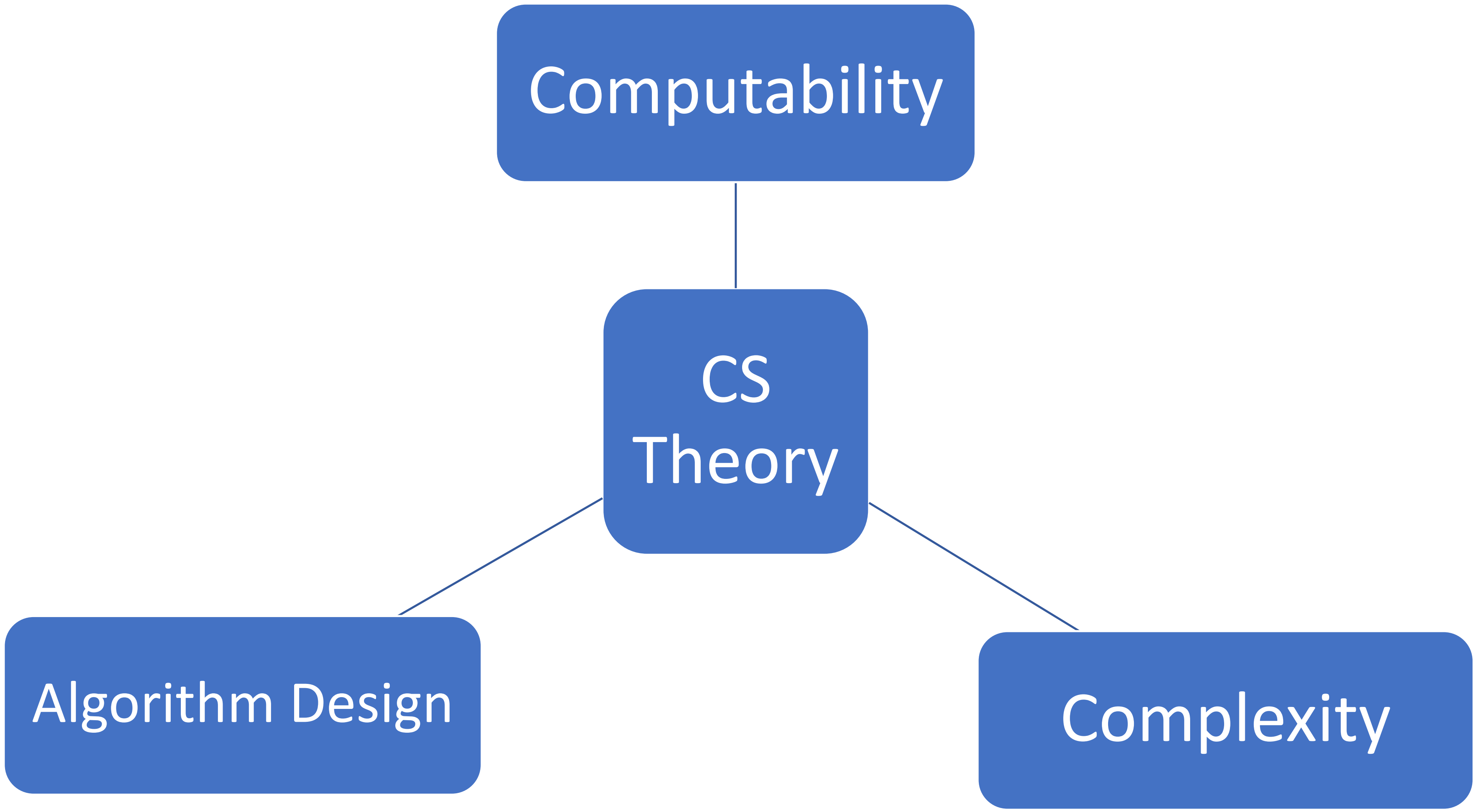
Introduction and Mathematical Preliminaries

