An Introduction to Data Structures

NullPointerException

An Introduction to Data Structures

Trailer

Introduction

Introduction

Put slideshow of
all slides here

Introduction

Introduction

This Series

Introduction

This Series

What they are

Introduction

This Series

What they are

The Different Types

Introduction

This Series

What they are

The Different Types

How we can use them

Introduction

- This series will be a general overview of data structures
  - Won't be confined to one specific language
  - Will require a basic understanding of Computer Science

Introduction

- This series will be a general overview of data structures
  - Won't be confined to one specific language
  - Will require a basic understanding of Computer Science

Introduction

- This series will be a general overview of data structures
  - Won't be confined to one specific language
  - Will require a basic understanding of Computer Science

Introduction

- This series will be a general overview of data structures
  - Won't be confined to one specific language
  - Will require a basic understanding of Computer Science

Introduction

- There are a few housekeeping items I want to go over beforehand
  - Could help enhance your learning experience

Introduction

- There are a few housekeeping items I want to go over beforehand
  - Could help enhance your learning experience

Introduction - Timestamps

- In the description will be Timestamps for each major topic covered
  - Also Timestamps for each smaller section contained within those topics
  - Feel free to skip around

Time Stamps

Sectioned
Time Stamps

Introduction - Script and Visuals

- The **Script** and **Visuals** used for this series are also linked in the description below

For those staying though, there are just a few things I would like to mention.

Firstly, in the description you will find Time Stamps for each major topic covered in this video, as well as TimeStamps taking you to every smaller section contained within those topics. So please feel free to skip around if you are already comfortable with one topic or only interested in a certain data structure.

Next, we've decided to include the script and visuals used for this series also in description below. That way you can follow along if you like, or simply read the script if my soothing voice isn't your style.

Introduction - References + Research

- We'll also be including the references and research materials used to write the script for each topic in the description below
  - A different way of explaining things
  - Extra supplemental material

References:

https://www.geeksforgeeks.org/data-structures/

https://towardsdatascience.com/8-common-data-structures-every-programmer-must-know-171acf6a1a42?gi=999b39731c9f

https://www.freecodecamp.org/news/the-top-data-structures-you-should-know-for-your-next-coding-interview-36af0831f5e3/

Introduction - Questions

- If you have any questions throughout the series please leave them in the comments below
  - I'll try to respond as many as possible

Introduction - Shameless Plug

- If you end up liking the video, check out our personal channel **NullPointerException**

Introduction - What are Data Structures?

- A Data Structure...
  - A way to store, organize, and manage information (or data) in a way that allows you the programmer to easily access or modify the values within them

Introduction - What are Data Structures?

- A **Data Structure...**
  - A way to **store**, organize, and **manage** information (or data) in a way that allows you the programmer to easily **access** or **modify** the values within them

Introduction - What are Data Structures?

- A **Data Structure...**
  - A way to **store**, organize, and **manage** information (or data) in a way that allows you the programmer to easily **access** or **modify** the values within them

Introduction - What are Data Structures?

- A **Data Structure...**
  - A way to **store**, organize, and **manage** information (or data) in a way that allows you the programmer to easily **access** or **modify** the values within them

Introduction - What are Data Structures?

- A **Data Structure...**
  - A way to **store**, organize, and **manage** information (or data) in a way that allows you the programmer to easily **access** or **modify** the values within them

Introduction - What are Data Structures?

- A Data Structure...
  - A way to store, organize, and manage information (or data) in a way that allows you the programmer to easily access or modify the values within them

Introduction - What are Data Structures?

- A Data Structure...
  - A way to store, organize, and manage information (or data) in a way that allows you the programmer to easily access or modify the values within them

Data Structures

Introduction - What are Data Structures?

- A **Data Structure...**
  - A way to **store**, organize, and **manage** information (or data) in a way that allows you the programmer to easily **access** or **modify** the values within them

Data Structures

Algorithms

Introduction - What are Data Structures?

- A **Data Structure...**
  - A way to **store**, organize, and **manage** information (or data) in a way that allows you the programmer to easily **access** or **modify** the values within them

Introduction - What are Data Structures?

The GOAL of a data structure

Introduction - What are Data Structures?

The GOAL of a data structure

Store Information

Introduction - What are Data Structures?

The GOAL of a data structure

Store Information

Access and Manipulate that Information

Introduction - What are Data Structures?

- If you have a basic understanding of programming, you probably know about a few Data Structures already
  - Arrays and ArrayLists

Introduction - What are Data Structures?

- If you have a basic understanding of programming, you probably know about a few Data Structures already
  - Arrays and ArrayLists

Introduction - What are Data Structures?

- If you have a basic understanding of programming, you probably know about a few Data Structures already
  - Arrays and ArrayLists

Introduction - What are Data Structures?

- If you have a basic understanding of programming, you probably know about a few Data Structures already
  - Arrays and ArrayLists

Introduction - What are Data Structures?

- <u>Basic Data Structures</u>
  - Password Databases
  - Online Directories
- Advanced Data Structures
  - Undo/Redo Function
  - Spell Check
  - Text Searching

Introduction - What are Data Structures?

- **Basic Data Structures**
  - Password Databases
  - Online Directories
- **Advanced Data Structures**
  - Undo/Redo Function
  - Spell Check
  - Text Searching

Introduction - What are Data Structures?

- **Basic Data Structures**
  - Password Databases
  - Online Directories
- **Advanced Data Structures**
  - Undo/Redo Function
  - Spell Check
  - Text Searching

Introduction - What are Data Structures?

- **Basic Data Structures**
  - Password Databases
  - Online Directories
- **Advanced Data Structures**
  - Undo/Redo Function
  - Spell Check
  - Text Searching

Introduction - What are Data Structures?

- **Basic Data Structures**
  - Password Databases
  - Online Directories
- **<u>Advanced Data Structures</u>**
  - Undo/Redo Function
  - Spell Check
  - Text Searching

Introduction - What are Data Structures?

- **Basic Data Structures**
  - Password Databases
  - Online Directories
- **Advanced Data Structures**
  - Undo/Redo Function
  - Spell Check
  - Text Searching

Introduction - What are Data Structures?

- **Basic Data Structures**
  - Password Databases
  - Online Directories
- **<u>Advanced Data Structures</u>**
  - Undo/Redo Function
  - Spell Check
  - Text Searching

Introduction - What are Data Structures?

- **Basic Data Structures**
  - Password Databases
  - Online Directories
- **<u>Advanced Data Structures</u>**
  - Undo/Redo Function
  - Spell Check
  - Text Searching

Introduction - Series Overview

- We'll Start with efficiency
  - The metrics used to judge the speed and efficiency of different data structures

Introduction - Series Overview

- The Basics
  - Arrays and ArrayLists

Introduction - Series Overview

- The **Basics**
  - **Arrays** and **ArrayLists**

Arrays

Introduction - Series Overview

- The **Basics**
  - **Arrays** and **ArrayLists**

Arrays

ArrayLists

Introduction - Series Overview

- The Basics
  - Arrays and ArrayLists

Introduction - Series Overview

- The **Basics**
  - **Arrays** and **ArrayLists**

Introduction - Series Overview

- **Intermediate Data Structures**
  - A little more **complicated** than the basics
  - Have **special attributes** which make them stand out

Introduction - Series Overview

- **Intermediate Data Structures**
  - A little more complicated than the basics
  - Have special attributes which make them stand out

Stacks

Introduction - Series Overview

- **Intermediate Data Structures**
  - A little more **complicated** than the basics
  - Have **special attributes** which make them stand out

Stacks

Queues

Introduction - Series Overview

- **Intermediate Data Structures**
  - A little more **complicated** than the basics
  - Have **special attributes** which make them stand out

Introduction - Series Overview

- **Intermediate Data Structures**
  - A little more complicated than the basics
  - Have special attributes which make them stand out

LinkedLists

Introduction - Series Overview

- **Intermediate Data Structures**
  - A little more **complicated** than the basics
  - Have **special attributes** which make them stand out

LinkedLists

Doubly-LinkedLists

Introduction - Series Overview

- **Intermediate Data Structures**
  - A little more **complicated** than the basics
  - Have **special attributes** which make them stand out

Introduction - Series Overview

- **Intermediate Data Structures**
  - A little more complicated than the basics
  - Have special attributes which make them stand out

Dictionaries

Introduction - Series Overview

- **Intermediate Data Structures**
  - A little more complicated than the basics
  - Have special attributes which make them stand out

Dictionaries

Hash-Tables

Introduction - Series Overview

- **Tree-Based Data Structures**
  - Less **linear,** more **abstract**

Introduction - Series Overview

- **Tree-Based Data Structures**
  - Less **linear,** more **abstract**

Trees

Introduction - Series Overview

- **Tree-Based Data Structures**
  - Less **linear,** more **abstract**

Trees

Tries

Introduction - Series Overview

- **Tree-Based Data Structures**
  - Less **linear,** more **abstract**

Introduction - Series Overview

- **Tree-Based Data Structures**
  - Less **linear,** more **abstract**

Heaps

Introduction - Series Overview

- **Tree-Based Data Structures**
  - Less **linear,** more **abstract**

Heaps

Graphs

Introduction - Series Overview

Introduction - Series Overview

Arrays

Introduction - Series Overview

Arrays

ArrayLists

Introduction - Series Overview

Arrays

ArrayLists

Stacks

Introduction - Series Overview

Arrays

ArrayLists

Stacks

Queues

Introduction - Series Overview

Arrays

ArrayLists

Stacks

Queues

LinkedLists

Introduction - Series Overview

- Arrays
- ArrayLists
- Stacks
- Queues
- LinkedLists
- Doubly-LinkedLists

Introduction - Series Overview

- Arrays
- ArrayLists
- Stacks
- Queues
- LinkedLists
- Doubly-LinkedLists
- Dictionaries

Introduction - Series Overview

- Arrays
- ArrayLists
- Stacks
- Queues
- LinkedLists
- Doubly-LinkedLists
- Dictionaries
- Hash-Tables

Introduction - Series Overview

- Arrays
- ArrayLists
- Stacks
- Queues
- LinkedLists
- Doubly-LinkedLists
- Dictionaries
- Hash-Tables
- Trees

Introduction - Series Overview

- Arrays
- ArrayLists
- Stacks
- Queues
- LinkedLists
- Doubly-LinkedLists
- Dictionaries
- Hash-Tables
- Trees
- Tries

Introduction - Series Overview

- Arrays
- ArrayLists
- Stacks
- Queues
- LinkedLists
- Doubly-LinkedLists
- Dictionaries
- Hash-Tables
- Trees
- Tries
- Heaps

Arrays

ArrayLists

Stacks

Queues

LinkedLists

Doubly-LinkedLists

Dictionaries

Hash-Tables

Trees

Tries

Heaps

Graphs

Introduction - Series Overview

Introduction - Series Overview

Efficiency

Introduction - Series Overview

Efficiency

Basic Data Structures

Introduction - Series Overview

Efficiency

Basic Data Structures

Intermediate Data Structures

Introduction - Series Overview

Efficiency

Basic Data Structures

Intermediate Data Structures

Tree-based Data Structures

An Introduction to Data Structures

Measuring Efficiency with BigO Notation

Measuring Efficiency with BigO Notation - Introduction

- We want a quantifiable way to measure how efficient certain data structures are at different tasks we might ask of it
  - Searching through
  - Modifying
  - Accessing

Measuring Efficiency with BigO Notation - Introduction

- We want a quantifiable way to measure how efficient certain data structures are at different tasks we might ask of it
  - Searching through
  - Modifying
  - Accessing

Measuring Efficiency with BigO Notation - Introduction

- We want a quantifiable way to measure how efficient certain data structures are at different tasks we might ask of it
  - Searching through
  - Modifying
  - Accessing

Measuring Efficiency with BigO Notation - Introduction

- We want a quantifiable way to measure how efficient certain data structures are at different tasks we might ask of it
  - Searching through
  - Modifying
  - Accessing

Measuring Efficiency with BigO Notation - Introduction

- The **industry standard** for this kind of measurement
  - **BigO Notation**

Measuring Efficiency with BigO Notation - Introduction

WHAT IS IT?

