CSCI 241: Data Structures

Lecture 1

Introduction

ADTs

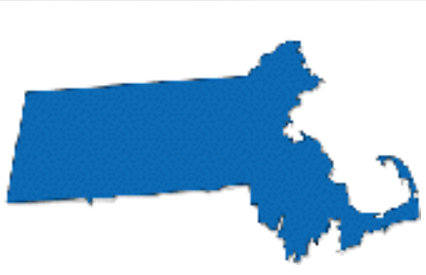
Runtime Analysis

Today

1. About Me
2. Course Overview
3. Review: Abstract Data Types
4. Introduction to Runtime Analysis

About Me:

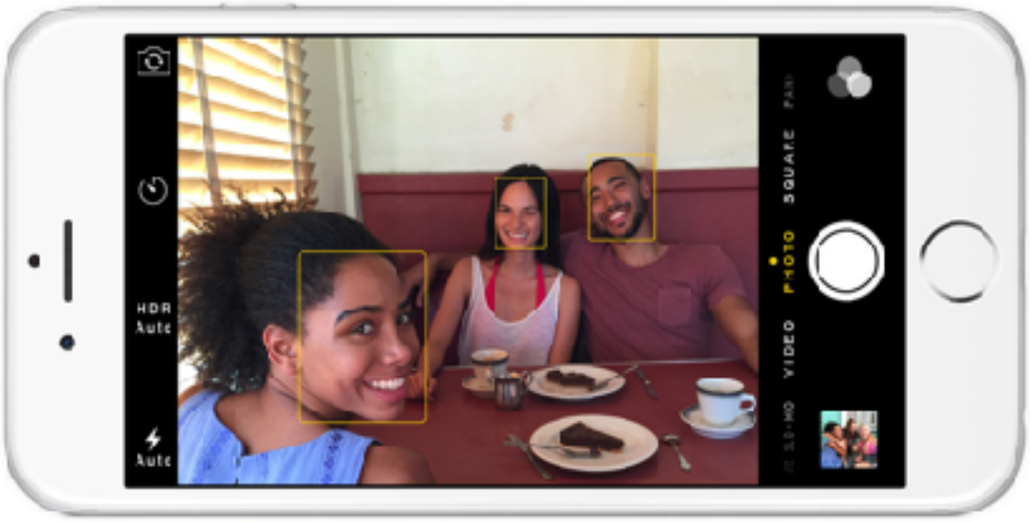