COMP 6651 – Algorithm Design Techniques

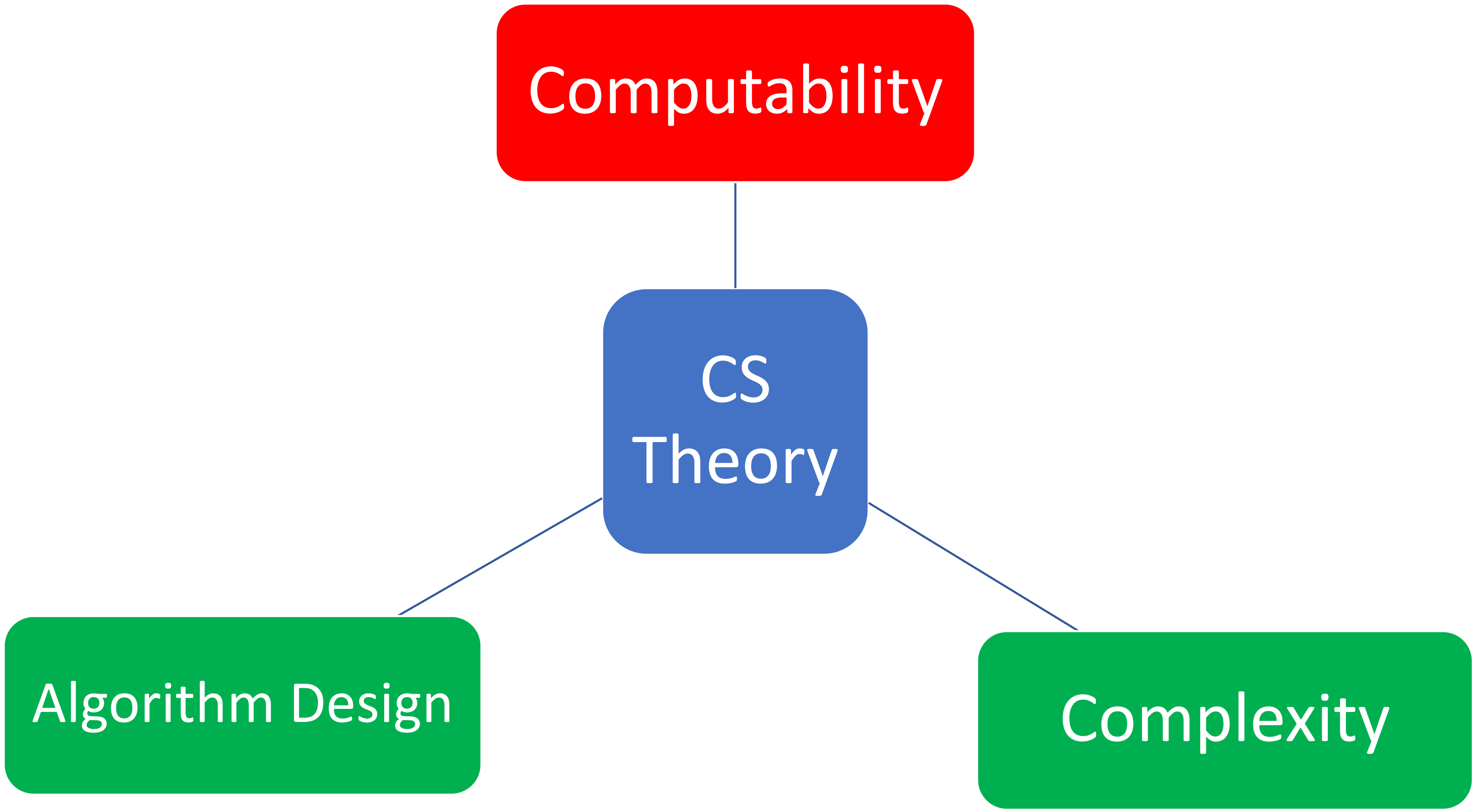
Denis Pankratov

Given a problem, how **uncomputable** is it?

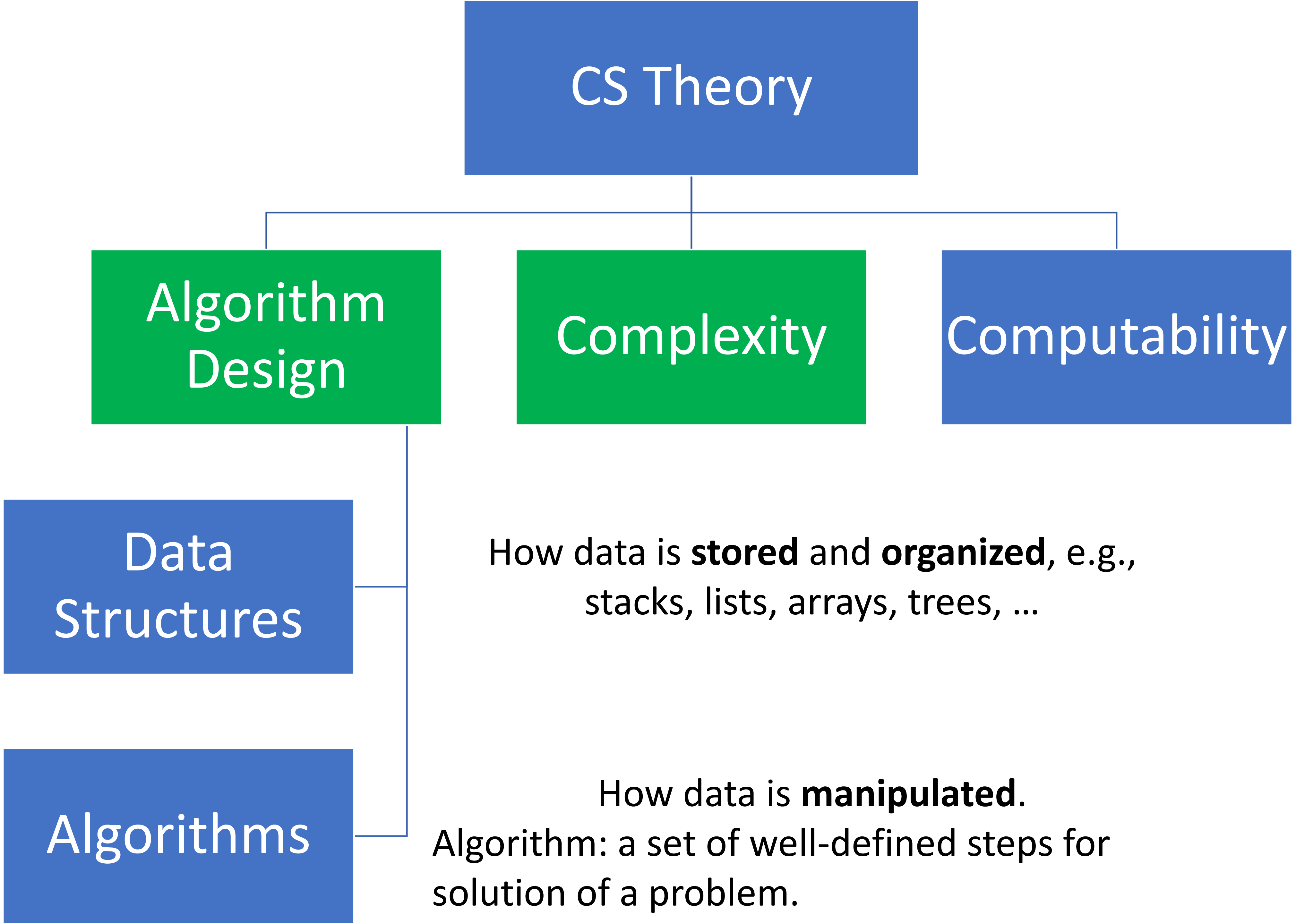


Given a problem, how much of a given resource (time, space, communication, circuit size, …) is **sufficient** to solve it?

Given a problem, how much of a given resource (time, space, communication, circuit size, …) is **necessary** to solve it?

