Algorithm Design & Analysis

Part 1: Introduction & Paradigms

What is an algorithm?

Computation

input → output

series of steps

What makes a *good* algorithm?

repeatable → consistent output

ease of implementation

efficiency - time & space

generalizable → handle enough inputs

wide coverage

→ handle errors or bad input

well-defined

accuracy

clearly defined input & output

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

modular → dynamic

access to data → operations

well-defined

consistency

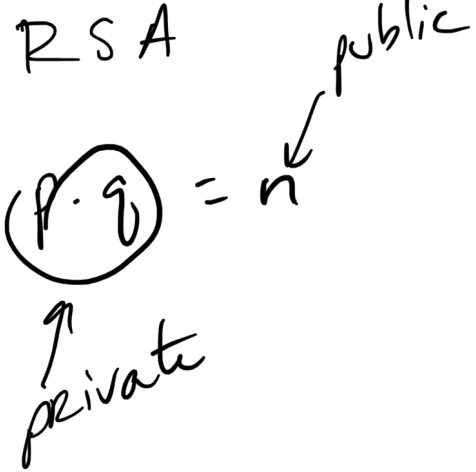
Why is it important to think about data structures when you are implementing an algorithm?

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.



What are some algorithms and data structures you are already aware of and how would you describe them?

Sorting - Bubblesort, Quicksort, Mergesort, Bogo Sort, Insertion Sort

Graphs

Recursive Backtracking

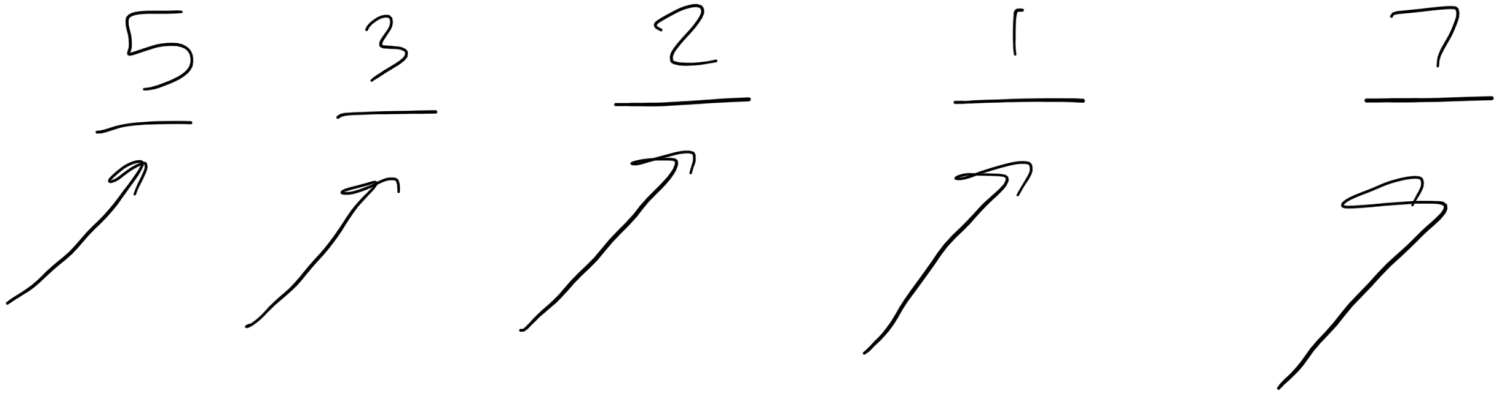
Heaps

binary trees

Stacks & Queues

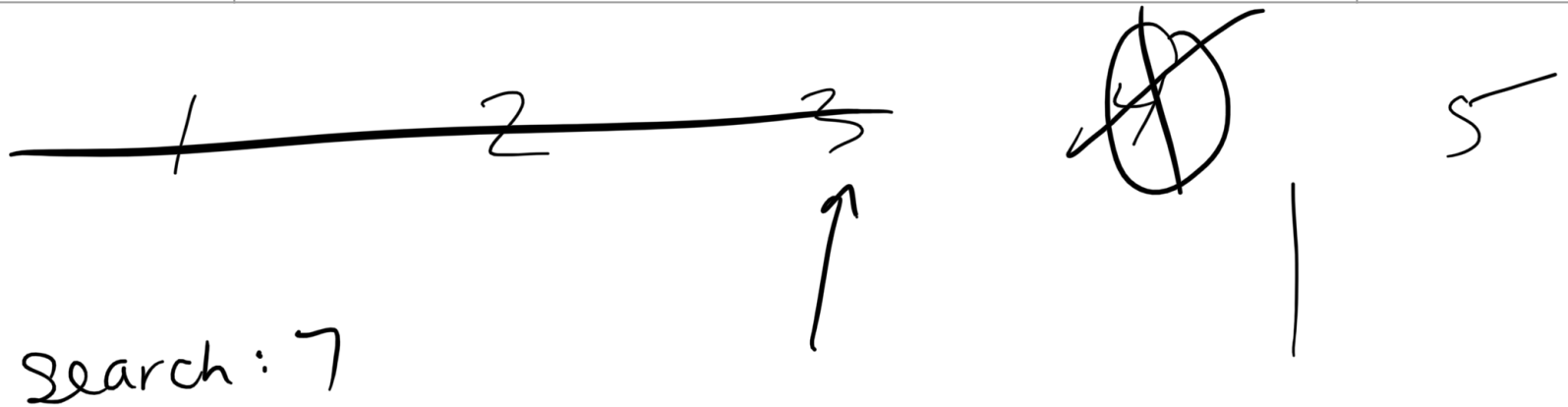
Algorithm Paradigms

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



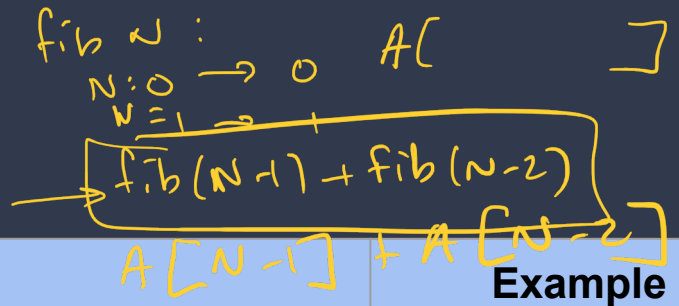
Algorithm Paradigms

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