Scott Wehrwein




OLYMPUS
MANUAL ADAPTER
OM-SYSTEM
MC AUTO-S 1:1.8 f=50mm
Japan


1 Hour Photo
CVS/P

Computer Vision: Familiar Examples



In-Camera Face Detection

Autonomous Driving

Panorama Stitching

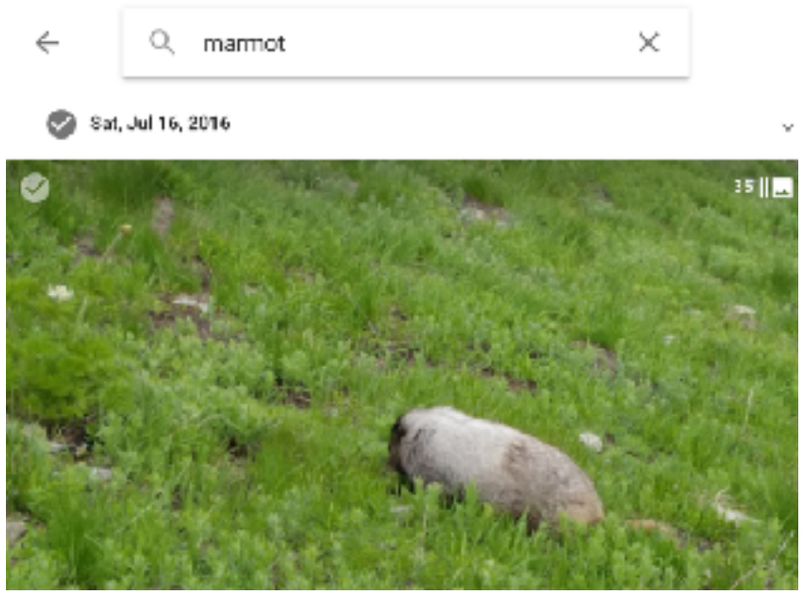
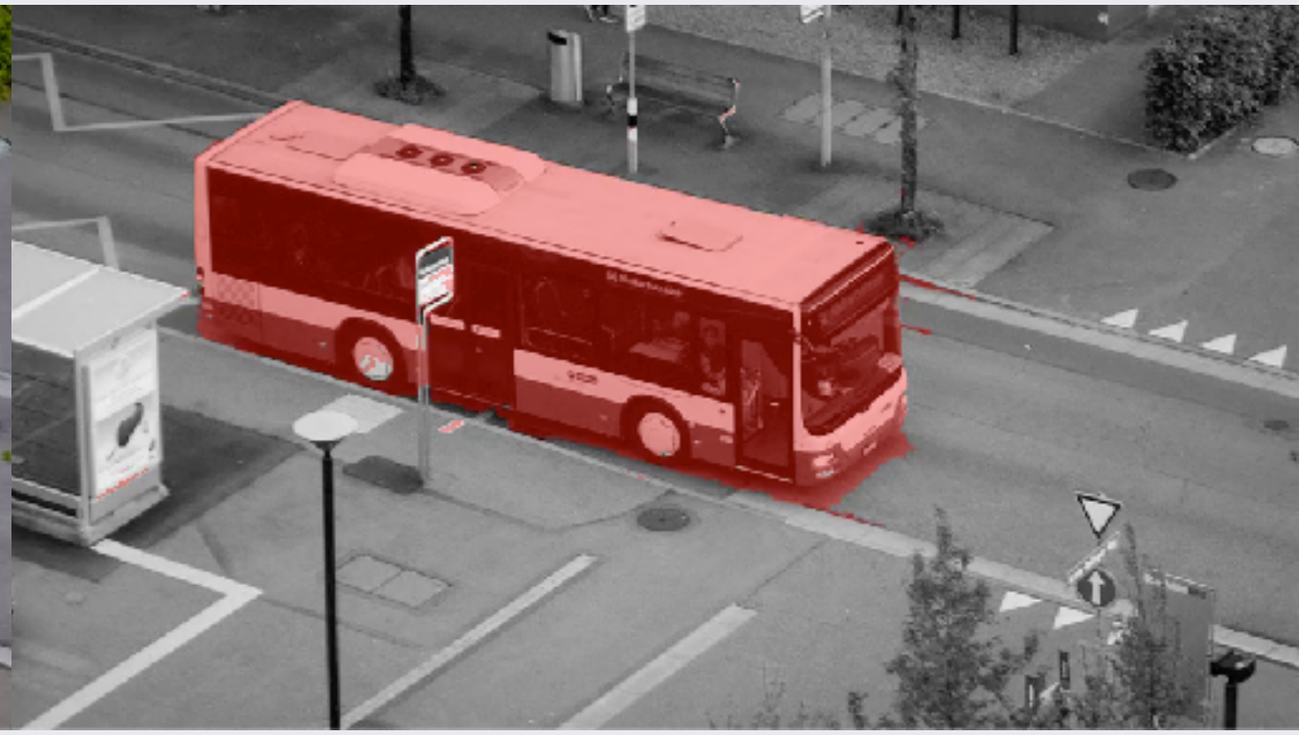