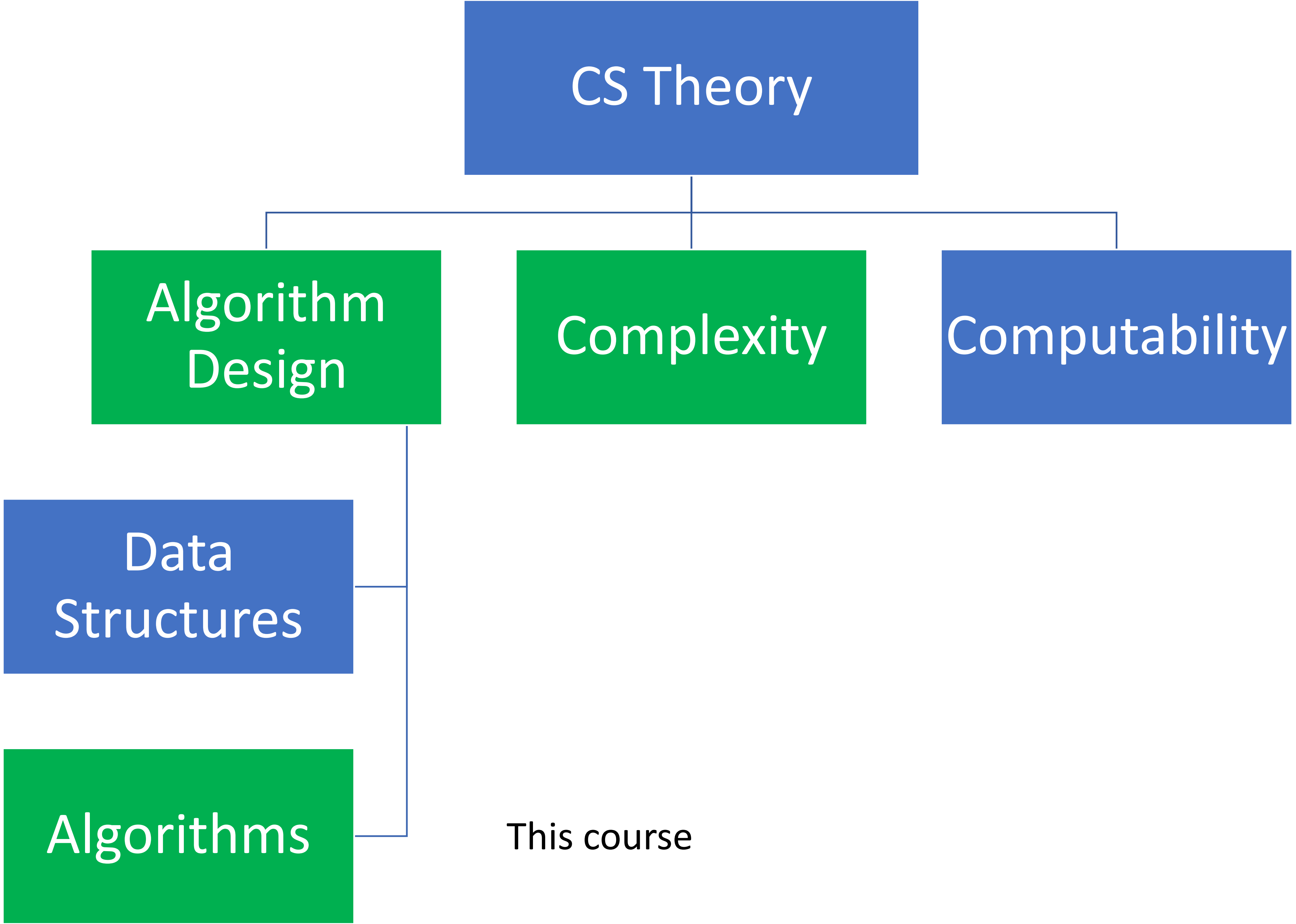


This course is mostly about Algorithm Design
+ a little bit of Complexity Theory

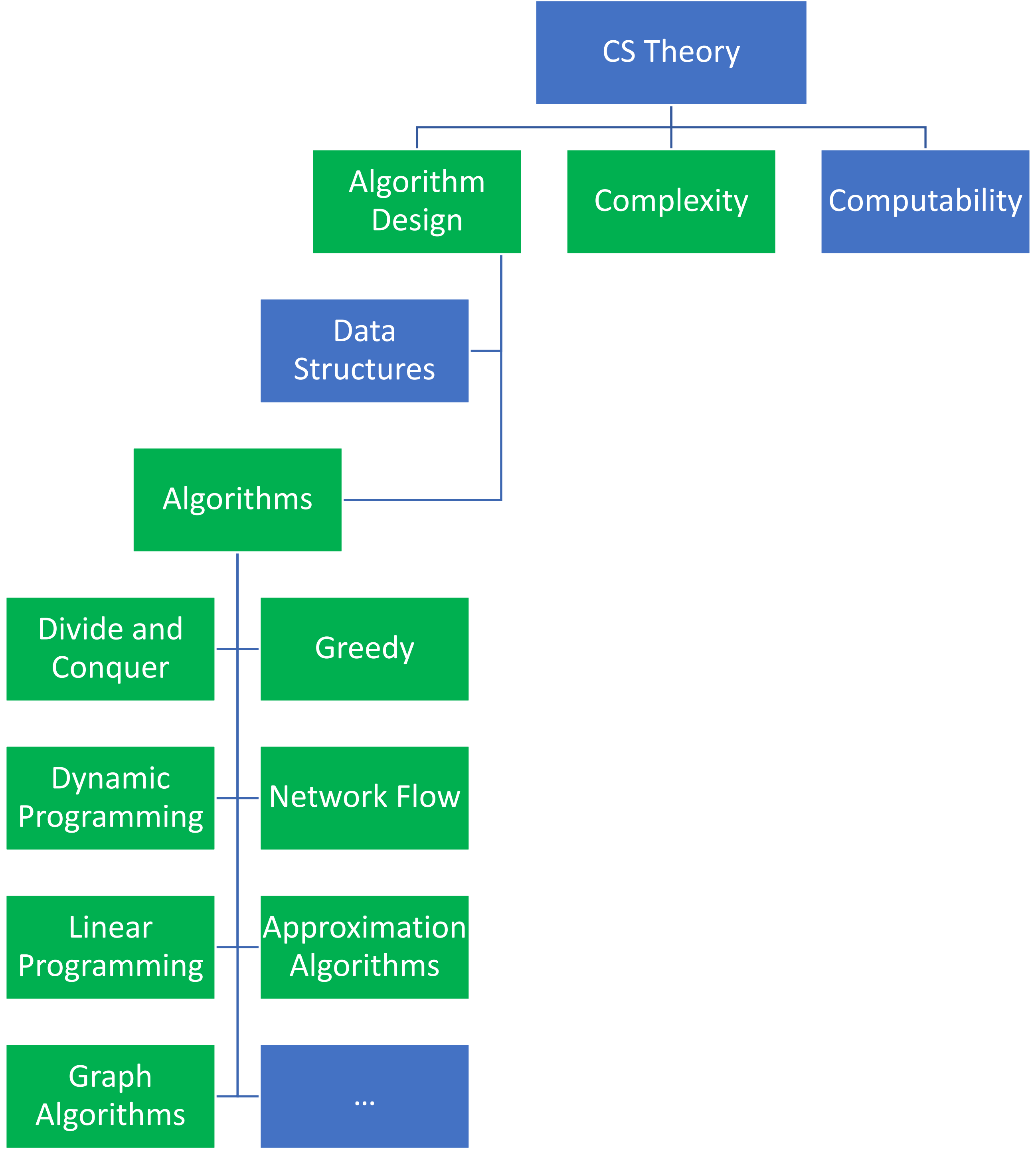


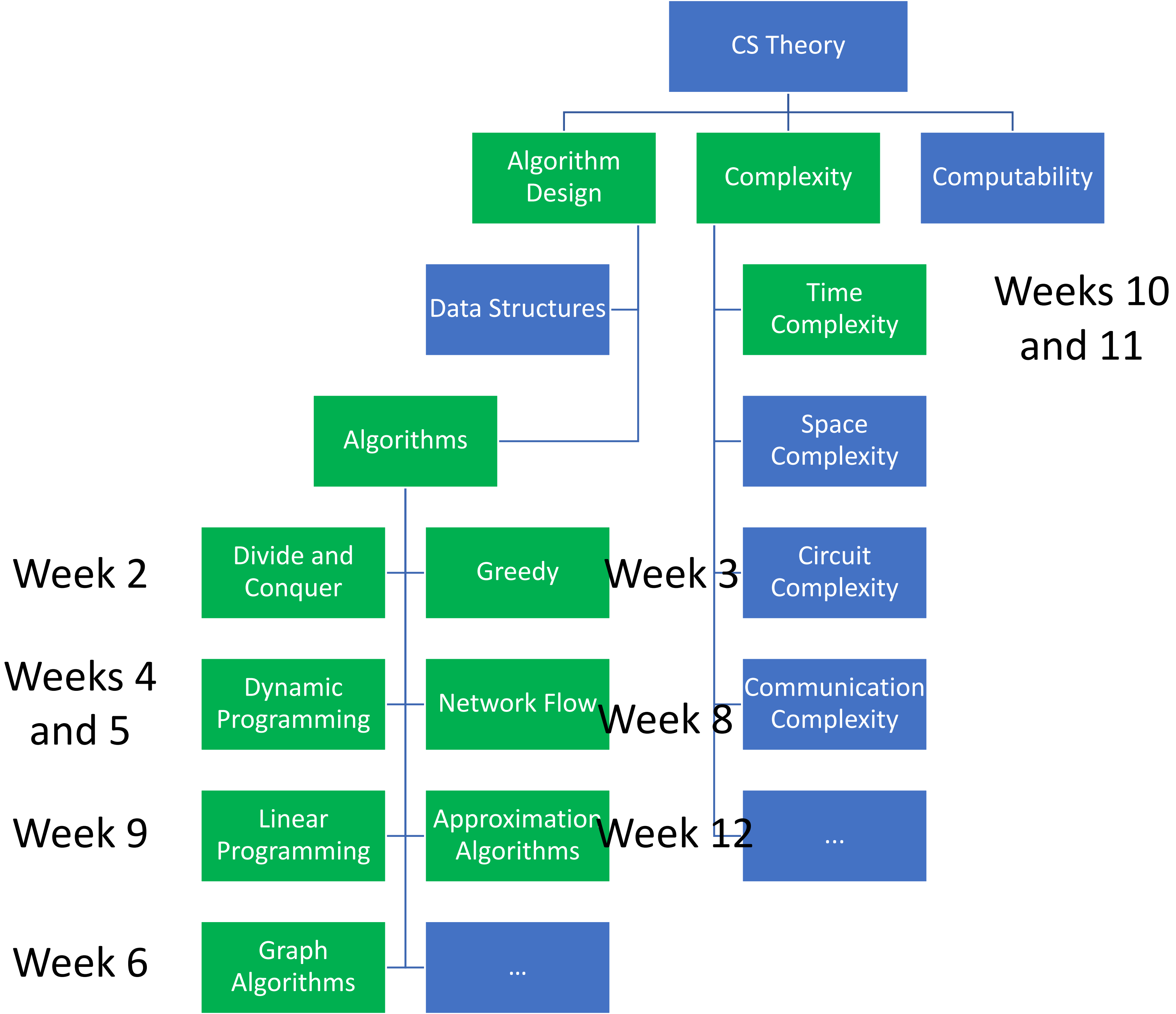
How data is **stored** and **organized**, e.g., stacks, lists, arrays, trees, …

How data is **manipulated**.
Algorithm: a set of well-defined steps for solution of a problem.