- ...y standard for ...is kind of measurement
  - BigO Notation

Measuring Efficiency with BigO Notation - Introduction

WHAT IS IT?

- ...y standard for
  ...this kind of measurement
  - BigO Notation

HOW DO WE MEASURE IT?

Measuring Efficiency with BigO Notation - Introduction

- **BigO Notation**
  - A way to basically "Score" a data structure based on 4 criteria
  - The most common functions you might want from a data structure
    - Accessing elements
    - Searching for an element
    - Inserting an element
    - Deleting an element

Measuring Efficiency with BigO Notation - Introduction

- **BigO Notation**
  - A way to basically "Score" a data structure based on 4 criteria
  - The most common functions you might want from a data structure
    - Accessing elements
    - Searching for an element
    - Inserting an element
    - Deleting an element

Measuring Efficiency with BigO Notation - Introduction

- **BigO Notation**
  - A way to basically "**Score**" a data structure based on **4 criteria**
  - The most **common functions** you might want from a data structure
    - Accessing elements
    - **Searching for an element**
    - Inserting an element
    - Deleting an element

Measuring Efficiency with BigO Notation - Introduction

- **BigO Notation**
  - A way to basically "Score" a data structure based on 4 criteria
  - The most common functions you might want from a data structure
    - Accessing elements
    - Searching for an element
    - Inserting an element
    - Deleting an element

Measuring Efficiency with BigO Notation - Introduction

- **BigO Notation**
  - A way to basically "Score" a data structure based on 4 criteria
  - The most common functions you might want from a data structure
    - Accessing elements
    - Searching for an element
    - Inserting an element
    - Deleting an element

Measuring Efficiency with BigO Notation - Introduction

- By measuring how **efficiently** a data structure can do these 4 things
  - We can create a "**report card**"

Measuring Efficiency with BigO Notation - Introduction

- By measuring how **efficiently** a data structure can do these 4 things
  - We can create a "**report card**"

Accessing

Measuring Efficiency with BigO Notation - Introduction

- By measuring how **efficiently** a data structure can do these 4 things
  - We can create a "**report card**"

Accessing

Searching

Measuring Efficiency with BigO Notation - Introduction

- By measuring how **efficiently** a data structure can do these 4 things
  - We can create a "**report card**"

Accessing

Searching

Inserting

Measuring Efficiency with BigO Notation - Introduction

- By measuring how **efficiently** a data structure can do these 4 things
  - We can create a "**report card**"

Accessing

Searching

Inserting

Deleting

Measuring Efficiency with BigO Notation - Introduction

- By measuring how **efficiently** a data structure can do these 4 things
  - We can create a "**report card**"

Accessing

Searching

Inserting

Deleting

Measuring Efficiency with BigO Notation - Introduction

- By measuring how **efficiently** a data structure can do these 4 things
  - We can create a "**report card**"

Accessing

Searching

Inserting

Deleting

Measuring Efficiency with BigO Notation - Introduction

- By measuring how **efficiently** a data structure can do these 4 things
  - We can create a "**report card**"

Accessing

Searching

Inserting

Deleting

Measuring Efficiency with BigO Notation - Introduction

- By measuring how **efficiently** a data structure can do these 4 things
  - We can create a "**report card**"

Accessing

Searching

Inserting

Deleting

Measuring Efficiency with BigO Notation - Time Complexity Equations

- A **Time Complexity Equation** works by inserting the **size** of the data-set as an integer **n**, and returning the number of **operations** that need to be conducted by the computer before the function can finish

Measuring Efficiency with BigO Notation - Time Complexity Equations

- A **Time Complexity Equation** works by inserting the **size** of the data-set as an integer **n**, and returning the number of **operations** that need to be conducted by the computer before the function can finish

N = The Size of the Data Set (Amount of elements contained within the Data Structure)

Measuring Efficiency with BigO Notation - Time Complexity Equations

- A **Time Complexity Equation** works by inserting the **size** of the data-set as an integer **n**, and returning the number of **operations** that need to be conducted by the computer before the function can finish

$$N = 10$$

Measuring Efficiency with BigO Notation - Time Complexity Equations

- A **Time Complexity Equation** works by inserting the **size** of the data-set as an integer **n**, and returning the number of **operations** that need to be conducted by the computer before the function can finish

$$N = 10$$

Accessing Equation

Measuring Efficiency with BigO Notation - Time Complexity Equations

- A **Time Complexity Equation** works by inserting the **size** of the data-set as an integer **n**, and returning the number of **operations** that need to be conducted by the computer before the function can finish

$$N = 10$$

Accessing Equation

Searching Equation

Measuring Efficiency with BigO Notation - Time Complexity Equations

- A **Time Complexity Equation** works by inserting the **size** of the data-set as an integer **n**, and returning the number of **operations** that need to be conducted by the computer before the function can finish

$$N = 10$$

Accessing Equation

Searching Equation

Inserting Equation

Measuring Efficiency with BigO Notation - Time Complexity Equations

- A **Time Complexity Equation** works by inserting the **size** of the data-set as an integer **n**, and returning the number of **operations** that need to be conducted by the computer before the function can finish

$$N = 10$$

- Accessing Equation
- Searching Equation
- Inserting Equation
- Deleting Equation

Measuring Efficiency with BigO Notation - Time Complexity Equations

- A **Time Complexity Equation** works by inserting the **size** of the data-set as an integer **n**, and returning the number of **operations** that need to be conducted by the computer before the function can finish

The Number of operations that need to be conducted by the computer before completion of that function

- Accessing Equation
- Searching Equation
- Inserting Equation
- Deleting Equation

Measuring Efficiency with BigO Notation - Time Complexity Equations

- We always use the worst-case scenario when judging these data structures

8 Operations

2 Operations

50 Operations

42 Operations

1898 Operations

Measuring Efficiency with BigO Notation - Time Complexity Equations

- We always use the worst-case scenario when judging these data structures

8 Operations

2 Operations

50 Operations

42 Operations

1898 Operations

Measuring Efficiency with BigO Notation - Time Complexity Equations

- We always use the worst-case scenario when judging these data structures

8 Operations

2 Operations

50 Operations

42 Operations

1898 Operations

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

A Function

Random Function()

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

A Function		Time Complexity
Random Function()	→	O(2)

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

A Function		Time Complexity
Random Function()	→	O(5)

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

Most of the time, our integer n, is going to have some adverse-effect on how many operations it takes

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

Most of the time, our integer n, is going to have some adverse-effect on how many operations it takes

Larger Data Set

Measuring Efficiency with BigO Notation - The Meaning of BigO

- It's called BigO notation because the syntax for the Time Complexity equations includes a BigO and then a set of parentheses
  - The parenthesis houses the function

Most of the time, our integer n, is going to have some adverse-effect on how many operations it takes

Larger Data Set → More Instructions

Measuring Efficiency with BigO Notation - Why BigO?

- We measure efficiency in # of operations performed because measuring by how long the function takes to run would be silly
  - Measuring by time is biased towards better hardware

Measuring Efficiency with BigO Notation - Why BigO?

- We measure efficiency in # of operations performed because measuring by how long the function takes to run would be silly
  - Measuring by time is biased towards better hardware

Measuring Efficiency with BigO Notation - Why BigO?

- We measure efficiency in # of operations performed because measuring by how long the function takes to run would be silly
  - Measuring by time is biased towards better hardware

Measuring Efficiency with BigO Notation - Why BigO?

- We measure efficiency in # of operations performed because measuring by how long the function takes to run would be silly
  - Measuring by time is biased towards better hardware

›

Speed

Measuring Efficiency with BigO Notation - Why BigO?

- We measure efficiency in # of operations performed because measuring by how long the function takes to run would be silly
  - Measuring by time is biased towards better hardware

=

Operations

Measuring Efficiency with BigO Notation - Quick Recap

We measure efficiency based on 4 metrics

Measuring Efficiency with BigO Notation - Quick Recap

We measure efficiency based on 4 metrics

Accessing

Measuring Efficiency with BigO Notation - Quick Recap

We measure efficiency based on 4 metrics

Accessing

Searching

Measuring Efficiency with BigO Notation - Quick Recap

We measure efficiency based on 4 metrics

Accessing

Searching

Inserting

Measuring Efficiency with BigO Notation - Quick Recap

We measure efficiency based on 4 metrics

- Accessing
- Searching
- Inserting
- Deleting

Measuring Efficiency with BigO Notation - Quick Recap

We measure efficiency based on 4 metrics

Accessing

Searching

Inserting

Deleting

Modeled by an equation which takes in size of data-set (n) and returns number of operations needed to be performed by the computer to complete that task

Measuring Efficiency with BigO Notation - Quick Recap

We measure efficiency based on 4 metrics

Accessing

Searching

Inserting

Deleting

Modeled by an equation which takes in size of data-set (n) and returns number of operations needed to be performed by the computer to complete that task

What the data structure is good at, and what the data structure is bad at

Measuring Efficiency with BigO Notation - Quick Recap

- A Data Structures efficiency isn't the end all be all for deciding on which data structure to use in your program
  - Some have specific quarks or features which separate them

Measuring Efficiency with BigO Notation - Quick Recap

- A Data Structures efficiency isn't the end all be all for deciding on which data structure to use in your program
  - Some have specific quarks or features which separate them

Accessing

Searching

Inserting

Deleting

Measuring Efficiency with BigO Notation - Quick Recap

- A Data Structures efficiency isn't the end all be all for deciding on which data structure to use in your program
  - Some have specific quarks or features which separate them

Should not be the ONLY metric used

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- Let's dive straight into what the actual equations mean in terms of efficiency
  - 6 most common Time Complexity Equations

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- Let's dive straight into what the actual equations mean in terms of efficiency
  - 6 most common Time Complexity Equations

O(1)

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- Let's dive straight into what the actual equations mean in terms of efficiency
  - 6 most common Time Complexity Equations

O(1)

O(n)

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- Let's dive straight into what the actual equations mean in terms of efficiency
  - 6 most common Time Complexity Equations

O(1)

O(n)

O(log n)

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- Let's dive straight into what the actual equations mean in terms of efficiency
  - 6 most common Time Complexity Equations

O(1)	O(n)	O(log n)	O(n log n)

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- Let's dive straight into what the actual equations mean in terms of efficiency
  - 6 most common Time Complexity Equations

O(1)	O(n)	O(log n)	O(n log n)	$O(n^2)$

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- Let's dive straight into what the actual equations mean in terms of efficiency
  - 6 most common Time Complexity Equations

$O(1)$	$O(n)$	$O(\log n)$	$O(n \log n)$	$O(n^2)$	$O(2^n)$

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

Size of Data Set (N)
1

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

Operations Required	Size of Data Set (N)
1	1

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

Operations Required	Size of Data Set (N)
1	100

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

Operations Required	Size of Data Set (N)
1	1,000,000

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

Operations Required	Size of Data Set (N)
1	800 Quadrillion

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

When we graph Volume of Data vs # Of Instructions Required

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

When we graph Volume of Data vs # Of Instructions Required

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

When we graph Volume of Data vs # Of Instructions Required

Of Operations remains constant

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The absolute best a data structure can "score" on each criteria is O(1)
  - No matter what the size of your data set is, the task will be completed in a single instruction