Image Search

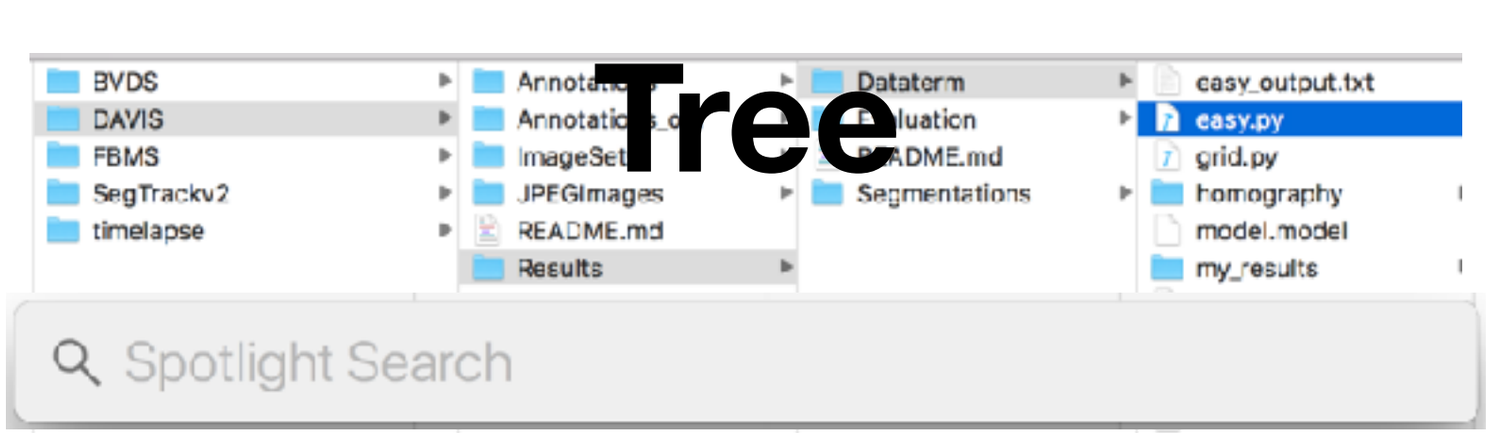



Sage Chapel
147 Ho Plaza

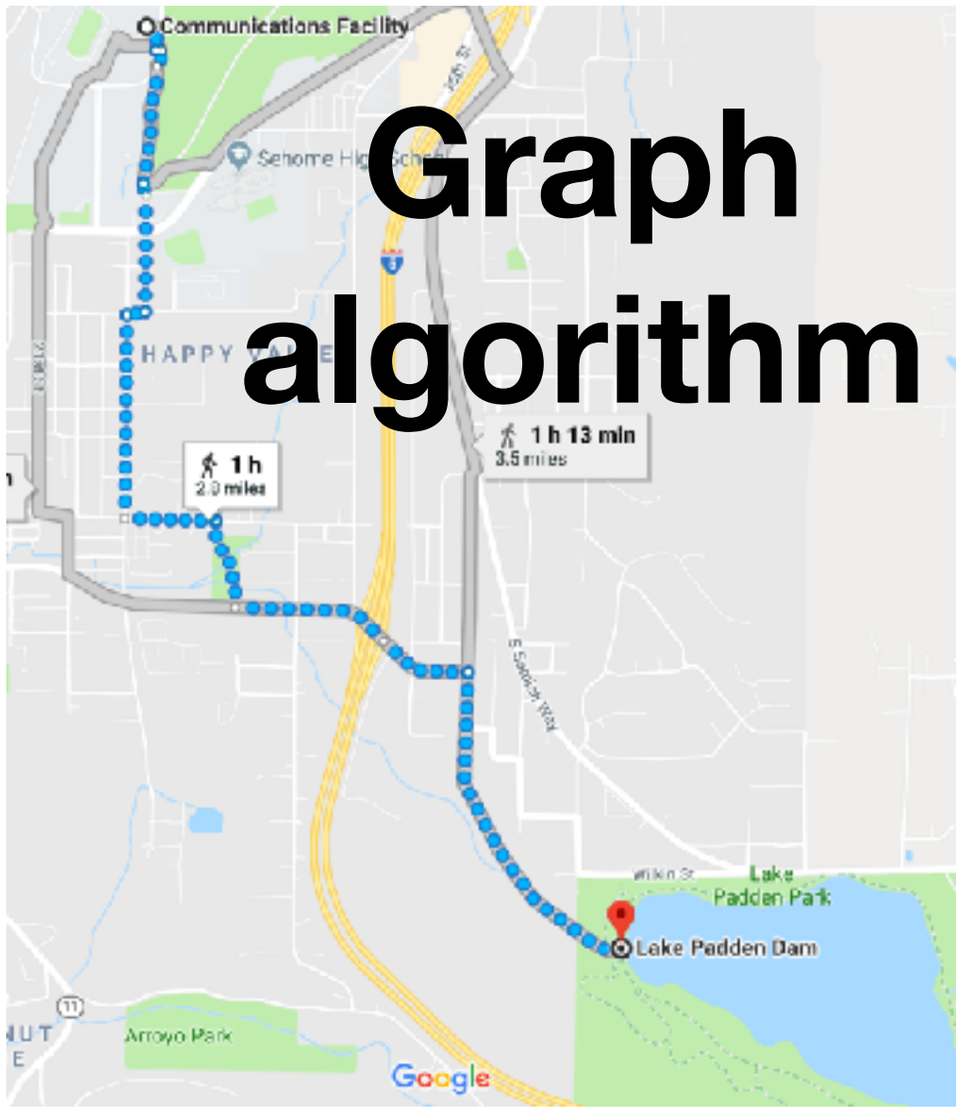
Data Structures: Why?