CS Theory
Algorithm Design
Complexity
Computability
Data Structures
Algorithms

This course

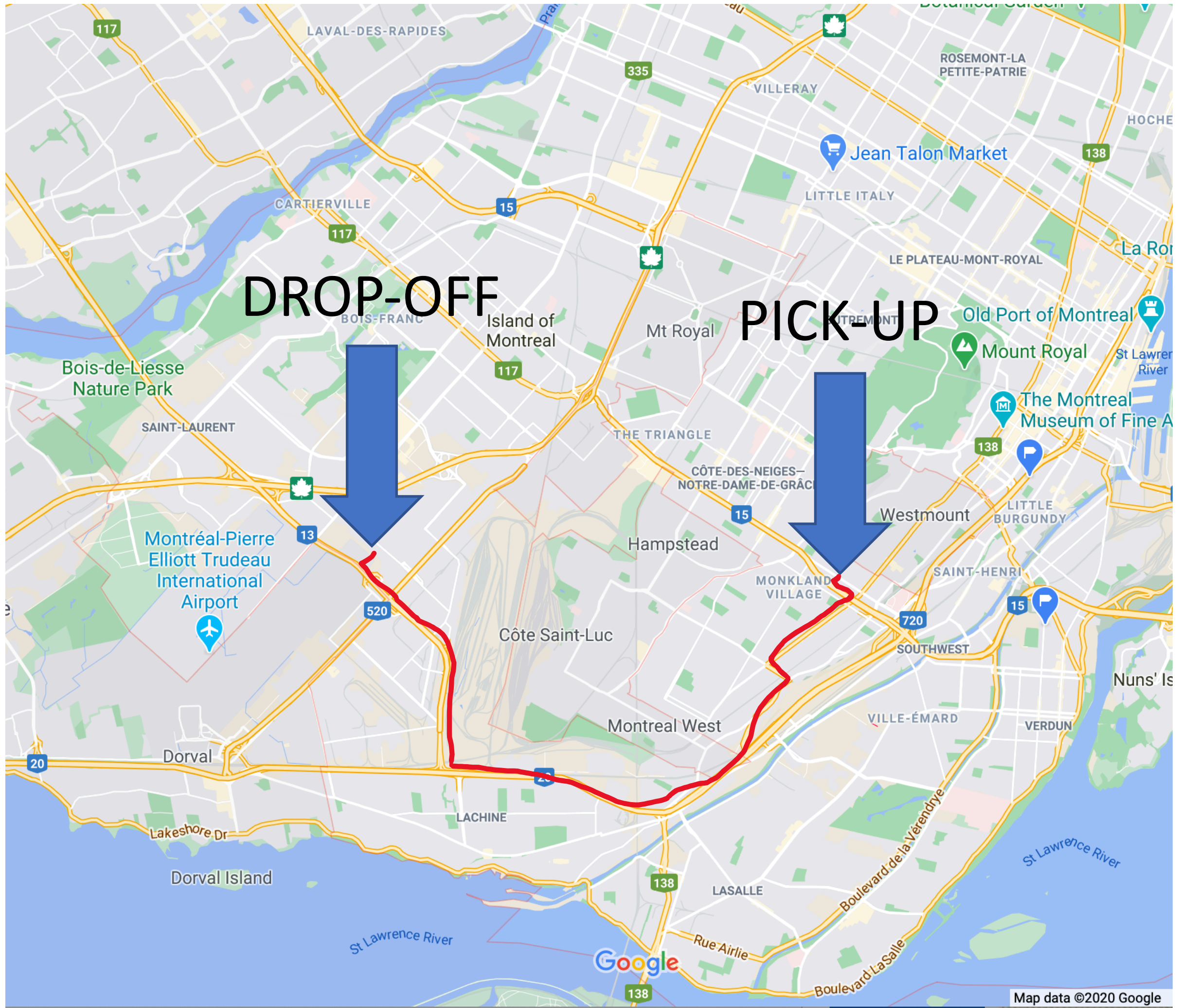

CS Theory
Algorithm Design
Complexity
Computability
Data Structures
Algorithms
Divide and Conquer
Greedy
Dynamic Programming
Network Flow
Linear Programming
Approximation Algorithms
Graph Algorithms
…