When we graph Volume of Data vs # Of Instructions Required

Of Operations remains constant

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next fastest type of time complexity equation os O(log n)
  - Still provides fast completion time
  - Gets more efficient as the size of the data set increases

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next fastest type of time complexity equation os O(log n)
  - Still provides fast completion time
  - Gets more efficient as the size of the data set increases

Slower Than

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next fastest type of time complexity equation os O(log n)
  - Still provides fast completion time
  - Gets more efficient as the size of the data set increases

Slower Than

Faster Than

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next fastest type of time complexity equation os O(log n)
  - Still provides fast completion time
  - Gets more efficient as the size of the data set increases

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next fastest type of time complexity equation os O(log n)
  - Still provides fast completion time
  - Gets more efficient as the size of the data set increases

Skyrockets

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next fastest type of time complexity equation os O(log n)
  - Still provides fast completion time
  - Gets more efficient as the size of the data set increases

Increases More Slowly

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next fastest type of time complexity equation os O(log n)
  - Still provides fast completion time
  - Gets more efficient as the size of the data set increases

Increases Way More Slowly

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next fastest type of time complexity equation os **O(log n)**
  - Still provides **fast** completion time
  - Gets **more efficient** as the size of the data set increases

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next fastest type of time complexity equation os O(log n)
  - Still provides fast completion time
  - Gets more efficient as the size of the data set increases

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next common time complexity efficiency equation type that's going to come up is O(n)
  - The last of the "decent" equations

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next common time complexity efficiency equation type that's going to come up is O(n)
  - The last of the "decent" equations

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next common time complexity efficiency equation type that's going to come up is O(n)
  - The last of the "decent" equations

Size of Data Set (N)

10

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next common time complexity efficiency equation type that's going to come up is O(n)
  - The last of the "decent" equations

Operations Required	Size of Data Set (N)
10	10

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next common time complexity efficiency equation type that's going to come up is O(n)
  - The last of the "decent" equations

Operations Required	Size of Data Set (N)
50	50

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next common time complexity efficiency equation type that's going to come up is **O(n)**
  - The last of the "**decent**" equations

Operations Required	Size of Data Set (N)
1,000	1,000

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next type of equation that will come up is O(n log n)
  - The first which is relatively bad in terms of efficiency

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next type of equation that will come up is O(n log n)
  - The first which is relatively bad in terms of efficiency

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next type of equation that will come up is O(n log n)
  - The first which is relatively bad in terms of efficiency

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The next type of equation that will come up is O(n log n)
  - The first which is relatively bad in terms of efficiency

Slope Increases

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The last 2 types of equations are $O(n^2)$ and $O(2^n)$
  - Very bad in terms of efficiency
  - Exponential in structure

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The last 2 types of equations are $O(n^2)$ and $O(2^n)$
  - Very bad in terms of efficiency
  - Exponential in structure

Measuring Efficiency with BigO Notation - Types of Time Complexity Equations

- The last 2 types of equations are $O(n^2)$ and $O(2^n)$
  - Very bad in terms of efficiency
  - Exponential in structure

Measuring Efficiency with BigO Notation - Final Note on Time Complexity Equations

- Time Complexity Equations are NOT the only metric you should be using the gauge which data structure to use
  - Some provide other functionality that make them extremely useful

Measuring Efficiency with BigO Notation - Final Note on Time Complexity Equations

- Time Complexity Equations are NOT the only metric you should be using the gauge which data structure to use
  - Some provide other functionality that make them extremely useful

Some Data Structure

Measuring Efficiency with BigO Notation - Final Note on Time Complexity Equations

- Time Complexity Equations are NOT the only metric you should be using the gauge which data structure to use
  - Some provide other functionality that make them extremely useful

Measuring Efficiency with BigO Notation - Final Note on Time Complexity Equations

- Time Complexity Equations are NOT the only metric you should be using the gauge which data structure to use
  - Some provide other functionality that make them extremely useful

An Introduction to Data Structures

The Array

The Array - Introduction

- The array is a pretty common data structure taught in most programming classes...

The Array - Introduction

- The array is a pretty common data structure taught in most programming classes...
  - But today we'll be covering the Array more in-depth as a data structure

Time Complexity

The Array - Introduction

- The array is a pretty common data structure taught in most programming classes...
  - But today we'll be covering the Array more in-depth as a data structure

Time Complexity

Storage Methods

The Array - Introduction

- The array is a pretty common data structure taught in most programming classes...
  - But today we'll be covering the Array more in-depth as a data structure

Time Complexity

Storage Methods

Check Description to skip ahead

The Array - Array Basics

- An array is fundamentally a list of similar values
- Can be used to store anything
  - Usernames
  - High Scores
  - Prices
- Store values of the same type
  - Integer
  - String
  - Float

The Array - Array Basics

- An **array** is fundamentally a list of **similar** values
- Can be used to store **anything**
  - **Usernames**
  - High Scores
  - Prices
- Store values of the same **type**
  - Integer
  - String
  - Float

Today's Date is: 11.08.2012

Name	Login	Password
Treshon Adams	treshonadams661	lemon
Norah Alexander	norahalexander965	table
Tracy Allen	tracyallen533	lemon
Cory Armstrong	coryarmstrong776	potato
Aaron Axelman	aaronaxelman123	onion
Andrea Botwin	andreabotwin311	omelette
Carla Burton	carlaburton47	pepper
Devon Charles	devoncharles504	cherry
Lauren Conrad	laurenconrad801	sugar
Caroline Daly	carolinedaly65	raspberry
Ivan Denisovich	ivandenisovich9	prune
Sharon Douglas	sharondouglas64	candy
Sylvain DuPont	sylvaindupont658	grapefruit

The Array - Array Basics

- An **array** is fundamentally a list of **similar** values
- Can be used to store **anything**
  - Usernames
  - **High Scores**
  - Prices
- Store values of the same **type**
  - Integer
  - String
  - Float

Today's Date is: 11.08.2012

Name	Login	Password
Treshon Adams	treshonadams661	lemon
Norah Alexander	norahalexander965	table
Tracy Allen	tracyallen533	lemon
Cory Armstrong	coryarmstrong776	potato
Aaron Axelman	aaronaxelman123	onion
Andrea Botwin	andreabotwin311	omelette
Carla Burton	carlaburton47	pepper
Devon Charles	devoncharles504	cherry
Lauren Conrad	laurenconrad801	sugar
Caroline Daly	carolinedaly65	raspberry
Ivan Denisovich	ivandenisovich9	prune
Sharon Douglas	sharondouglas64	candy
Sylvain DuPont	sylvaindupont658	grapefruit

TOP 10 SCORES 2P

RANK	SCORE	RND	NAME	CHR
1ST	20000	16	PEN	
2ND	19000	15	DON	
3RD	18000	14	HBW	
4TH	17000	13	DJP	
5TH	16000	12	CWN	
6TH	15000	11	BDG	
7TH	14000	10	CBR	
8TH	13000	9	GMR	
9TH	12000	8	CDJ	
10TH	11000	7	ANB	

The Array - Array Basics

- An **array** is fundamentally a list of **similar** values
- Can be used to store **anything**
  - Usernames
  - High Scores
  - **Prices**
- Store values of the same **type**
  - Integer
  - String
  - Float

Today's Date is: 11.08.2012

Name	Login	Password
Treshon Adams	treshonadams661	lemon
Norah Alexander	norahalexander965	table
Tracy Allen	tracyallen533	lemon
Cory Armstrong	coryarmstrong776	potato
Aaron Axelman	aaronaxelman123	onion
Andrea Botwin	andreabotwin311	omelette
Carla Burton	carlaburton47	pepper
Devon Charles	devoncharles504	cherry
Lauren Conrad	laurenconrad801	sugar
Caroline Daly	carolinedaly65	raspberry
Ivan Denisovich	ivandenisovich9	prune
Sharon Douglas	sharondouglas64	candy
Sylvain DuPont	sylvaindupont658	grapefruit

TOP 10 SCORES 2P

RANK	SCORE	RND	NAME	CHR
1ST	20000	16	PEN	
2ND	19000	15	DON	
3RD	18000	14	HBW	
4TH	17000	13	DJP	
5TH	16000	12	CWN	
6TH	15000	11	BDG	
7TH	14000	10	CBR	
8TH	13000	9	GMR	
9TH	12000	8	CDJ	
10TH	11000	7	ANB	

The Array - Array Basics

- An array is fundamentally a list of similar values
- Can be used to store anything
  - Usernames
  - High Scores
  - Prices
- Store values of the same type
  - Integer
  - String
  - Float

The Array - Array Basics

- An array is fundamentally a list of similar values
- Can be used to store anything
  - Usernames
  - High Scores
  - Prices
- Store values of the same type
  - Integer
  - String
  - Float

The Array - Array Basics

- An array is fundamentally a list of similar values
- Can be used to store anything
  - Usernames
  - High Scores
  - Prices
- Store values of the same type
  - Integer
  - String
  - Float

The Array - Array Basics

- An **array** is fundamentally a list of **similar** values
- Can be used to store **anything**
  - Usernames
  - High Scores
  - Prices
- Store values of the same **type**
  - Integer
  - String
  - **Float**

The Array - Array Basics

- An array is fundamentally a list of similar values
- Can be used to store anything
  - Usernames
  - High Scores
  - Prices
- Store values of the same type
  - Integer
  - String
  - Float

The Array - Array Basics

- Every item in in the list of data is referred to as an "element"
  - The collective total of elements is the array

1	2	3	4	5	6	7	8	9	10

The Array - Array Basics

- Every item in in the list of data is referred to as an "element"
  - The collective total of elements is the array

The Array - Array Basics

- Every item in in the list of data is referred to as an "element"
  - The collective total of elements is the array

The Array - Array Basics

- Every item in in the list of data is referred to as an "element"
  - The collective total of elements is the array

The Array - Array Basics

- Every item in in the list of data is referred to as an "element"
  - The collective total of elements is the array

This is going to be true for almost all Data Structures we talk about, so keep this in mind

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

Type

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

Type

Size

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

Type

Size

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

Type

Size

The Array - Array Names

- A name is simply a name for the array
  - Used to reference and interact with it

The Array - Array Names

- A name is simply a **name** for the array
  - Used to **reference** and interact with it

Names	“John Smith”	“Gary Vee”	“David Lee”	“Adam Knox”

The Array - Array Names

- A name is simply a **name** for the array
  - Used to **reference** and interact with it

Names	"John Smith"	"Gary Vee"	"David Lee"	"Adam Knox"
Salaries	10,000	12,500	8,750	15,000

The Array - Array Names

- A name is simply a **name** for the array
  - Used to **reference** and interact with it

Names	"John Smith"	"Gary Vee"	"David Lee"	"Adam Knox"
Salaries	10,000	12,500	8,750	15,000

Search through "Salaries" Array

The Array - Array Names

- A name is simply a **name** for the array
  - Used to **reference** and interact with it

Names	"John Smith"	"Gary Vee"	"David Lee"	"Adam Knox"
Salaries	10,000	12,500	8,750	15,000

Search through "Salaries" Array

The Array - Array Names

- A name is simply a **name** for the array
  - Used to **reference** and interact with it

Names	"John Smith"	"Gary Vee"	"David Lee"	"Adam Knox"
Salaries	10,000	12,500	8,750	15,000

Search through "Salaries" Array

The Array - Array Names

- A name is simply a **name** for the array
  - Used to **reference** and interact with it

Names	"John Smith"	"Gary Vee"	"David Lee"	"Adam Knox"
Salaries	10,000	12,500	8,750	15,000

