CS2 Notes (1/22)

What do you think when you hear "data structures"?

- The structure and order of the code in your program.

What do you think when you hear "algorithm"?

- The program you write that solves something.

Given answers to 1 and 2, what is something you would like to use a data structure or an algorithm for?

- Build some cool stuff.

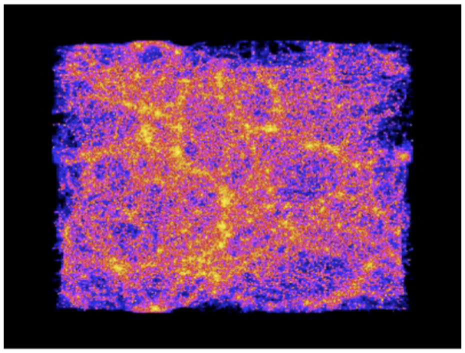
Examples

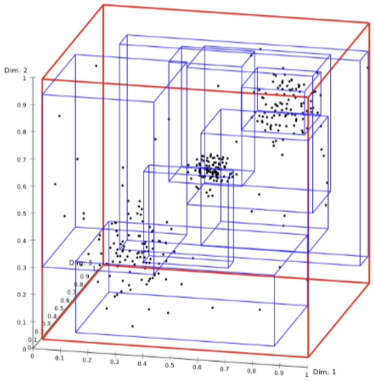
Algorithms:

- Barnes-Hut approximation for an n-body simulation.

Data Structure:

- 3-dimensional R-tree that stores points (bodies) hierarchically.





Key Concept: Stability

- Using the correct algorithm and data structure ensures stability.

Sorting Algorithms

- **Question:** Which sorting algorithm is faster? Why is it faster? Is it always faster?
- **Answer:** The speed of a sorting algorithm depends on the context. For example:
  - QuickSort is faster on average but has a worst-case time complexity of O(n2)O(n^2)O(n2).
  - MergeSort is stable and guarantees O(nlogn)O(n \log n)O(nlogn) but uses more memory.

Where are the "sticks" stored?

- In data structures such as:
  - Arrays
  - Linked Lists
  - ArrayLists
  - Heaps

What CS:22030 Offers

Goal:

- Become better at solving problems with programs.

Key Topics:

1. **How to organize a large program:**
   - Interfaces: "I don’t need to know how it works; just tell me what it does."
   - Object-Oriented Programming: Modeling entities as objects that interact with each other.
   - Continuous Integration:

        - Include tests specifying expected input/output behavior.
        - Run tests to ensure changes won’t break the program.
2. **Learn a new programming language and features:**
    - Encapsulation
    - Static Typing
    - Immutable Data Structures
    - Visibility and Protection

Example: Car Racing Game

Objects to Include:

- **Car**, **Racetrack**, **Timer**, **Race**

Class: Car

- **Objects:** Polo, Minivan, Beetle
- **Methods:**
    - `refuel()`
    - `getFuel()`
    - `setSpeed()`
    - `drive()`
- **Attributes:**
    - Fuel
    - Max Speed

Course Objectives

1. Analyze an algorithm in terms of time and space complexity.
2. Calculate the time complexity of operations on a data structure.
3. Choose the right data structure for a given problem.
4. Specify an interface and implement an abstract data type.
5. Use essential data structures:
    - Maps
    - Trees
    - Arrays
    - Lists
    - Queues
6. Solve problems using iteration or recursion.

7. Recognize scenarios where data structures are useful.
8. Write informal proofs about program properties.

Cuz java sucks, c# is better.