CS Theory
Algorithm Design
Complexity
Computability
Data Structures
Time Complexity
Weeks 10 and 11
Algorithms
Space Complexity
Week 2
Divide and Conquer
Greedy
Week 3
Circuit Complexity
Weeks 4 and 5
Dynamic Programming
Network Flow
Week 8
Communication Complexity
Week 9
Linear Programming
Approximation Algorithms
Week 12
…
Week 6
Graph Algorithms
…

Because of the pandemic, demand on delivery services has increased.

I am going to open a custom delivery start-up!

The underlying computational problem



Abstract problem

Shortest Path in a Weighted Graph Problem

Represent city as a graph $G = (V, E)$

V – set of vertices

E – set of edges

Weight function $w : E \to \mathbb{R}$

$w(e)$ = length of the edge e

Request

$v \in V$ – pick-up location

$u \in V$ – drop-off location

Goal: find a shortest path between v and u

```
For $k \in [n]$
  For $i \in [n]$
    For $j \in [n]$
      $dist[i,j] \leftarrow \min(dist[i,j],$
                $dist[i,k] + dist[k,j])$
```

Is the solution correct?

Is the solution efficient?

Is the solution scalable?

Is there a better solution?

And so on…

Empirical approach: code it and run	
Pros:	**Cons:**
Relatively easy	Might be costly
Fast	Limited guarantees
Fun	Limited insight
Practical	Limited guidance
	Less transferable to new technologies

Scientific approach:

think about it

Pros:	**Cons**:
Doesn’t require special hardware	Might be impractical:
Comes with guarantees (worst-case, average-case, etc.)	too simplistic
A lot of insight and guidance	too pessimistic
More universal	Much slower
	Might be impossible

Ideally you should strive to do both in your life!

This course is about the scientific/analytical approach:

design and analyze algorithms without running them

“In theory, theory and practice are the same. In practice, they are not.”

Albert Einstein

“Experience without theory is blind, but theory without experience is mere intellectual play.”

Immanuel Kant

“Theory is when you know everything but nothing works. Practice is when everything works but no one knows why. In our lab, theory and practice are combined: nothing works and no one knows why.”

Scientific/analytical approach

Requires arguing about programs without referencing specific hardware, operating system, programming language, etc.

- Abstract machine model

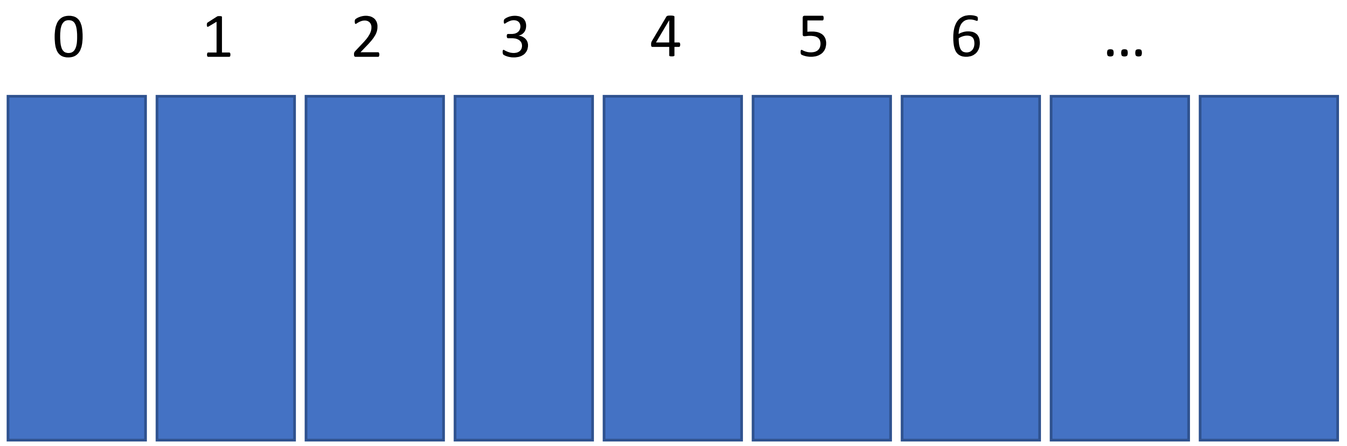
Random Access Machine

(by Church-Turing thesis, doesn’t really matter)

- Abstract programming language

Pseudocode

Random Access Machine (RAM)



Random Access Memory

Each cell stores a **word**

Single processor

Sequential execution

Unlimited memory

Instructions:

Arithmetic (add, subtract, multiply, divide, remainder, floor, ceiling)

Data movement (load, store, copy)

Control (conditional branch, subroutine call and return)

