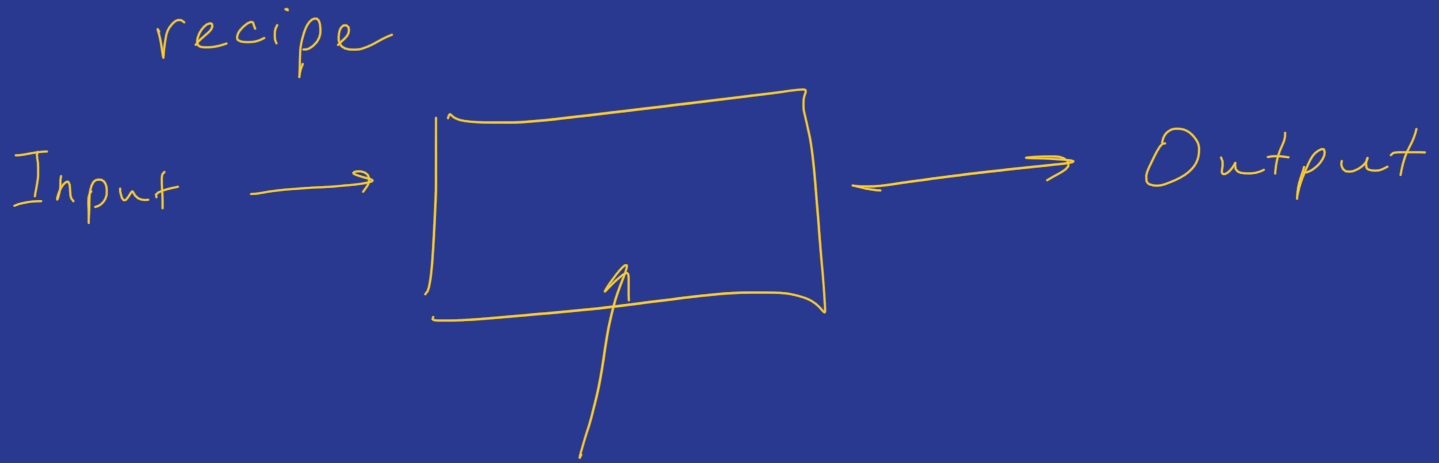
Algorithm Design & Analysis II

What is an algorithm?



What makes a *good* algorithm?

→ efficiency — time & space *

→ modularity → compatibility

→ correctness *

→ well-defined & specific

→ consistency

→ security → no crashing!

→ extensibility

What is a data structure and what makes a *good* data structure?

→ structure that holds data

→ abstract

→ reusable on multiple data types

→ manipulate data in an efficiency

→ (functionality)

Why is it important to think about data structures when you are implementing an algorithm?

→ algorithms require certain data structures

Does efficiency really matter?

Does efficiency really matter?

More efficient algorithms and data structures can…

- enable new research.
- enable new technology.
- enable faster response times for different applications.
- enable faster processing for large amounts of data.
- make it less likely that your program will crash.
- potentially break some cryptographic systems!
- make coding a little easier.

Algorithm Paradigms

Paradigm	**Description**	**Example**
brute force	systematically tries every possible solution	linear search

Algorithm Paradigms

Paradigm	Description	Example
brute force	systematically tries every possible solution	linear search
prune & search	eliminates fractions of the search space	binary search

Algorithm Paradigms

Paradigm	Description	Example
brute force	systematically tries every possible solution	linear search
prune & search	eliminates fractions of the search space	binary search
divide & conquer	divides solutions into smaller solutions, solves those, and combines them back together	Mergesort

Algorithm Paradigms

Paradigm	Description	Example
brute force	systematically tries every possible solution	linear search
prune & search	eliminates fractions of the search space	binary search
divide & conquer	divides solutions into smaller solutions, solves those, and combines them back together	Mergesort
dynamic programming	stores the solutions to overlapping subsolutions that will then be accessed to solve the larger problems	non-recursive version of fibonacci numbers

Algorithm Paradigms

Paradigm	Description	Example
brute force	systematically tries every possible solution	linear search
prune & search	eliminates fractions of the search space	binary search
divide & conquer	divides solutions into smaller solutions, solves those, and combines them back together	Mergesort
dynamic programming	stores the solutions to overlapping subsolutions that will then be accessed to solve the larger problems	non-recursive version of fibonacci numbers
greedy	builds a global solution to a problem by repeatedly making the best possible local choice	Dijkstra’s Algorithm