Graph



Tree

Hash table



Graph algorithm

Syllabus Overview

Course website:

https://facultyweb.cs.wwu.edu/~wehrwes/courses/csci241_18f

Interface vs Implementation

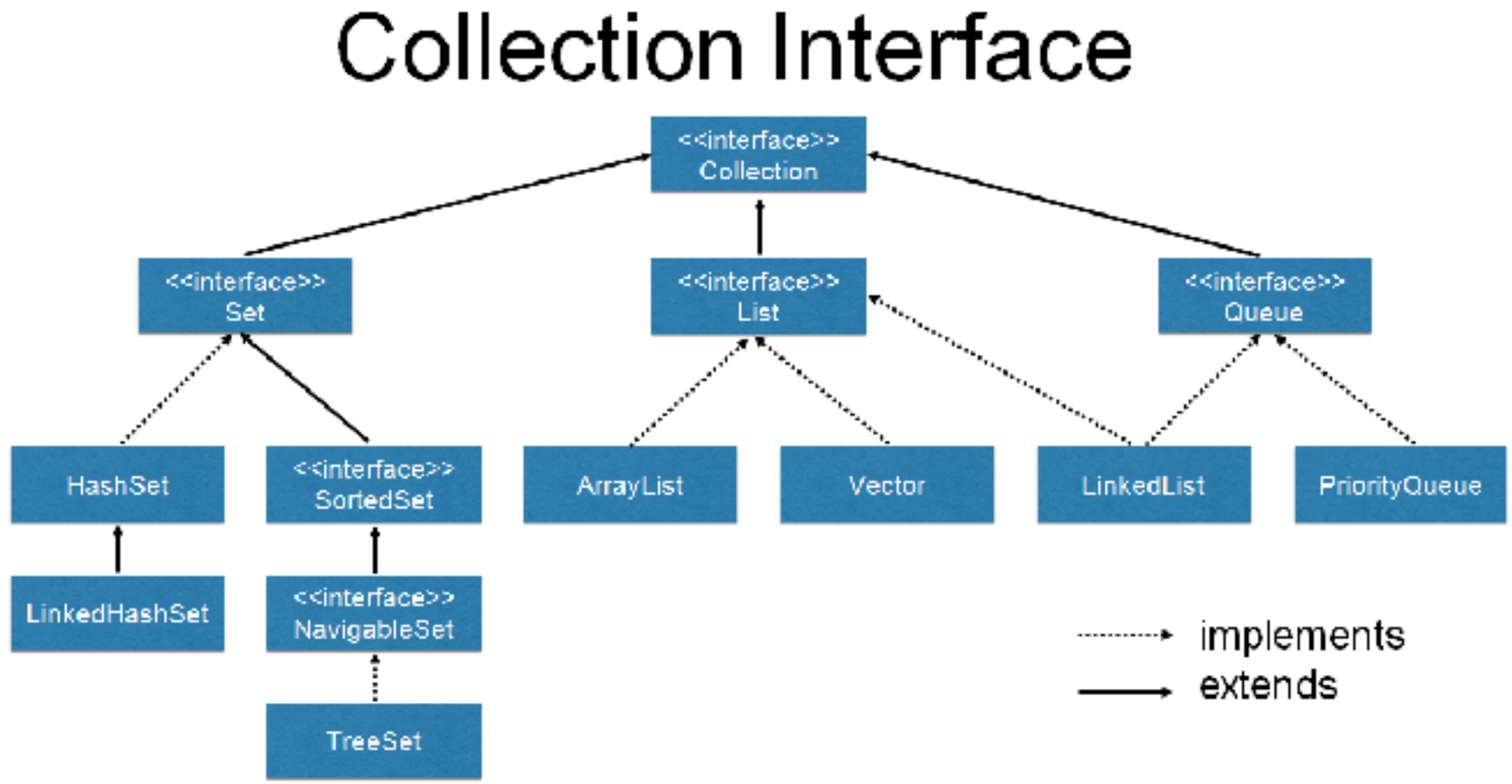
What the operations do

An abstract data type specifies only **interface**, not **implementation**