Basic instructions have the same cost

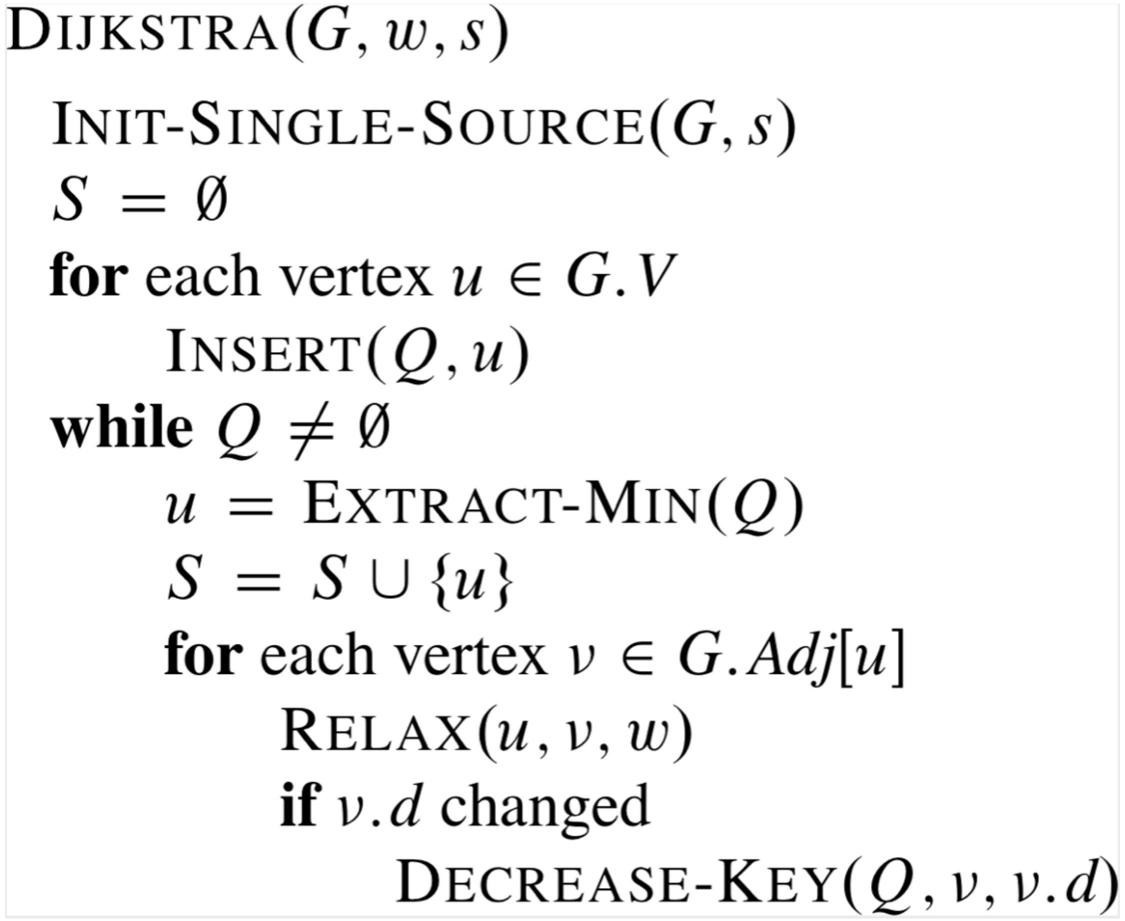
Pseudocode

Simplified way of writing RAM programs

Resembles many modern languages, e.g., C++, Java, Python

To what level of details is it specified?

Rule of thumb: a person who doesn't know your algorithm but knows C++ should be able to implement it and run it by using only your pseudocode



```
DIJKSTRA($G, w, s$)
  INIT-SINGLE-SOURCE($G, s$)
  $S = \emptyset$
  for each vertex $u \in G.V$
      INSERT($Q, u$)
  while $Q \neq \emptyset$
      $u$ = EXTRACT-MIN($Q$)
      $S = S \cup \{u\}$
      for each vertex $v \in G.Adj[u]$
          RELAX($u, v, w$)
          if $v.d$ changed
              DECREASE-KEY($Q, v, v.d$)
```

PROBLEM

Input: …

Output: …

ALGORITHM 1

```
For …
 For …
  $dist[i, j] \leftarrow$
```

ALGORITHM 2

```
For …
 For …
  $dist[i, j] \leftarrow$
```

ALGORITHM …

```
For …
 For …
  $dist[i, j] \leftarrow$
```

Instances

Measures of efficiency?

Time and **space**

as functions of <u>the size of input</u>

Start-up example:

size of the graph (number of vertices + number of edges)

+ number of requests

How many clients do you anticipate?

Can you run this service from your laptop or do you need a powerful server?

Will you be able to handle exponential growth of clients?

Will you be able to add other cities (Ottawa, Toronto, etc.)?

Types of analysis

Instances

n – size of the input

$T(n)$ – running time on inputs of length n

Which inputs?

<u>Worst case analysis:</u>

$T(n)$ = the longest running time for any input of length n

FOCUS in this course

<u>Average case analysis:</u>

$T(n)$ = the running time averaged over all inputs of length n

<u>Best case analysis:</u>

$T(n)$ = the shortest running time for any input of length n

Other types: smoothed analysis, parameterized analysis, pareto optimality, etc.

Scalability ≈ asymptotics

Counting of time is not exact AND we are interested in order of growth

We express the run-time using the **asymptotic notation:**

$$f = O(g), f = \Omega(g), f = \Theta(g), f = o(g), f = \omega(g), f \sim g, \ldots$$

More on this next ☺

Asymptotic Notation

Big-O

Let $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$

$$f = O(g) \text{ or alternatively } f \in O(g)$$

Meaning: “eventually f is dominated by a scaled version of g”

Big-O

Let $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$

$$f = O(g) \text{ or alternatively } f \in O(g)$$

Meaning: “eventually f is dominated by a **scaled version of $\boldsymbol{g}$**”

Formalization:

There exists a constant c so that $c \cdot g$ is a scaled version of g

Big-O

Let $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$

$$f = O(g) \text{ or alternatively } f \in O(g)$$

Meaning: “eventually $\boldsymbol{f}$ **is dominated by** a scaled version of g”

Formalization:

There exists a constant c so that $c \cdot g$ is a scaled version of g
$f(n) \leq c \cdot g(n)$

Big-O

Let $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$

$$f = O(g) \text{ or alternatively } f \in O(g)$$

Meaning: "**eventually** f is dominated by a scaled version of g"

Formalization:

There exists a constant c so that $c \cdot g$ is a scaled version of g
$f(n) \leq c \cdot g(n)$

There exists some value n_0 such that for all $n \geq n_0$ …

Big-O

Let $f : \mathbb{N} \rightarrow \mathbb{N}$ and $g : \mathbb{N} \rightarrow \mathbb{N}$