Search through "Names" Array

The Array - Array Names

- A name is simply a **name** for the array
  - Used to **reference** and interact with it

Names	"John Smith"	"Gary Vee"	"David Lee"	"Adam Knox"
Salaries	10,000	12,500	8,750	15,000

Search through "Names" Array

The Array - Array Names

- A name is simply a **name** for the array
  - Used to **reference** and interact with it

Names	"John Smith"	"Gary Vee"	"David Lee"	"Adam Knox"
Salaries	10,000	12,500	8,750	15,000

Search through "Names" Array

The Array - Parallel Arrays

- These are also examples of parallel arrays
  - 2 or more arrays which...
    - Contain the same number of elements
    - Have corresponding values in the same position

Names	“John Smith”	“Gary Vee”	“David Lee”	“Adam Knox”
Salaries	10,000	12,500	8,750	15,000

The Array - Parallel Arrays

- These are also examples of parallel arrays
  - 2 or more arrays which...
    - Contain the same number of elements
    - Have corresponding values in the same position

Names	"John Smith"	"Gary Vee"	"David Lee"	"Adam Knox"
Salaries	10,000	12,500	8,750	15,000

The Array - Parallel Arrays

- These are also examples of parallel arrays
  - 2 or more arrays which...
    - Contain the same number of elements
    - Have corresponding values in the same position

Names	"John Smith"	"Gary Vee"	"David Lee"	"Adam Knox"
Salaries	10,000	12,500	8,750	15,000

The Array - Parallel Arrays

- These are also examples of parallel arrays
  - 2 or more arrays which...
    - Contain the same number of elements
    - Have corresponding values in the same position

Names	“John Smith”	“Gary Vee”	“David Lee”	“Adam Knox”
Salaries	10,000	12,500	8,750	15,000

The Array - Parallel Arrays

- **Parallel arrays are extremely useful for storing differing types of data about the same entity**

Names	“John Smith”	“Gary Vee”	“David Lee”	“Adam Knox”
Salaries	10,000	12,500	8,750	15,000

The Array - Array Types

- An array's type
  - what type of information is stored or will be stored within that array
  - HAS to hold all the same type of information

The Array - Array Types

- An array's **type**
  - what type of information is stored or will be stored **within** that array
  - **HAS** to hold all the same type of information

Wrong	"Hello"	5	2	"World"

The Array - Array Types

- An array's **type**
  - what type of information is stored or will be stored **within** that array
  - **HAS** to hold all the same type of information

Wrong	~~"Hello"~~	~~[illegible]~~	~~2~~	"World"

The Array - Array Types

- An array's **type**
  - what type of information is stored or will be stored **within** that array
  - **HAS** to hold all the same type of information

Wrong	~~"Hello"~~	~~5~~	~~2~~	"World"

Right	5	2

The Array - Array Types

- An array's **type**
  - what type of information is stored or will be stored **within** that array
  - **HAS** to hold all the same type of information

Wrong	~~"Hello"~~	~~5~~	~~2~~	"World"

Right	5	2

Right	"Hello"	"World"

The Array - Array Types

- An array's **type**
  - what type of information is stored or will be stored **within** that array
  - **HAS** to hold all the same type of information

Wrong	"Hello"	5	2	"World"

Right	5	2

Right	"Hello"	"World"

The Array - Array Size

- An array's size is a set integer that is fixed upon creation of the array
  - Represents the total amount of elements that are able to be stored within the array
  - Cannot be changed

The Array - Array Size

- An array's size is a set integer that is fixed upon creation of the array
  - Represents the total amount of elements that are able to be stored within the array
  - Cannot be changed

The Array - Array Size

- An array's size is a set integer that is fixed upon creation of the array
  - Represents the total amount of elements that are able to be stored within the array
  - Cannot be changed

34	...
35	Define an array

The Array - Array Size

- An array's size is a set integer that is fixed upon creation of the array
  - Represents the total amount of elements that are able to be stored within the array
  - Cannot be changed

34	...
35	Define an array

THE ARRAY'S SIZE CANNOT BE CHANGED BY CONVENTIONAL METHODS PAST THIS POINT

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

Type

Size

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

Type

Size

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

Type

Size

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

Type

Size

The Array - Array Basics

- An array usually has 3 attributes
  - A Name
  - A Type
  - A Size

Name

Type

Size

The Array - Creating Arrays

- There are actually 2 different ways to create an array in most languages
  - Populate the array with elements right then and there
  - Set a specific size for the array, then populate it later

The Array - Creating Arrays

- There are actually 2 different ways to create an array in most languages
  - Populate the array with elements right then and there
  - Set a specific size for the array, then populate it later

Populate First

The Array - Creating Arrays

- There are actually 2 different ways to create an array in most languages
  - Populate the array with elements right then and there
  - Set a specific size for the array, then populate it later

Populate First

Populate Later

The Array - Creating Arrays

- There are actually 2 different ways to **create** an array in most languages
  - **Populate** the array with **elements** right then and there
  - Set a specific size for the array, then populate it later

Populate First

Populate Later

The Array - Creating Arrays

- There are actually 2 different ways to create an array in most languages
  - Populate the array with elements right then and there
  - Set a specific size for the array, then populate it later

Populate First

Populate Later

The Array - Creating Arrays

- There are actually 2 different ways to create an array in most languages
  - Populate the array with elements right then and there
  - Set a specific size for the array, then populate it later

Populate First

Populate Later

The Array - Populate-First Arrays

- Defining and filling an array as soon as you create it is used mainly for when you already know which values are going to be held within it

The Array - Populate-First Arrays

- **Defining** and **filling** an array as soon as you create it is used mainly for when you already **know** which values are going to be held within it

Salaries										

The Array - Populate-First Arrays

- **Defining** and **filling** an array as soon as you create it is used mainly for when you already **know** which values are going to be held within it

Salaries

The Array - Populate-First Arrays

- **Defining** and **filling** an array as soon as you create it is used mainly for when you already **know** which values are going to be held within it

Read from a text file

The Array - Populate-First Arrays

- **Defining** and **filling** an array as soon as you create it is used mainly for when you already **know** which values are going to be held within it

Salaries	10K	15K	13K	11K	25K	13K	13K	20K	15K	8K

Read from a text file

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

Size = 3

```
int array[] = { 1, 2, 3 };
```

```
array = [1, 2, 3]
```

```
int[] array = { 1, 2, 3 };
```

The Array - Populate-First Arrays

- The way you do this varies from language to language
  - Shown in Java, Python, and C#

Size = 4

```
int array[] = { 1, 2, 3, 4 };
```

```
array = [1, 2, 3, 4]
```

```
int[] array = { 1, 2, 3, 4 };
```

The Array - Populate-Later Arrays

- Creating an array by setting an initial size for our array, but not filling it with any elements
  - Slowly populate it as the programs run
  - Used for user-entered information

The Array - Populate-Later Arrays

- **Creating** an array by setting an initial size for our array, but not filling it with any elements
  - Slowly populate it as the programs run
  - Used for **user-entered** information

User Info										

The Array - Populate-Later Arrays

- Creating an array by setting an initial size for our array, but not filling it with any elements
  - Slowly populate it as the programs run
  - Used for user-entered information

User Info										

The information varies based on which user runs the code/what they input

The Array - Populate-Later Arrays

- The way you do this varies from language to language
  - Shown in Java and C#

```
int array[] = new int[10];
```

```
int[] array = new int[10];
```

The Array - Populate-Later Arrays

- The way you do this varies from language to language
  - Shown in Java and C#

```
int array[] = new int[10];
```

```
int[] array = new int[10];
```

The Array - Populate-Later Arrays

- The way you do this varies from language to language
  - Shown in Java and C#

```
int array[] = new int[10];
```

```
int[] array = new int[10];
```

The Array - Populate-Later Arrays

- The way you do this varies from language to language
  - Shown in Java and C#

```
int array[] = new int[10];
```

```
int[] array = new int[10];
```

The Array - Populate-Later Arrays

- The way you do this varies from language to language
  - Shown in Java and C#

```
int array[] = new int[10];
```

```
int[] array = new int[10];
```

The Array - Populate-Later Arrays

- The way you do this varies from language to language
  - Shown in Java and C#

```
int array[] = new int[10];
```

Size = FINAL

```
int[] array = new int[10];
```

The Array - Populate-Later Arrays

- The way you do this varies from language to language
  - Shown in Java and C#

```
int array[] = new int[20];
```

Size = FINAL

```
int[] array = new int[20];
```

The Array - Populate-Later Arrays

- The way you do this varies from language to language
  - Shown in Java and C#

```
int array[] = new int[20];
```

```
int[] array = new int[20];
```

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index										
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0									
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1								
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

- We reference it using both the arrays name and index number of the element you wish to retrieve

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

```
print(Numbers
```

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

```
print(Numbers[5])
```

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

```
print(Numbers[10])
```

The Array - Numerical Indexes

- To get information that is stored within the array, we use a numerical index
  - An integer which corresponds to an element within the array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

~~print(Numbers[10])~~

The Array - Replacing information in an Array

- Referencing an arrays index is also how we replace elements within an array

The Array - Replacing information in an Array

- Referencing an arrays **index** is also how we **replace** elements within an array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Replacing information in an Array

- **Referencing an arrays index is also how we replace elements within an array**

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Replacing information in an Array

- Referencing an arrays **index** is also how we **replace** elements within an array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

The Array - Replacing information in an Array

- Referencing an arrays **index** is also how we **replace** elements within an array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

Numbers[9]

The Array - Replacing information in an Array

- Referencing an arrays **index** is also how we **replace** elements within an array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

Numbers[9] = 11

The Array - Replacing information in an Array

- Referencing an arrays **index** is also how we **replace** elements within an array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

Numbers[9] = 11

The Array - Replacing information in an Array

- Referencing an arrays **index** is also how we **replace** elements within an array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	11

Numbers[9] = 11

The Array - Replacing information in an Array

- **Referencing an arrays index is also how we replace elements within an array**

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	11

The Array - 2-Dimensional Arrays

- An array with an array at each index is known as a 2-dimensional array

The Array - 2-Dimensional Arrays

- An array with an array at each index is known as a 2-dimensional array

Index	0	1	2	3	4
Array					

The Array - 2-Dimensional Arrays

- An array with an array at each index is known as a 2-dimensional array

Index	0	1	2	3	4
0					
1					
2					
3					
4					

The Array - 2-Dimensional Arrays

- An array with an array at each index is known as a 2-dimensional array

The Array - 2-Dimensional Arrays

- An array with an array at each index is known as a 2-dimensional array

The Array - 2-Dimensional Arrays

- An array with an array at each index is known as a 2-dimensional array

The Array - 2-Dimensional Arrays

- An array with an array at each index is known as a 2-dimensional array

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0				
1				
2				
3				

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

The Array - 2-Dimensional Arrays

- Referencing an element within a 2-dimensional array is mostly the same as with 1-dimensional arrays, except you now need 2 indexes
  - One for the column and row

Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

The Array - 2-Dimensional Arrays

The Array - 2-Dimensional Arrays

The Array - 2-Dimensional Arrays

The Array - Arrays as a Data Structure

The Array - Arrays as a Data Structure

Accessing

The Array - Arrays as a Data Structure

Accessing

Searching

The Array - Arrays as a Data Structure

Accessing

Searching

Inserting

The Array - Arrays as a Data Structure

Accessing

Searching

Inserting

Deleting

The Array - Arrays as a Data Structure

Accessing

Searching

Inserting

Deleting

The Array - Arrays as a Data Structure

Accessing

Searching

Inserting

Deleting

The Array - Arrays as a Data Structure

Accessing

O(1)