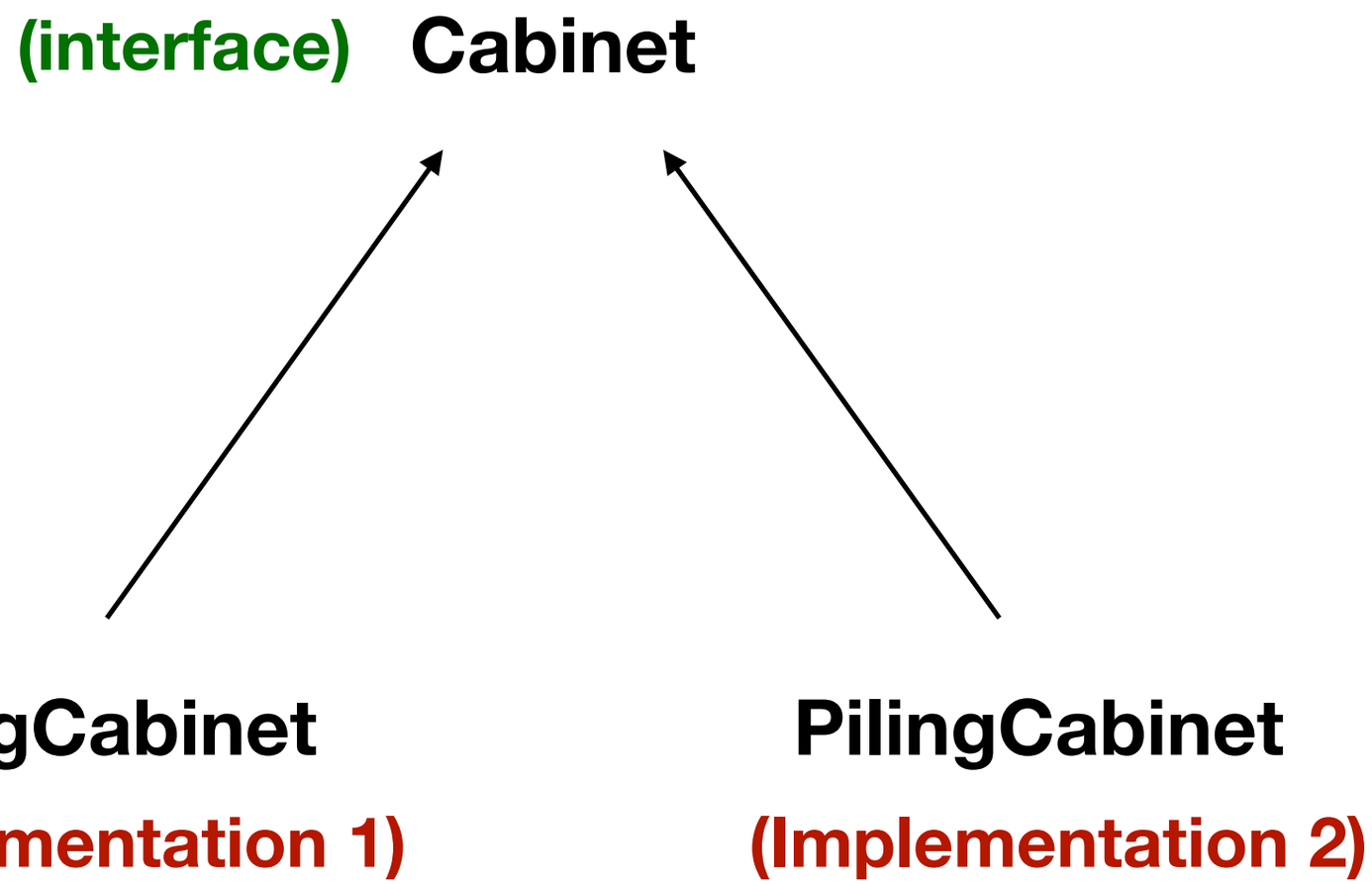
How they are accomplished

Abstract Data Types: Examples

- Set
- Tree
- Graph
- Map
- Priority Queue



Interface vs Implementation: Example



Interface vs Implementation: Example

Interface

Cabinet:

