that no efficient solution can possibly be developed.
  - These are known as NP-Hard problems

Hard Problems

- Also there's another set of problems for which no efficient solution has been found, nor has it been proven that one can't exist
  - These are the NP-Complete problems, and there has been a great deal of study on them in the last several decades
  - One of the appealing challenges of studying NP-complete problems is that if we can come up with an efficient solution for *one* of them, then it has been proven that we can have an efficient solution for *all* of them.
    - All NP Complete problems are NP-Hard but vice versa is not true.

Algorithms as a Technology

- Moore's law has made computers incredibly powerful compared to when algorithms were first studied in any appreciable detail.
- RAM and Hard Drive space has become (comparatively) enormous and cheap.
- Nevertheless, CPU time and Memory are still bounded resources.
- Even with a fast CPU, we should still strive for an efficient algorithm.
  - Running a bad algorithm on a fast system is still a waste of a perfectly good resource (time).

Algorithmic Efficiency

- When the problem gets big enough, a good algorithm, on slow hardware, even if poorly implemented, can beat a bad algorithm, on fast hardware, crafted by the best programmer.

Program Termination Problem

Consider the following program. Does it terminate for all values of $n \geq 1$?

```
while (n > 1) {
    if even(n) {
        n = n / 2;
    } else {
        n = n * 3 + 1;
    }
}
```

Program Termination Problem (2)

Not as easy to answer as it might first seem.

Say we start with n = 7, for example:

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

In fact, for all numbers that have been tried (***a lot!***), it does terminate . . .

. . . but in general?