$$f = O(g) \text{ or alternatively } f \in O(g)$$

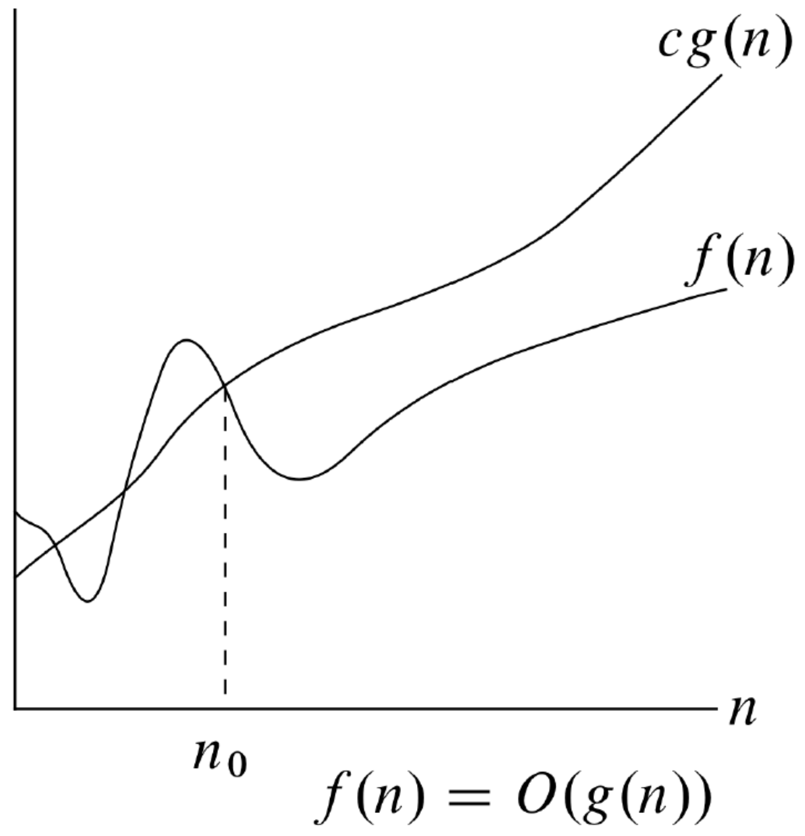
Meaning: "**eventually $\boldsymbol{f}$ is dominated by a scaled version of $\boldsymbol{g}$**"

Formalization:

There exists a constant c so that $c \cdot g$ is a scaled version of g
$f(n) \leq c \cdot g(n)$

There exists some value n_0 such that for all $n \geq n_0$ …

$$\exists c, n_0 \forall n \geq n_0 \; f(n) \leq c \cdot g(n)$$



(b)

$$\exists c, n_0 \forall n \geq n_0 \; f(n) \leq c \cdot g(n)$$

Examples:

(1) $f(n) = 4 \cdot n$ and $g(n) = n + 8$

Take $c = 4$ and $n_0 = 1$ and observe that

$$f(n) = 4 \cdot n \leq 4 \cdot n + 32 = 4\,(n + 8) = 4\,g(n)$$

For all $n \geq 1$

Therefore, $4 \cdot n = O(n + 8)$

(2) $f(n) = n^2$ and $g(n) = n(n + 1)(n + 2)$

Take c = 1 and $n_0 = 1$ and observe that

$$g(n) = n(n^2 + 3n + 2) = n^3 + 3n^2 + 2n \geq n^2 = f(n)$$

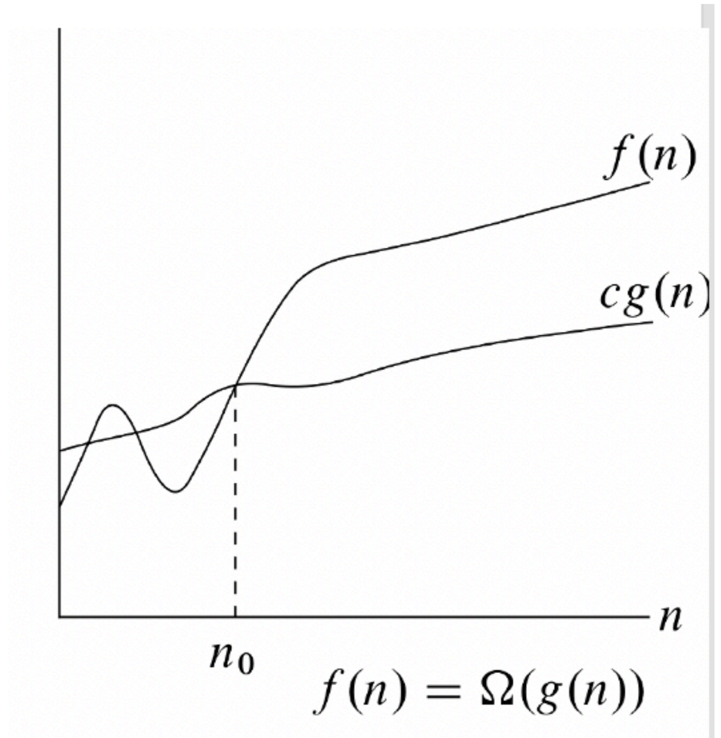
For all $n \geq 1$

Big-Ω

$f = \Omega(g)$ or alternatively $f \in \Omega(g)$

"eventually f dominates a nontrivial scaled version of g"

$\exists c > 0, n_0 \forall n \geq n_0 \; f(n) \geq c \cdot g(n)$

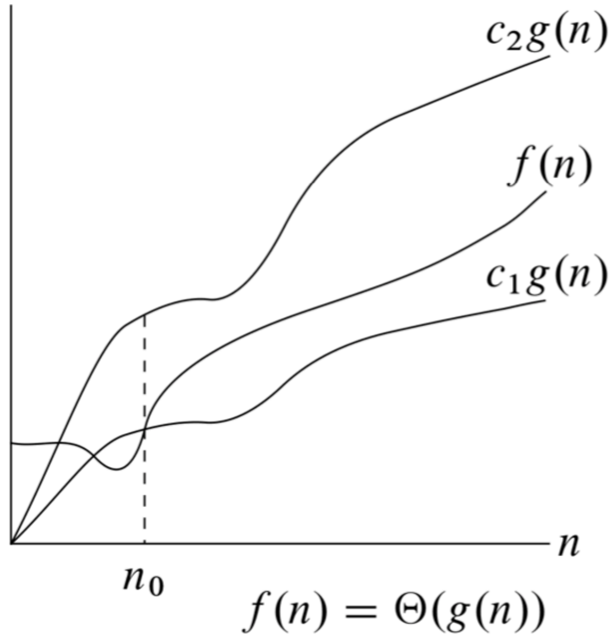


Big-Θ

$f = \Theta(g)$ or alternatively $f \in \Theta(g)$

"f and g have asymptotically similar growth"

$f = O(g)$ and $f = \Omega(g)$



Simplifications of expressions in asymptotic notation

c is a constant:

$$O(f(n) + c) = O(f(n))$$

$$O(cf(n)) = O(f(n))$$

If, $f_1 = O(f_2)$:

$$O(f_1(n) + f_2(n)) = O(f_2(n))$$

If $f_1 = O(g_1)$ and $f_2 = O(g_2)$ then

$$O(f_1(n) * f_2(n)) = O(g_1(n) * g_2(n))$$

Be careful with simplifications!