Searching

Inserting

Deleting

The Array - Arrays as a Data Structure

Memory

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String						
Stored Value	78	“Hey”						

The Array - Arrays as a Data Structure

Memory

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String						
Stored Value	78	“Hey”						

The Array - Arrays as a Data Structure

Memory

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String						
Stored Value	78	“Hey”						

The Array - Arrays as a Data Structure

Memory

```
int array[] = new int[3];
```

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String						
Stored Value	78	“Hey”						

The Array - Arrays as a Data Structure

Memory

```
int array[] = new int[3];
```

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String						
Stored Value	78	“Hey”						

The Array - Arrays as a Data Structure

Memory

int array[] = new int[3];

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String	Reserved For Array	Reserved For Array	Reserved For Array			
Stored Value	78	“Hey”	Reserved For Array	Reserved For Array	Reserved For Array			

The Array - Arrays as a Data Structure

Memory

```
int array[] = new int[3];
```

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String	int[]	int[]	int[]			
Stored Value	78	“Hey”	1	2	3			

The Array - Arrays as a Data Structure

Memory

array[1];

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String	int[]	int[]	int[]			
Stored Value	78	“Hey”	1	2	3			

The Array - Arrays as a Data Structure

Memory

array[1];

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String	int[]	int[]	int[]			
Stored Value	78	“Hey”	1	2	3			

The Array - Arrays as a Data Structure

Memory

array[1];

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String	int[]	int[]	int[]			
Stored Value	78	“Hey”	1	2	3			

The Array - Arrays as a Data Structure

Memory

array[1]; → **2**

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String	int[]	int[]	int[]			
Stored Value	78	“Hey”	1	2	3			

The Array - Arrays as a Data Structure

Memory

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String	int[]	int[]	int[]	int	int[]	int[]
Stored Value	78	“Hey”	1	2	3	567	2	3

The Array - Arrays as a Data Structure

Memory

Spot in Memory	0	1	2	3	4	5	6	7
Memory Type	int	String	int[]	int[]	int[]	int	int[]	int[]
Stored Value	78	“Hey”	1	2	3	567	2	3

The Array - Arrays as a Data Structure

Memory

Spot in Memory	0	1	2		4	5	6	
Memory Type	int	String	int[]	int[]	int[]		int[]	int[]
Stored Value	78	“Hey”	1		3	567	2	

The Array - Arrays as a Data Structure

Accessing

O(1)

Searching

Inserting

Deleting

The Array - Arrays as a Data Structure

Accessing

O(1)

Searching

O(n)

Inserting

Deleting

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	“Dave”	“Bob”	“Adam”	“Frank”	“Evan”	“Gary”

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	“Dave”	“Bob”	“Adam”	“Frank”	“Evan”	“Gary”

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

Search For: "Dave"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

Search For: "Dave"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

Search For: "Dave"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	“Dave”	“Bob”	“Adam”	“Frank”	“Evan”	“Gary”

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

- When using BigO notation, we always use worst-case scenario

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

- When using BigO notation, we always use worst-case scenario

Search For: "Gary"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

- When using BigO notation, we always use worst-case scenario

Search For: "Gary"

The Array - Arrays as a Data Structure

- Searching through arrays is O(n) because most of the time we are working with unsorted lists
  - must use linear search

Names	"Dave"	"Bob"	"Adam"	"Frank"	"Evan"	"Gary"

- When using BigO notation, we always use worst-case scenario

Search For: "Gary"

The Array - Arrays as a Data Structure

Accessing

O(1)

Searching

O(n)

Inserting

Deleting

The Array - Arrays as a Data Structure

Accessing → O(1)

Searching → O(n)

Inserting → O(n)

Deleting → O(n)

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Index	0	1	2	3	4
Numbers	1	2	3	4	

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Insert 1 at Index 0

Index	0	1	2	3	4
Numbers	2	3	4	5	

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Insert 1 at Index 0

Index	0	1	2	3	4
Numbers		2	3	4	5

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Insert 1 at Index 0

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Index	0	1	2	3	4
Numbers	1	2	3	4	

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Insert 5 at Index 4

Index	0	1	2	3	4
Numbers	1	2	3	4	

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Insert 5 at Index 4

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Insert 5 at Index 4 → O(1)

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

Insert 5 at Index 4 → **O(1)**

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Inserting is O(n) because inserting an element within the array requires you to shift every element that's after the index you want to insert the value at to the right one space

O(n) ← **Insert 5 at Index 4** → O(1)

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[0] = Numbers[1]

Index	0	1	2	3	4
Numbers	1	2	3	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[0] = Numbers[1]

Index	0	1	2	3	4
Numbers	2	2	3	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[1] = Numbers[2]

Index	0	1	2	3	4
Numbers	2	2	3	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[1] = Numbers[2]

Index	0	1	2	3	4
Numbers	2	3	3	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[2] = Numbers[3]

Index	0	1	2	3	4
Numbers	2	3	3	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[2] = Numbers[3]

Index	0	1	2	3	4
Numbers	2	3	4	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[3] = Numbers[4]

Index	0	1	2	3	4
Numbers	2	3	4	4	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[3] = Numbers[4]

Index	0	1	2	3	4
Numbers	2	3	4	5	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[4] = null;

Index	0	1	2	3	4
Numbers	2	3	4	5	5

The Array - Arrays as a Data Structure

- Deleting is O(n) because deleting an element within the array requires you to shift every element to the right of the one you want to delete down one index

Numbers[4] = null;

Index	0	1	2	3	4
Numbers	2	3	4	5	

The Array - Arrays as a Data Structure

Accessing → O(1)

Searching → O(n)

Inserting → O(n)

Deleting → O(n)

The Array - Arrays as a Data Structure

Accessing

O(1)

Searching

O(n)

Inserting

O(n)

Deleting

O(n)

The Array - Arrays as a Data Structure

Accessing

O(1)

Searching

O(n)

Inserting

O(n)

Deleting

O(n)

The Array - Arrays as a Data Structure

Accessing

O(1)

Searching

O(n)

Inserting

O(n)

Deleting

O(n)

The Array - Arrays as a Data Structure

Accessing

O(1)

Searching

O(n)

Inserting

O(n)

Deleting

O(n)

The Array - Arrays as a Data Structure

Accessing → O(1)

Searching → O(n)

Inserting → O(n)

Deleting → O(n)

The Array - Pros and Cons

Pros

The Array - Pros and Cons

Pros

Good for storing similar contiguous data

The Array - Pros and Cons

Pros

Good for storing similar contiguous data

O(1) Accessing Power

The Array - Pros and Cons

Pros

Good for storing similar contiguous data

O(1) Accessing Power

Very basic. Easy to learn and master

The Array - Pros and Cons

Pros	Cons
Good for storing similar contiguous data	
O(1) Accessing Power	
Very basic. Easy to learn and master	

The Array - Pros and Cons

Pros	Cons
Good for storing similar contiguous data	Size of the array cannot be changed once initialized
O(1) Accessing Power	
Very basic. Easy to learn and master	

The Array - Pros and Cons

Pros	Cons
Good for storing similar contiguous data	Size of the array cannot be changed once initialized
O(1) Accessing Power	Inserting and Deleting are not efficient
Very basic. Easy to learn and master	

The Array - Pros and Cons

Pros	Cons
Good for storing similar contiguous data	Size of the array cannot be changed once initialized
O(1) Accessing Power	Inserting and Deleting are not efficient
Very basic. Easy to learn and master	Can be wasting storage space

The Array - Pros and Cons

Pros	Cons
Good for storing similar contiguous data	Size of the array cannot be changed once initialized
O(1) Accessing Power	Inserting and Deleting are not efficient
Very basic. Easy to learn and master	Can be wasting storage space

Overall, pretty reliable. Has some flaws as well as advantages. Can be used in almost any program if need be, but sometimes you may want extra functionality

An Introduction to Data Structures

The Arraylist

The ArrayList - Introduction

- The arrayList, fundamentally, can be thought of as a growing array

The ArrayList - Introduction

- The **arrayList**, fundamentally, can be thought of as a **growing array**

Nums	4	8	24	6

The ArrayList - Introduction

- The **arrayList**, fundamentally, can be thought of as a **growing array**

Array Size is Final

Nums	4	8	24	6

The ArrayList - Introduction

- The **arrayList**, fundamentally, can be thought of as a **growing array**

ArrayList Size is Dynamic

Nums	4	8	24	6

The ArrayList - Introduction

- The **arrayList**, fundamentally, can be thought of as a **growing array**

ArrayList Size is Dynamic

Nums	4	8	24	6	9

The ArrayList - Introduction

- The ar[illegible]tally, can be thought of as a growing array

Why not ALWAY use arrayLists?

Nums	4	8	24		9

The ArrayList - Structure of the ArrayList

- An arrayList is backed by an array
  - This makes the arrayList have a lot of similar functionality to an array

The ArrayList - Structure of the ArrayList

- An arrayList is backed by an array
  - This makes the arrayList have a lot of similar functionality to an array

ArrayList

The ArrayList - Structure of the ArrayList

- An arrayList is backed by an array
  - This makes the arrayList have a lot of similar functionality to an array

The ArrayList - Structure of the ArrayList

- An arrayList is backed by an array
  - This makes the arrayList have a lot of similar functionality to an array

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Arrays and arrayLists are frankensteined together into a single data structure called Lists
  - Take some functionality from both data structures

The ArrayList - Initializing an ArrayList

- Arrays and arrayLists are frankensteined together into a single data structure called Lists
  - Take some functionality from both data structures

Python Lists

The ArrayList - Initializing an ArrayList

- Arrays and arrayLists are frankensteined together into a single data structure called Lists
  - Take some functionality from both data structures

Python Lists

The ArrayList - Initializing an ArrayList

- Arrays and arrayLists are frankensteined together into a single data structure called Lists
  - Take some functionality from both data structures

Python Lists

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

Class Hierarchy

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

Class Hierarchy

Object-Oriented Programming

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

Size = 50

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>(50);
```

```
ArrayList arrayList = new ArrayList(50);
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

Size = 10

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

Size = 3

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>(1, 2, 3);
```

```
ArrayList arrayList = new ArrayList(1, 2, 3);
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

Size = 3

```
ArrayList<Integer> arr                    ayList<Integer>(1, 2, 3);
```

```
Array    arrayList = new Array                , 2, 3);
```

The ArrayList - Initializing an ArrayList

- Initializing an arrayList varies based on the language
  - Shown below in Java and C#

Size = 10

```
ArrayList<Integer> arrayList  = new ArrayList<Integer>();
```

```
ArrayList arrayList = new ArrayList();
```

The ArrayList - ArrayList Functionality

- An arrayList can be thought of as an evolved array
  - Beefier
  - More Functionality
  - More Powerful

The ArrayList - ArrayList Functionality

- An arrayList can be thought of as an evolved array
  - Beefier
  - More Functionality
  - More Powerful

"Who are you?"

"I'm you, but Stronger"

The ArrayList - ArrayList Functionality

- This is mainly because it belongs to the pre-built arrayList "class"
  - Means it has pre-built functions that are at our disposal

The ArrayList - ArrayList Functionality

- This is mainly because it belongs to the pre-built arrayList "class"
  - Means it has pre-built functions that are at our disposal

The ArrayList - ArrayList Functionality

- This is mainly because it belongs to the pre-built **arrayList "class"**
  - Means it has **pre-built functions** that are at our disposal

The ArrayList - ArrayList Functionality

- This is mainly because it belongs to the pre-built arrayList "class"
  - Means it has pre-built functions that are at our disposal

The ArrayList - ArrayList Functionality

- This is mainly because it belongs to the pre-built **arrayList "class"**
  - Means it has **pre-built functions** that are at our disposal

The ArrayList - ArrayList Functionality

- This is mainly because it belongs to the pre-built **arrayList "class"**
  - Means it has **pre-built functions** that are at our disposal

The ArrayList - ArrayList Functionality

- This is mainly because it belongs to the pre-built **arrayList "class"**
  - Means it has **pre-built functions** that are at our disposal

The ArrayList - ArrayList Functionality

- The type of functionality you're going to get is going to vary based on language

Java

The ArrayList - ArrayList Functionality

- The type of functionality you're going to get is going to vary based on language

Java

boolean add(E e)

This method appends the specified element to the end of this list.

void add(int index, E element)

This method inserts the specified element at the specified position in this list.