- Contains(item) - returns true iff item is in the cabinet
- Add(item) - adds item to the cabinet
- Remove(item) - removes item from the cabinet if it exists

Implementation

FilingCabinet implements Cabinet:

Contains(item):

```
look up drawer by first letter range
find folder by first letter
search folder for item
return true if item is found, false otherwise
```

Comparing Implementations

class FilingCabinet:

- Contains(item):

```
look up drawer by first letter range
find folder by first letter
search folder for item
return true if item is found, false otherwise
```

class PilingCabinet:

- Contains(item):

```
for each drawer:
  exhaustively search drawer
  if found, return true
return false
```

Comparing Implementations

class FilingCabinet:

- Add(item):

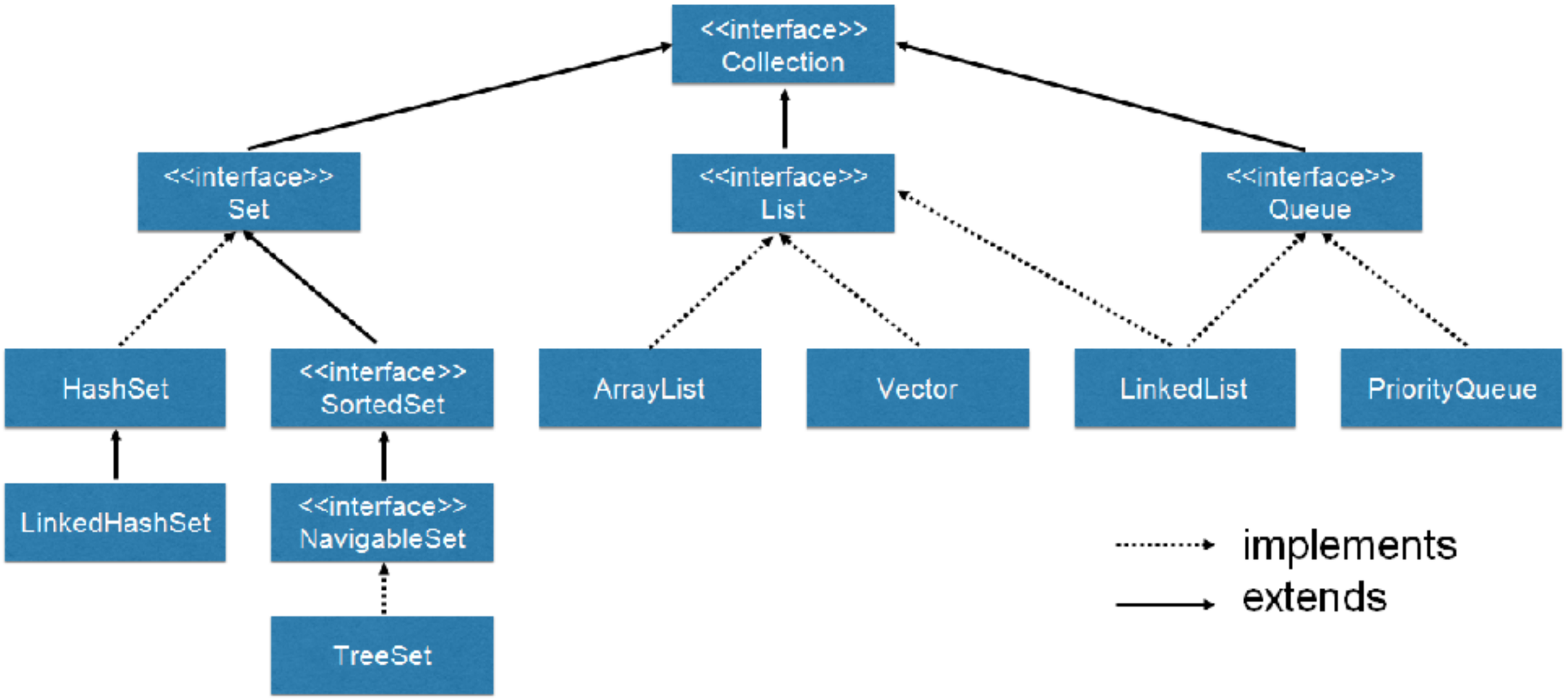
```
look up drawer by first letter range
find folder by first letter
insert item into folder
```

class PilingCabinet:

- Add(item):

```
open random drawer
insert item into drawer
```

Collection Interface



Is an array an ADT?