It is okay to apply simplification rules a few (constantly many) times:

$$O(2n^2 + 3n + 2\sqrt{n}) = O(2n^2 + 3n) = O(2n^2) = O(n^2)$$

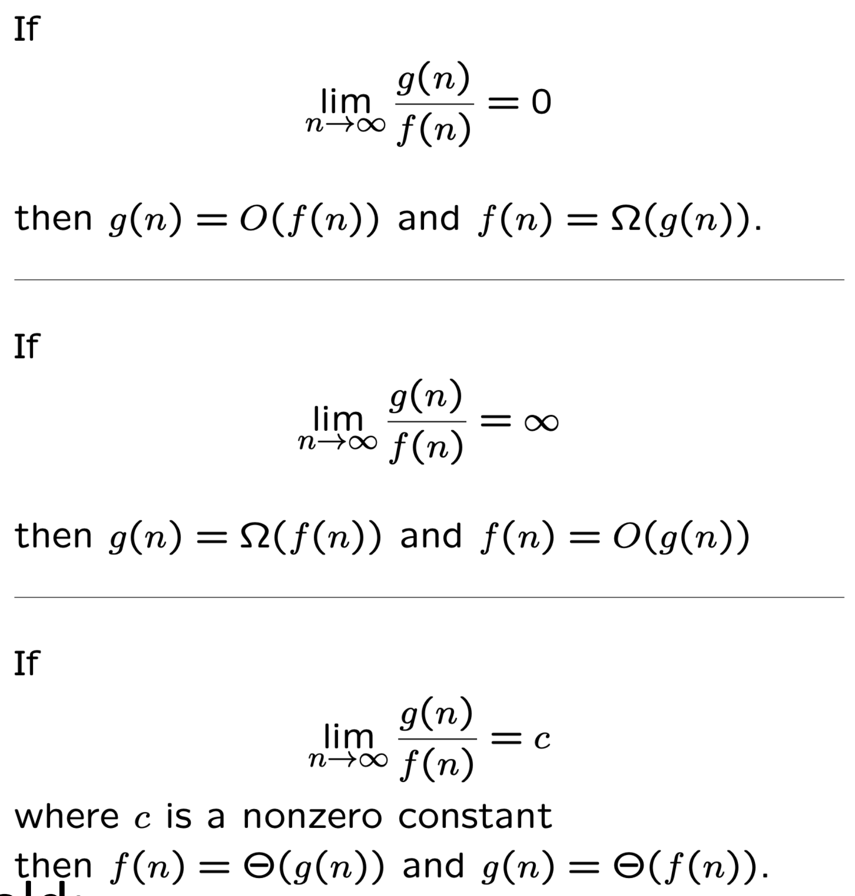
It is not okay to apply simplification rules non-constantly many times:

$$O\left(\sum_{i=1}^{n} 5\right) = O(\underbrace{5 + 5 + \cdots + 5}_{n \text{ times}}) = O\big((\underbrace{5 + 5 + \cdots + 5}_{n-1 \text{ times}}) + 5\big)$$

$$= O(\underbrace{5 + 5 + \cdots 5}_{n-1 \text{ times}}) = \cdots = O(5) = O(1)$$

MISTAKE

Simplifications of expressions in asymptotic notation



If

$$\lim_{n\to\infty}\frac{g(n)}{f(n)} = 0$$

then $g(n) = O(f(n))$ and $f(n) = \Omega(g(n))$.

If

$$\lim_{n\to\infty}\frac{g(n)}{f(n)} = \infty$$

then $g(n) = \Omega(f(n))$ and $f(n) = O(g(n))$

If

$$\lim_{n\to\infty}\frac{g(n)}{f(n)} = c$$

where c is a nonzero constant
then $f(n) = \Theta(g(n))$ and $g(n) = \Theta(f(n))$.

WARNING!!!

These are not definitions!

The other direction does not hold:

Give an example of functions f and g such that $f = O(g)$ but $\lim_{n\to\infty}\frac{g(n)}{f(n)} = 0$ does not hold!

Little-o, little-ω, asymptotic equality

$f(n) = o\big(g(n)\big)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$f(n) = \omega\big(g(n)\big)$ if $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$

$f(n) \sim g(n)$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

Examples:

$$n = o(n^2),$$
$$3^n = \omega(2^n),$$
$$4n^2 + 8n + 8 \sim 4n^2.$$

Examples

$$2n^2 + 5n - 6 = O(2^n)$$
$$2n^2 + 5n - 6 = O(n^3)$$
$$2n^2 + 5n - 6 = O(n^2)$$
$$2n^2 + 5n - 6 \neq O(n)$$

$$2n^2 + 5n - 6 \neq \Omega(2^n)$$
$$2n^2 + 5n - 6 \neq \Omega(n^3)$$
$$2n^2 + 5n - 6 = \Omega(n^2)$$
$$2n^2 + 5n - 6 = \Omega(n)$$

$$2n^2 + 5n - 6 \neq \Theta(2^n)$$
$$2n^2 + 5n - 6 \neq \Theta(n^3)$$
$$2n^2 + 5n - 6 = \Theta(n^2)$$
$$2n^2 + 5n - 6 \neq \Theta(n)$$

$$2n^2 + 5n - 6 = o(2^n)$$
$$2n^2 + 5n - 6 = o(n^3)$$
$$2n^2 + 5n - 6 \neq o(n^2)$$
$$2n^2 + 5n - 6 \neq o(n)$$

Quick summary

$f(n) = O(g(n))$:
$f(n)$ grows at most as fast as $g(n)$,
$g(n)$ is an upper bound on the growth of $f(n)$.

$f(n) = \Omega(g(n))$:
$f(n)$ grows at least as fast as $g(n)$,
$g(n)$ is a lower bound on the growth of $f(n)$.

$f(n) = \Theta(g(n))$:
$f(n)$ grows exactly as fast as $g(n)$.

$f(n) = o(g(n))$:
$f(n)$ grows slower than $g(n)$,

You should now be able to…

- Describe the main topics and their organization within theoretical computer science.
- Explain the pros and cons of empirical analysis and theoretical analysis.
- Describe the Random Access Machine model.
- Explain differences and similarities of pseudocode vs real code.
- Define worst case, average case, and best case types of analysis.
- Give a formal as well as informal definitions of asymptotic notation: big-O, big-Ω, big-Θ, little-o, little-ω, asymptotic equality.

Review questions…

- True or false: if an algorithm works correctly on a large set of carefully chosen input instances then the algorithm is correct.
- True or false: if $f = \Theta(g)$ then $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ =constant.
- In the Random Access Machine model, when are we not allowed to use the built-in “multiply” instruction on two input integers?
- Sort the following functions according to their asymptotic order of growth:

2^n, $(\log n)^{\frac{\log n}{\log \log n}}$, $\sqrt{n}$, $100 \cdot n + n^2$, $\log n$, 3^n, $2^{\sqrt{n}}$, $n!$

Other material you should know/review for this course

- Functions (injective, surjective, bijective, partial, total, …)
- Sets and operations on sets
- Relations (equivalence relations)
- Recurrences and solutions to them, e.g., $T(n) = 2\,T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 10 \cdot n$ and $T(1) = 1$
  - Guess and substitute method
  - Characteristic polynomial method
  - Master’s theorem
- Basic proof techniques: induction, contradiction, pigeonhole principle