The ArrayList - ArrayList Functionality

- The type of **functionality** you're going to get is going to vary based on **language**

Java

E remove(int index)

This method removes the element at the specified position in this list.

boolean remove(Object o)

This method removes the first occurrence of the specified element from this list, if it is present.

boolean add(E e)

This method appends the specified element to the end of this list.

void add(int index, E element)

This method inserts the specified element at the specified position in this list.

The ArrayList - ArrayList Functionality

- The type of **functionality** you're going to get is going to vary based on **language**

Java

E remove(int index)

This method removes the element at the specified position in this list.

boolean remove(Object o)

This method removes the first occurrence of the specified element from this list, if it is present.

boolean add(E e)

This method appends the specified element to the end of this list.

void add(int index, E element)

This method inserts the specified element at the specified position in this list.

void clear()

This method removes all of the elements from this list.

The ArrayList - ArrayList Functionality

- The type of **functionality** you're going to get is going to vary based on **language**

Java

E remove(int index)

This method removes the element at the specified position in this list.

boolean remove(Object o)

This method removes the first occurrence of the specified element from this list, if it is present.

boolean add(E e)

This method appends the specified element to the end of this list.

void add(int index, E element)

This method inserts the specified element at the specified position in this list.

void clear()

This method removes all of the elements from this list.

int size()

This method returns the number of elements in this list.

The ArrayList - ArrayList Functionality

- The type of functionality you're going to get is going to vary based on language

The ArrayList - ArrayList Functionality

- The type of functionality you're going to get is going to vary based on language

C#

The ArrayList - ArrayList Functionality

- The type of functionality you're going to get is going to vary based on language

C#

Add(Object)	Adds an object to the end of the ArrayList.

The ArrayList - ArrayList Functionality

- The type of functionality you're going to get is going to vary based on language

C#

Add(Object)	Adds an object to the end of the ArrayList.
Clear()	Removes all elements from the ArrayList.

The ArrayList - ArrayList Functionality

- The type of functionality you're going to get is going to vary based on language

C#

Add(Object)	Adds an object to the end of the ArrayList.
Clear()	Removes all elements from the ArrayList.
BinarySearch(Object, IComparer)	Searches the entire sorted ArrayList for an element using the specified comparer and returns the zero-based index of the element.

The ArrayList - ArrayList Methods

- Because of the variability surrounding the arrayList, we will only be covering 6 common methods

The ArrayList - ArrayList Methods

- Because of the variability surrounding the arrayList, we will only be covering 6 common methods

Add Method

The ArrayList - ArrayList Methods

- Because of the variability surrounding the arrayList, we will only be covering 6 common methods

Add Method

Remove Method

The ArrayList - ArrayList Methods

- Because of the variability surrounding the arrayList, we will only be covering 6 common methods

Add Method

Remove Method

Get Method

The ArrayList - ArrayList Methods

- Because of the variability surrounding the arrayList, we will only be covering 6 common methods

Add Method

Set Method

Remove Method

Get Method

The ArrayList - ArrayList Methods

- Because of the variability surrounding the arrayList, we will only be covering 6 common methods

Add Method

Set Method

Remove Method

Clear Method

Get Method

The ArrayList - ArrayList Methods

- Because of the variability surrounding the arrayList, we will only be covering 6 common methods

Add Method

Set Method

Remove Method

Clear Method

Get Method

toArray Method

The ArrayList - ArrayList Methods

- Because of the **variability** surrounding the arrayList, we will only be covering **6** common methods

The ArrayList - ArrayList Methods

- Because of the variability surrounding the arrayList, we will only be covering 6 common methods

Add Method

Set Method

Remove Method

Clear Method

Get Method

toArray Method

The ArrayList - ArrayList Methods

ArrayList

The ArrayList - ArrayList Methods

ArrayList exampleAList

The ArrayList - ArrayList Methods

```
ArrayList exampleAList = new ArrayList();
```

The ArrayList - ArrayList Methods

```
ArrayList exampleAList = new ArrayList(4);
```

The ArrayList - ArrayList Methods

```
ArrayList exampleAList = new ArrayList(4);
```

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- **The add method has 2 different types...**

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

Add(Object, Index)

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

Add(Object, Index)

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

Add(Object, Index)

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(2);
```

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(2);
```

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(2);
```

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(2);
```

2

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(2);
```

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

Autoboxing

```
exampleAList.add(2);
```

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(2);
```

Index	0	1	2	3
ExampleAList				

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(2);
```

Index	0	1	2	3
ExampleAList	2			

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(5);
```

Index	0	1	2	3
ExampleAList	2			

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(5);
```

Index	0	1	2	3
ExampleAList	2			

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(5);
```

Index	0	1	2	3
ExampleAList	2			

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(2);
```

Index	0	1	2	3
ExampleAList	2	5		

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object)

- Appends the element you pass in as an argument to the end of the arrayList

```
exampleAList.add(2);
```

Index	0	1	2	3
ExampleAList	2	5		

The ArrayList - Add Method

- **The add method has 2 different types...**

Index	0	1	2	3
ExampleAList	2	5		

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

Index	0	1	2	3
ExampleAList	2	5		

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

- Appends the element you pass in as an argument at the index you pass in

Index	0	1	2	3
ExampleAList	2	5		

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

```
exampleAList.add(1,0);
```

- Appends the element you pass in as an argument at the index you pass in

Index	0	1	2	3
ExampleAList	2	5		

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

```
exampleAList.add(1,0);
```

- Appends the element you pass in as an argument at the index you pass in

Index	0	1	2	3
ExampleAList		2	5	

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

```
exampleAList.add(1,0);
```

- Appends the element you pass in as an argument at the index you pass in

Index	0	1	2	3
ExampleAList	1	2	5	

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

```
exampleAList.add(3,2);
```

- Appends the element you pass in as an argument at the index you pass in

Index	0	1	2	3
ExampleAList	1	2	5	

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

```
exampleAList.add(3,2);
```

- Appends the element you pass in as an argument at the index you pass in

Index	0	1	2	3
ExampleAList	1	2		5

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

```
exampleAList.add(3,2);
```

- Appends the element you pass in as an argument at the index you pass in

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Add Method

- The add method has 2 different types...

Integer

```
exampleAList.add(3,2);
```

Add(Object, Index)

- Appends the element you pass in as an argument at the index you pass in

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Add Method

- The add method has 2 different types...

Integer

```
exampleAList.add(3,2);
```

Index

Add(Object, Index)

- Appends the element you pass in as an argument at the index you pass in

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- **The remove method also has 2 different types...**

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

Remove(Object)

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

Remove(Object)

- Removes the first instance of the object passed into the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

Remove(Object)

- Removes the first instance of the object passed into the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

Remove(Object)

- Removes the first instance of the object passed into the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

```
exampleAList.remove(3);
```

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

```
exampleAList.remove(3);
```

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

```
exampleAList.remove(3);
```

Index	0	1	2	3
ExampleAList	1	2	3	

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Index)

- Removes the Object at the index you provide

```
exampleAList.remove(3);
```

Index	0	1	2	3
ExampleAList	1	2	3	

The ArrayList - Remove Method

- **The remove method also has 2 different types...**

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Object)

- Removes the first instance of the object passed into the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Object)

```
exampleAList.remove(new Integer(5));
```

- Removes the first instance of the object passed into the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

```
exampleAList.remove(new Integer(5));
```

Remove(Object)

- Removes the first instance of the object passed into the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Object)

```
exampleAList.remove(new Integer(5));
```

- Removes the first instance of the object passed into the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Object)

- Removes the first instance of the object passed into the arrayList

```
exampleAList.remove(new Integer(5));
```

True

Index	0	1	2	3
ExampleAList	1	2	3	

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Object)

```
exampleAList.remove(new Integer(5));
```

- Removes the first instance of the object passed into the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	

The ArrayList - Remove Method

- The remove method also has 2 different types...

Remove(Object)

```
exampleAList.remove(new Integer(5));
```

- Removes the first instance of the object passed into the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Get Method

- The get method is the same as referencing an index for an array

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Get Method

- The get method is the same as referencing an index for an array

Get(Index)

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Get Method

- The get method is the same as referencing an index for an array

Get(Index)

- Returns the object contained at the given index

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Get Method

- The get method is the same as referencing an index for an array

Get(Index)

- Returns the object contained at the given index

```
exampleAList.get(0);
```

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Get Method

- The get method is the same as referencing an index for an array

Get(Index)

- Returns the object contained at the given index

```
exampleAList.get(0);
```

↓

1

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Get Method

- The get method is the same as referencing an index for an array

Get(Index)

- Returns the object contained at the given index

```
exampleAList.get(2);
```

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Get Method

- The get method is the same as referencing an index for an array

Get(Index)

- Returns the object contained at the given index

```
exampleAList.get(2);
```

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Get Method

- The get method is the same as referencing an index for an array

Get(Index)

- Returns the object contained at the given index

```
exampleAList.get(2);
```

↓

3

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

exampleAList.set

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

exampleAList.set(3,

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

```
exampleAList.set(3, 4);
```

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

```
exampleAList.set(3, 4);
```

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

```
exampleAList.set(3, 4);
```

Index	0	1	2	3
ExampleAList	1	2	3	5

The ArrayList - Set Method

- The set method is how we **replace** elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

```
exampleAList.set(3, 4);
```

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - Set Method

- The set method is how we replace elements within the arrayList

Set(Index, Object)

- Sets the element at the index which you passed in, to the object you also passed in

```
exampleAList.set(3, 4);
```

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - Clear Method

- For if you just HATE your arrayList

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - Clear Method

- For if you just HATE your arrayList

Clear()

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - Clear Method

- For if you just **HATE** your arrayList

Clear()

- Clears the arrayList, deleting every element entirely

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - Clear Method

- For if you just **HATE** your arrayList

Clear()

- Clears the arrayList, deleting every element entirely

```
exampleAList.clear();
```

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - Clear Method

- For if you just **HATE** your arrayList

Clear()

- Clears the arrayList, deleting every element entirely

```
exampleAList.clear();
```

Index	0	1	2	3
ExampleAList				

The ArrayList - Clear Method

- For if you just **HATE** your arrayList

Clear()

- Clears the arrayList, deleting every element entirely

```
exampleAList.clear();
```

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - toArray Method

- Used to convert an **arrayList** to an **Array**

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - toArray Method

- Used to convert an **arrayList** to an **Array**

toArray()

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - toArray Method

- Used to convert an **arrayList** to an **Array**

toArray()

- Converts the arrayList into an array. Must be set equal to the creation of a new array

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - toArray Method

- Used to convert an **arrayList** to an **Array**

toArray()

- Converts the arrayList into an array. Must be set equal to the creation of a new array

```
Object convertedExample[] = exampleAList.toArray();
```

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - toArray Method

- Used to convert an **arrayList** to an **Array**

toArray()

- Converts the arrayList into an array. Must be set equal to the creation of a new array

