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
Computer Vision: Familiar Examples

In-Camera Face Detection

Autonomous Driving

Panorama Stitching

Image Search
Data Structures: Why?

Graph

Tree

Hash table
Syllabus Overview

Course website:

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

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

**What** the operations do

**How** they are accomplished
Abstract Data Types: Examples

- Set
- Tree
- Graph
- Map
- Priority Queue
Interface vs Implementation: Example

(interface) Cabinet

FilingCabinet (Implementation 1)

PilingCabinet (Implementation 2)
Interface vs Implementation: Example

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

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
Is an array an ADT?
# Runtime Analysis: Why we care

## Runtime comparison of list implementations:

<table>
<thead>
<tr>
<th>Class:</th>
<th>ArrayList</th>
<th>LinkedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backing storage:</td>
<td>array</td>
<td>chained nodes</td>
</tr>
<tr>
<td><code>add(i, val)</code></td>
<td>slow</td>
<td>slow</td>
</tr>
<tr>
<td><code>add(0, val)</code></td>
<td>slow</td>
<td>fast</td>
</tr>
<tr>
<td><code>add(n, val)</code></td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td><code>get(i)</code></td>
<td>fast</td>
<td>slow</td>
</tr>
<tr>
<td><code>get(0)</code></td>
<td>fast</td>
<td>fast</td>
</tr>
<tr>
<td><code>get(n)</code></td>
<td>fast</td>
<td>fast</td>
</tr>
</tbody>
</table>
Runtime Analysis:
How should we measure?

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

```java
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?

```java
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
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;
}
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

```java
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;
}
```

Let $N = a.length$. How many times does each primitive operation happen?
public int findMax(int[] a) {
    int currentMax = a[0]; get, set 1
    1
    set 2(N-1) 2(N-1)
    eval, get 2(N-1)
    for (int i = 1; i < a.length; i++) {
        eval, get 2(N-1)
        if (currentMax < a[i]) {
            set, get
            currentMax = a[i];
        }
    }
    let N = a.length. How many times does each primitive operation happen?
    return return currentMax;
public int findMax(int[] a) {
    int currentMax = a[0]; // get, set 1
    for (int i = 1; i < a.length; i++) {
        if (currentMax < a[i]) { // eval, get 2(N-1)
            set, get ????
            currentMax = a[i]; // set 1
        }
    }
    return currentMax; // return
}

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

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;
}

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

1. get, set
2. set
3. eval, get
4. eval, set
5. for
6. if
7. set, get
8. return
```java
public int findMax(int[] a) {
    int currentMax = a[0]; get, set 1
    for (int i = 1; i < a.length; i++) {
        eval, get 2(N-1)
        if (currentMax < a[i]) {
            set, get 2(N-1)
            currentMax = a[i];
        }
    }
    return currentMax; 1
}
```

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

public int findMax(int[] a) {
    int currentMax = a[0]; \textcolor{red}{\textbf{get, set}} \quad 1
    \textcolor{green}{\textbf{set}} \quad 1 \quad \textcolor{blue}{\textbf{2(N-1)}} \quad \textcolor{purple}{\textbf{eval, get}} \quad \textcolor{blue}{\textbf{2(N-1)}}
    for (int i = 1; i < a.length; i++) {
        \textcolor{blue}{\textbf{eval, get}} \quad \textcolor{blue}{\textbf{2(N-1)}}
        if (currentMax < a[i]) {
            \textcolor{green}{\textbf{set, get}} \quad \textcolor{blue}{\textbf{2(N-1)}}
            currentMax = a[i];
        }
    }
    \textcolor{red}{\textbf{return}} \quad 1
    return currentMax;
} \textcolor{red}{\textbf{Total: 8N-5}}