Runtime Analysis: Why we care

Runtime comparison of list implementations:

Class:	ArrayList	LinkedList
Backing storage:	array	chained nodes
add(i, val)	slow	slow
add(0, val)	slow	fast
add(n, val)	fast	fast
get(i)	fast	slow
get(0)	fast	fast
get(n)	fast	fast

Runtime Analysis: How should we measure?

How many ways can you think of to describe the runtime of an algorithm?

```
public int findMax(int[] a) {
  int currentMax = a[0];
  for (int i = 1; i < a.length; i++) {
    if (currentMax < a[i]) {
      currentMax = a[i];
    }
  }
  return currentMax;
}
```

Runtime Analysis: How should we measure?

How about metrics that are **invariant** to

- Length of the array a?
- How fast your computer is?

```
public int findMax(int[] a) {
  int currentMax = a[0];
  for (int i = 1; i < a.length; i++) {
    if (currentMax < a[i]) {
      currentMax = a[i];
    }
  }
  return currentMax;
}
```

“Primitive” Operations

Things the computer can do in a “fixed” amount of time.

“fixed” - doesn’t depend on the input size (n)

A non-exhaustive list:

- Get or set the value of a variable or array location
- Evaluate a simple expression
- Return from a method

Analyzing Runtime

```
public int findMax(int[] a) {
   int currentMax = a[0];

   for (int i = 1; i < a.length; i++) {
      if (currentMax < a[i]) {

         currentMax = a[i];
      }
   }
   return currentMax;
}
```

Analyzing Runtime

```
public int findMax(int[] a) {
   int currentMax = a[0];
                    set      eval, get       eval, set
   for (int i = 1; i < a.length; i++) {
                          eval, get
      if (currentMax < a[i]) {
                          set, get
         currentMax = a[i];
      }
   }     return
   return currentMax;
}
```

Analyzing Runtime

```
public int findMax(int[] a) {
   int currentMax = a[0];
                  set       eval, get       eval, set
   for (int i = 1; i < a.length; i++) {
                     eval, get
      if (currentMax < a[i]) {
                        set, get
         currentMax = a[i];
      }
   }     return
   return currentMax;
}
```

Let N = `a.length`. How many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
   int currentMax = a[0];
   for (int i = 1; i < a.length; i++) {
      if (currentMax < a[i]) {
         currentMax = a[i];
      }
   }
   return currentMax;
}
```

get, set 1

1 2(N-1)

set eval, get eval, set 2(N-1)

eval, get 2(N-1)

set, get

1

return

Let N = `a.length`. How many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
   int currentMax = a[0];
   for (int i = 1; i < a.length; i++) {
      if (currentMax < a[i]) {
         currentMax = a[i];
      }
   }
   return currentMax;
}
```

get, set 1

1 2(N-1)

set eval, get eval, set 2(N-1)

eval, get 2(N-1)

set, get ????

1

return

Let N = `a.length`. How many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
   int currentMax = a[0];
   for (int i = 1; i < a.length; i++) {
      if (currentMax < a[i]) {
         currentMax = a[i];
      }
   }
   return currentMax;
}
```

get, set 1

1

2(N-1)

set

eval, get

eval, set 2(N-1)

eval, get 2(N-1)

set, get ????

1

return

Let N = `a.length`. AT MOST how many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
   int currentMax = a[0];
   for (int i = 1; i < a.length; i++) {
      if (currentMax < a[i]) {
         currentMax = a[i];
      }
   }
   return currentMax;
}
```

get, set 1

1 set

2(N-1) eval, get

eval, set 2(N-1)

eval, get 2(N-1)

set, get 2(N-1)

1 return

Let N = `a.length`. AT MOST how many times does each primitive operation happen?

Analyzing Runtime

```
public int findMax(int[] a) {
   int currentMax = a[0];
   for (int i = 1; i < a.length; i++) {
      if (currentMax < a[i]) {
         currentMax = a[i];
      }
   }
   return currentMax;
}
```

get, set 1

1 2(N-1)

set eval, get eval, set 2(N-1)

eval, get 2(N-1)

set, get 2(N-1)

1

return

Total: 8N-5