```
Object convertedExample[] =
exampleAList.toArray();
```

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - toArray Method

- Used to convert an **arrayList** to an **Array**

toArray()

- Converts the arrayList into an array. Must be set equal to the creation of a new array

```
Object convertedExample[] = exampleAList.toArray();
```

Index	0	1	2	3
ExampleAList	1	2	3	4

The ArrayList - ArrayList Methods

Add Method

Set Method

Remove Method

Clear Method

Get Method

toArray Method

- You should be able to find these, or versions of these, in any language which also supports arrayLists

The ArrayList - ArrayList as a Data Structure

The ArrayList - ArrayList as a Data Structure

Accessing

The ArrayList - ArrayList as a Data Structure

Accessing

Searching

The ArrayList - ArrayList as a Data Structure

Accessing

Searching

Inserting

The ArrayList - ArrayList as a Data Structure

Accessing

Searching

Inserting

Deleting

The ArrayList - ArrayList as a Data Structure

Accessing

O(1)

Searching

Inserting

Deleting

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

Inserting

Deleting

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

Inserting

Deleting

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	“hey”	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	“hey”	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Memory

Index	0	1	2	3
Example	87	91	100	42

Spot in Memory	42	...	87	88	...	91	...	99	100
Stored Value	4	...	1	"hey"	...	2	...	true	2

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

Inserting

Deleting

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

O(n)

Inserting

Deleting

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

O(n)

Inserting

Add(Object, Index)

Add(Object)

O(n)

Deleting

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

O(n)

Inserting

Add(Object, Index)

Add(Object)

O(n)

Deleting

Remove (Object)

Remove (Index)

O(n)

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

O(n)

Inserting

Add(Object, Index)

Add(Object)

O(n)

Deleting

Remove (Object)

Remove (Index)

O(n)

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

O(n)

Inserting

Add(Object, Index)

Add(Object)

O(n)

Deleting

Remove (Object)

Remove (Index)

O(n)

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

O(n)

Inserting

Add(Object, Index)

Add(Object)

O(n)

Deleting

Remove (Object)

Remove (Index)

O(n)

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

O(n)

Inserting

Add(Object, Index)

Add(Object)

O(n)

Deleting

Remove (Object)

Remove (Index)

O(n)

The ArrayList - ArrayList as a Data Structure

Accessing

Get Method

O(1)

Searching

O(n)

Inserting

Add(Object, Index)

Add(Object)

O(n)

Deleting

Remove (Object)

Remove (Index)

O(n)

The ArrayList - ArrayList as a Data Structure

Why not ALWAY use arrayLists?

Accessing

Get Method

O(1)

Searching

O(n)

Adding

Index)

O(n)

Deleting

Remove (Index)

O(n)

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists
Fixed Size	

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists
Fixed Size	Dynamic Size

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists
Fixed Size	Dynamic Size
Can store all data types	

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists
Fixed Size	Dynamic Size
Can store all data types	Can only store Objects

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists
Fixed Size	Dynamic Size
Can store all data types	Can only store Objects
Methods need to be created	

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists
Fixed Size	Dynamic Size
Can store all data types	Can only store Objects
Methods need to be created	Methods are created for you

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists
Fixed Size	Dynamic Size
Can store all data types	Can only store Objects
Methods need to be created	Methods are created for you
Doesn't require much memory use or upkeep	

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists
Fixed Size	Dynamic Size
Can store all data types	Can only store Objects
Methods need to be created	Methods are created for you
Doesn't require much memory use or upkeep	Requires more memory use and upkeep

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists

The ArrayList - Comparing and Contrasting with Arrays

Arrays

Smaller tasks, where you won't be interacting or changing the data that often

ArrayLists

The ArrayList - Comparing and Contrasting with Arrays

Arrays	ArrayLists
Smaller tasks, where you won't be interacting or changing the data that often	More interactive programs where you'll be modifying data quite a bit

The ArrayList - Conclusion

ArrayLists

The ArrayList - Conclusion

ArrayLists

A dynamically increasing array

The ArrayList - Conclusion

ArrayLists

A dynamically increasing array

- Add Method
- Set Method
- Remove Method
- Clear Method
- Get Method
- toArray Method

An Introduction to Data Structures

The Stack

The Stack - The Different types of Data Structures

This Video

The Stack - The Different types of Data Structures

This Video

Random Access Data Structures

The Stack - The Different types of Data Structures

The Stack - The Different types of Data Structures

The Stack - The Different types of Data Structures

The Stack - The Different types of Data Structures

Array	1	8	2	6	9

The Stack - The Different types of Data Structures

Array	1	8	2	6	9

ArrayList	3	5	4	8	7

The Stack - The Different types of Data Structures

Array	1	8	2	6	9

Index	0	1	2	3	4

ArrayList	3	5	4	8	7

The Stack - The Different types of Data Structures

Array	1	8	2	6	9
Index	0	1	2	3	4
ArrayList	3	5	4	8	7

The Stack - The Different types of Data Structures

Array	1	8	2	6	9
Index	0	1	2	3	4
ArrayList	3	5	4	8	7

The Stack - The Different types of Data Structures

Array	1	8	2	6	9
Index	0	1	2	3	4
ArrayList	3	5	4	8	7

The Stack - The Different types of Data Structures

Array	1	8	2	6	9
Index	0	1	2	3	4
ArrayList	3	5	4	8	7

The Stack - The Different types of Data Structures

Array	1	8	2	6	9

Index	0	1	2	3	4

ArrayList	3	5	4	8	7

The Stack - Random Access Data Structures

The Stack - Random Access Data Structures

Page 45

The Stack - Random Access Data Structures

Page 72

Page 45

The Stack - Random Access Data Structures

Page 34

Page 72

Page 45

The Stack - Random Access Data Structures

Independent

Page 34

Page 72

Page 45

The Stack - Random Access Data Structures

Independent

Page 34

Page 72

Page 45

Array	1	8	2	6	9

The Stack - Random Access Data Structures

Independent

Page 34

Page 72

Page 45

Array	1	8	2	6	9

Array[2]

The Stack - Random Access Data Structures

Independent

Page 34

Page 72

Page 45

Array	1	8	2	6	9

Array[2]

Array[0]

The Stack - Random Access Data Structures

Independent

Page 34

Page 72

Page 45

Array	1	8	2	6	9

Array[2]

Array[0]

Array[4]

The Stack - Random Access Data Structures

Independent

Page 34

Page 72

Page 45

Independent

Array	1	8	2	6	9

Array[2]

Array[0]

Array[4]

The Stack - Sequential Access Data Structures

- **Sequential Access Data Structures** can only be accessed in a **particular order**
  - Each element is **dependent** on the others
  - May only be obtainable **through** those other elements

The Stack - Sequential Access Data Structures

- **Sequential Access Data Structures** can only be accessed in a **particular order**
  - Each element is **dependent** on the others
  - May only be obtainable **through** those other elements

The Stack - Sequential Access Data Structures

- **Sequential Access Data Structures** can only be accessed in a **particular order**
  - Each element is **dependent** on the others
  - May only be obtainable **through** those other elements

72 Inches

The Stack - Sequential Access Data Structures

- **Sequential Access Data Structures** can only be accessed in a **particular order**
  - Each element is **dependent** on the others
  - May only be obtainable **through** those other elements

1-71 Inches

↓

72 Inches

The Stack - Sequential Access Data Structures

- **Sequential Access Data Structures** can only be accessed in a **particular order**
  - Each element is **dependent** on the others
  - May only be obtainable **through** those other elements

1-71 Inches

↓

72 Inches

The Stack - The Types of Data Structures

Random Access Data Structures

The Stack - The Types of Data Structures

Random Access Data Structures

Provides O(1) Accessing

The Stack - The Types of Data Structures

Random Access Data Structures

Provides O(1) Accessing

Independent Elements

The Stack - The Types of Data Structures

Random Access Data Structures

Provides O(1) Accessing

Independent Elements

Arrays and ArrayLists

The Stack - The Types of Data Structures

Random Access Data Structures

- Provides O(1) Accessing
- Independent Elements
- Arrays and ArrayLists

Sequential Access Data Structures

The Stack - The Types of Data Structures

Random Access Data Structures	Sequential Access Data Structures
Provides O(1) Accessing	Do not provide O(1) Accessing
Independent Elements	
Arrays and ArrayLists	

The Stack - The Types of Data Structures

Random Access Data Structures	Sequential Access Data Structures
Provides O(1) Accessing	Do not provide O(1) Accessing
Independent Elements	Dependent Elements
Arrays and ArrayLists	

The Stack - The Types of Data Structures

Random Access Data Structures	Sequential Access Data Structures
Provides O(1) Accessing	Do not provide O(1) Accessing
Independent Elements	Dependent Elements
Arrays and ArrayLists	Stacks, Queues, Linked Lists

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

Stack

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

Stack

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

Weakness?

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

Weakness?

The Stack - Stack Basics

- **The Stack**
  - A **sequential access data structure** in which we add elements and remove elements according to the **LIFO Principle**
    - Last In First Out

Weakness?

The Stack - Common Stack Methods

The Stack - Common Stack Methods

Push Method

The Stack - Common Stack Methods

Push Method

Pop Method

The Stack - Common Stack Methods

Push Method

Pop Method

Peek Method

The Stack - Common Stack Methods

Push Method

Pop Method

Peek Method

Contains Method

The Stack - Common Stack Methods

The Stack - Common Stack Methods

exampleStack

The Stack - Push Method

- Pushes an object or element onto the top of the Stack

Size = 0

exampleStack

The Stack - Push Method

- Pushes an object or element onto the top of the Stack

Push (Object)

Size = 0

exampleStack

The Stack - Push Method

- Pushes an object or element onto the top of the Stack

Push (Object)

```
exampleStack.push("now")
```

Size = 0

exampleStack

The Stack - Push Method

- Pushes an object or element onto the top of the Stack

Push (Object)

```
exampleStack.push("now")
```

Size = 1

now

exampleStack

The Stack - Push Method

- Pushes an object or element onto the top of the Stack

Push (Object)

```
exampleStack.push("channel")
```

Size = 1

now

exampleStack

The Stack - Push Method

- Pushes an object or element onto the top of the Stack

Push (Object)

```
exampleStack.push("channel")
```

Size = 2

The Stack - Push Method

- Pushes an object or element onto the top of the Stack

Push (Object)

exampleStack.push("this")

Size = 2

channel

now

exampleStack

The Stack - Push Method

- **Pushes** an object or element onto the **top** of the Stack

Push (Object)

```
exampleStack.push("this")
```

Size = 3

The Stack - Push Method

- **Pushes** an object or element onto the **top** of the Stack

Push (Object)

```
exampleStack.push("to")
```

Size = 3

The Stack - Push Method

- **Pushes** an object or element onto the **top** of the Stack

Push (Object)

```
exampleStack.push("to")
```

Size = 4

The Stack - Push Method

- **Pushes** an object or element onto the **top** of the Stack

Push (Object)

```
exampleStack.push("Subscribe")
```

Size = 4

The Stack - Push Method

- **Pushes** an object or element onto the **top** of the Stack

Push (Object)

```
exampleStack.push("Subscribe")
```

Size = 5

The Stack - Push Method

- Pushes an object or element onto the top of the Stack

Push (Object)

exampleStack.push("Subscribe")

Size = 5

The Stack - Push Method

- **Pushes** an object or element onto the **top** of the Stack

Push (Object)

```
exampleStack.push("Subscribe")
```

Size = 5

The Stack - Pop Method

- Used to remove an element from the top of the Stack

The Stack - Pop Method

- Used to remove an element from the top of the Stack

Pop ()

Subscribe

to

this

channel

now

exampleStack

The Stack - Pop Method

- Used to remove an element from the top of the Stack

Pop ()

