Data Structures and Algorithms

CS 110
Physical Data Structures

Which of these is the best?
Data Structures in a Computer

• We can program different ways of organizing data inside of the computer

• Each different way of organizing the data has various tradeoffs
  – What operations are supported? (e.g. add, delete, find, etc.)
  – How efficient are these operations?
  – How much overhead is involved in organizing the data?

• These issues are discussed at length in the Data Structures course
Data Structures in Processing

• Data structures can be implemented in Processing as classes
• What would the field of the class be in this case?
• What would the methods be?
Built-in Collection Classes

• **ArrayList**
  – A built-in object that stores and manages an *arbitrary* number of data items of any type (Objects).
  – Objects in an ArrayList are access by index [0..size-1]

• **HashMap**
  – A built-in object that stores and manages an *arbitrary* number of data items of any type (Objects).
  – Objects in a HashMap are access by a **key**, which can be another Object, frequently a String.
ArrayList

• Constructors

ArrayList<Object> lst1 = new ArrayList();
ArrayList<Object> lst2 = new ArrayList(initialSize);

• Fields

• Methods

size() // Returns the num of items held.
add(Object o) // Appends o to end.
add(int idx, Object o) // Inserts o at pos idx.
remove(int idx) // Removes item at pos idx.
get(int idx) // Gets items at idx. No removal.
set(int idx, Object o) // Replaces item at idx with o.
clear() // Removes all items.
isEmpty() // true if empty.
ArrayList Example – Box Dropper

// Box Dropper
ArrayList<Box> boxes = new ArrayList<>();
void setup() {
  size(500, 500);
}

void draw() {
  background(0);

  for (int i = boxes.size()-1; i>=0; i--) {
    boxes.get(i).update();
    boxes.get(i).draw();
    // Remove Box from ArrayList if below sketch
    if (boxes.get(i).y > height) {
      boxes.remove(i);
      println(boxes.size() + " boxes remaining");
    }
  }
}

void mousePressed() {
  Box b = new Box(mouseX, mouseY);
  boxes.add(b);
  println(boxes.size() + " boxes in ArrayList");
}

class Box {
  float x, y, v;

  Box(float tx, float ty) {
    x = tx; // x position
    y = ty; // y position
    v = 0.0; // y velocity
  }

  void draw() {
    fill(200);
    rect(x, y, 20, 20);
  }

  void update() {
    y += v; // Physics
    v += 0.02;
  }
}
Cooler Array List Example
The Basic Idea

• Start with a single tile that contains the entire image

• On each frame
  – Choose random tile to start falling
  – Choose random tile to split into two new tiles

• At a high-level, how would we use the methods of the ArrayList to implement these two steps?
HashMap

• Constructors

```java
HashMap<Object, Object> map1 = new HashMap();
HashMap<Object, Object> map2 = new HashMap(initialCapacity);
```

• Fields

• Methods

```java
size() // Returns num of items held.
put(Object key, Object o) // Puts o in map at key
remove(Object key) // Remove Object at key
get(Object key) // Get Object at key
containsKey(Object key) // True if map contains key
containsValue(Object val) // True if map contains val
clear() // Removes all items.
isEmpty() // true if empty.
```
More Data Structures

Trees,
Heaps,
Graphs,
Linked Lists,
Queues,
Stacks,
etc.

Imagine the Data Structures that Google Uses
Algorithm

• A well-defined set of instructions for solving a particular kind of problem.

• Algorithms exist for systematically solving many types of problems
  – Sorting
  – Searching
  – ...

Searching

This is a fundamentally important problem for a myriad of applications (from finding webpages to searching for fragments of DNA)

The problem:

Given a collection of data, determine if a query is contained in that collection.
Motivating Example: Spellchecker!
Exhaustive (Linear) Search

- Systematically enumerate all possible values and compare to value being sought
- For an array, iterate from the beginning to the end, and test each item in the array

Find “awsome"
Binary Search

- Quickly find an item (val) in a **sorted** list.

- Recursive Procedure:

```python
binarySearch(startIdx, stopIdx, data, query)
  if startIdx == stopIdx
    return data[startIdx] == query
  midPoint = (startIdx+stopIdx)/2
  if words[midPoint] >= query
    return binarySearch(startIdx, midPoint, data, query)
  else
    return binarySearch(midPoint+1, stopIdx, data, query)
```

The most efficient way to play "guess the number" ...
Binary Search Example

Find “awsome"

aardvark ... macabre ... Zyzzogeton
Binary Search Example

Find “awsome"

aardvark ... fable ... macabre
Binary Search Example

Find “awsome"

aardvark ... catfish ... fable
Binary Search Example

Find "awsome"

aardvark  ...  beetle  ...  catfish
Binary Search Example

Find “awsome”

aardvark ... awake ... beetle
Binary Search Example

Find “awsome"

awaken ... banjo ... beetle

Repeat a few more times…
Binary Search Example

Find “awsome" → awry

Return false
Spell-Checking Midsummer Night’s Dream

- Exhaustive Search: 385,554 milliseconds to check the entire text

- Binary Search: 104 milliseconds to check the entire text

- Speedup: 3,707 times!!!
"Can I Guess?"

Given the numbers 1 to 1,000, what is the minimum number of guesses needed to find a specific number if you are given the hint "higher" or "lower" for each guess you make?

As asked at Facebook
Sitcom/Dictator Game

http://www.smalltime.com/Dictator
Transitioning to Java in 1 Slide

Processing:

```java
void setup() {
    println("Hello World!");
}
```

Java:

class test {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```