```
exampleStack.pop()
```

Subscribe
to
this
channel
now
exampleStack

The Stack - Pop Method

- Used to remove an element from the top of the Stack

Pop ()

```
exampleStack.pop()
```

The Stack - Pop Method

- Used to **remove** an element from the top of the Stack

Pop ()

```
exampleStack.pop()
```

Returned

↓

Subscribe

The Stack - Pop Method

- Used to **remove** an element from the top of the Stack

Pop ()

```
exampleStack.pop()
```

Returned

↓

Subscribe

this

channel

now

exampleStack

The Stack - Pop Method

- Used to **remove** an element from the top of the Stack

Pop ()

exampleStack.pop()

Returned

↓

to

this

channel

now

exampleStack

The Stack - Pop Method

- Used to **remove** an element from the top of the Stack

Pop ()

```
exampleStack.pop()
```

Returned

↓

to

channel

now

exampleStack

The Stack - Pop Method

- Used to **remove** an element from the top of the Stack

Pop ()

```
exampleStack.pop()
```

Returned

this

channel

now

exampleStack

The Stack - Pop Method

- Used to **remove** an element from the top of the Stack

Pop ()

```
exampleStack.pop()
```

Returned

this

now

exampleStack

The Stack - Pop Method

- Used to remove an element from the top of the Stack

Pop ()

```
exampleStack.pop()
```

Returned

↓

channel

now

exampleStack

The Stack - Pop Method

- Used to remove an element from the top of the Stack

Pop ()

```
exampleStack.pop()
```

Returned

↓

channel

exampleStack

The Stack - Pop Method

- Used to **remove** an element from the top of the Stack

Pop ()

exampleStack.pop()

Returned

↓

now

exampleStack

The Stack - Pop Method

- Used to remove an element from the top of the Stack

exampleStack

The Stack - Pop Method

- Used to remove an element from the top of the Stack

The Stack - Common Stack Methods

Push Method

Pop Method

Peek Method

Contains Method

The Stack - Common Stack Methods

Push Method

Pop Method

Peek Method

Contains Method

The Stack - Common Stack Methods

Push Method

Pop Method

Peek Method

Contains Method

The Stack - Common Stack Methods

Push Method

Pop Method

Peek Method

Contains Method

The Stack - Common Stack Methods

Push Method

Pop Method

Peek Method

Contains Method

The Stack - Peek Method

- Allows you to get the value at the top of the list without removing it

The Stack - Peek Method

- Allows you to get the value at the top of the list without removing it

Peek ()

The Stack - Peek Method

- Allows you to get the value at the top of the list without removing it

Peek ()

```
exampleStack.peek()
```

The Stack - Peek Method

- Allows you to get the value at the top of the list without removing it

Peek ()

exampleStack.peek()

Returned

Subscribe

Subscribe

to

this

channel

now

exampleStack

The Stack - Peek Method

- Allows you to get the value at the top of the list without removing it

Peek ()

exampleStack.peek()

Returned

Subscribe

The Stack - Peek Method

- Allows you to get the value at the top of the list without removing it

Peek ()

```
exampleStack.peek()
```

Returned

↓

to

The Stack - Peek Method

- Allows you to get the value at the top of the list without removing it

Peek ()

```
exampleStack.peek()
```

Returned

↓

to

The Stack - Contains Method

- Used for searching through the Stack

The Stack - Contains Method

- Used for searching through the Stack

Contains (Object)

The Stack - Contains Method

- Used for searching through the Stack

Contains (Object)

```
exampleStack.contains("Subscribe")
```

The Stack - Contains Method

- Used for searching through the Stack

Contains (Object)

```
exampleStack.contains("Subscribe")
```

Returned

↓

True

Subscribe

to

this

channel

now

exampleStack

The Stack - Contains Method

- Used for searching through the Stack

Contains (Object)

```
exampleStack.contains("this")
```

The Stack - Contains Method

- Used for searching through the Stack

Contains (Object)

```
exampleStack.contains("this")
```

Returned

↓

True

Subscribe

to

this

channel

now

exampleStack

The Stack - Contains Method

- Used for searching through the Stack

Contains (Object)

```
exampleStack.contains("hello")
```

The Stack - Contains Method

- Used for searching through the Stack

Contains (Object)

exampleStack.contains("hello")

Returned

↓

False

Subscribe

to

this

channel

now

exampleStack

The Stack - Common Stack Methods

Push Method

Pop Method

Peek Method

Contains Method

The Stack - Time Complexity Equations

The Stack - Time Complexity Equations

Accessing

The Stack - Time Complexity Equations

Accessing

O(n)

The Stack - Time Complexity Equations

Accessing

O(n)

The Stack - Time Complexity Equations

Accessing

O(n)

The Stack - Time Complexity Equations

Accessing

O(n)

The Stack - Time Complexity Equations

Accessing

O(n)

The Stack - Time Complexity Equations

Accessing

O(n)

The Stack - Time Complexity Equations

Accessing

O(n)

The Stack - Time Complexity Equations

Accessing

O(n)

The Stack - Time Complexity Equations

Accessing

Searching

O(n)

The Stack - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The Stack - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

The Stack - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Deleting

The Stack - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

O(1)

Deleting

O(1)

The Stack - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Push (Object)

O(1)

Deleting

O(1)

The Stack - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Push (Object)

O(1)

Deleting

Pop ()

O(1)

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

someFunction()

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

someFunction()

someFunction()

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

someFunction()

someFunction()

someFunction()

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

someFunction()

someFunction()

someFunction()

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

someFunction()

someFunction()

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

someFunction()

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Recursion

The process of functions repeatedly calling themselves

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

Back-Paging

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

Back-Paging

Stack

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

Back-Paging

I'm

Stack

Google

Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

Back-Paging

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

These
Typing
I'm
Stack

Back-Paging

Twitter
Amazon
Google
Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

These
Typing
I'm
Stack

Back-Paging

Twitter
Amazon
Google
Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

These
Typing
I'm
Stack

Back-Paging

Twitter
Amazon
Google
Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

These
Typing
I'm
Stack

Back-Paging

Twitter
Amazon
Google
Stack

The Stack - Uses for Stacks

- Stacks are used everywhere, both in the actual writing of code as well as in real-world situations

Undo/Redo

Back-Paging

Recursion

The Stack - Conclusion

- **The Stack**
  - A **sequential access data structure** in which we use the **LIFO principle** to add and remove elements from
    - Last In First Out

An Introduction to Data Structures

The Queue

The Queue - Introduction

The Stack

The Queue - Introduction

The Stack

Sequential Access Data Structure

Follows the LIFO Principle

The Queue - Introduction

The Stack

Sequential Access Data Structure

Follows the LIFO Principle

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Stack ⟷ **The Queue**

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

LIFO - Last in First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

Last Element Pushed

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

FIFO - First in First Out

Queue

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

FIFO - First in First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

FIFO - First in First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

FIFO - First in First Out

Queue

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Queue Basics

- **The Queue**
  - A **sequential access data structure** which follows the **FIFO methodology**
    - First In First Out

The Queue - Common Queue Methods

Stack Method

Queue Method

The Queue - Common Queue Methods

Stack Method

Queue Method

Push Method

The Queue - Common Queue Methods

Stack Method

Push Method

Pop Method

Queue Method

The Queue - Common Queue Methods

Stack Method	Queue Method
Push Method	Enqueue Method
Pop Method	

The Queue - Common Queue Methods

Stack Method	Queue Method
Push Method	Enqueue Method
Pop Method	Dequeue Method

The Queue - Common Queue Methods

Stack Method	Queue Method
Push Method	Enqueue Method
Pop Method	Dequeue Method
Peek Method	Peek Method

The Queue - Common Queue Methods

Stack Method	Queue Method
Push Method	Enqueue Method
Pop Method	Dequeue Method
Peek Method	Peek Method
Contains Method	Contains Method

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

exampleQueue

Size: 0

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

```
exampleQueue.enqueue("now")
```

exampleQueue

Size: 0

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

exampleQueue.enqueue("now")

"now"

exampleQueue

Size: 0

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

```
exampleQueue.enqueue("now")
```

"now"

exampleQueue

Size: 1

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

exampleQueue.enqueue("video")

"now"

exampleQueue

Size: 1

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

exampleQueue.enqueue("video")

"video" "now"

exampleQueue

Size: 2

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

exampleQueue.enqueue("this")

"video" "now"

exampleQueue

Size: 2

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

exampleQueue.enqueue("this")

"this" "video" "now"

exampleQueue

Size: 3

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

exampleQueue.enqueue("like")

"this" "video" "now"

exampleQueue

Size: 3

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

Enqueue (Object)

exampleQueue.enqueue("like")

"like" "this" "video" "now"

exampleQueue

Size: 4

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

The Queue - Enqueue Method

- Adds an element to the tail of the Queue

“like” “this” “video” “now”

exampleQueue

Size: 4

The Queue - Dequeue Method

- Removes an element from the head of the Queue

“like” “this” “video” “now”

exampleQueue

Size: 4

The Queue - Dequeue Method

- Removes an element from the head of the Queue

Dequeue()

“like” “this” “video” “now”

exampleQueue

Size: 4

The Queue - Dequeue Method

- Removes an element from the head of the Queue

Dequeue()

exampleQueue.dequeue()

"like" "this" "video" "now"

exampleQueue

Size: 4

The Queue - Dequeue Method

- Removes an element from the head of the Queue

The Queue - Dequeue Method

- Removes an element from the **head** of the Queue

Dequeue()

Returned

now

exampleQueue.dequeue()

"like"

"this"

"video"

exampleQueue

Size: 3

The Queue - Dequeue Method

- Removes an element from the **head** of the Queue

The Queue - Dequeue Method

- Removes an element from the head of the Queue

The Queue - Dequeue Method

- Removes an element from the head of the Queue

Dequeue()

Returned

video

exampleQueue.dequeue()

"like"

"this"

exampleQueue

Size: 2

The Queue - Dequeue Method

- Removes an element from the **head** of the Queue

Dequeue()

Returned

video

exampleQueue.dequeue()

"like"

"this"

exampleQueue

Size: 2

The Queue - Dequeue Method

- Removes an element from the **head** of the Queue

Dequeue()

Returned

video

exampleQueue.dequeue()

"like"

"this"

exampleQueue

Size: 2

The Queue - Dequeue Method

- Removes an element from the head of the Queue

Dequeue()

Returned

video

exampleQueue.dequeue()

"like"

"this"

exampleQueue

Size: 2

The Queue - Dequeue Method

- Removes an element from the head of the Queue

Dequeue()

exampleQueue.dequeue()

"like" "this" "video" "now"

exampleQueue

Size: 4

The Queue - Dequeue Method

- Removes an element from the head of the Queue

“like” “this” “video” “now”

exampleQueue

Size: 4

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

"like" | "this" | "video" | "now"

exampleQueue

Size: 4

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

Peek()

"like" | "this" | "video" | "now"

exampleQueue

Size: 4

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

Peek()

"like" | "this" | "video" | "now"

exampleQueue

Size: 4

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

Peek()

"like" | "this" | "video" | "now"

exampleQueue

Size: 4

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

Peek()

exampleQueue.peek()

"like" "this" "video" "now"

exampleQueue

Size: 4

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

Peek()

Returned

exampleQueue.peek()

"like"

"this"

exampleQueue

Size: 2

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

Peek()

Returned

"this"

exampleQueue.peek()

"like"

"this"

exampleQueue

Size: 2

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

Peek()

exampleQueue.peek()

"like" "this" "video" "now"

exampleQueue

Size: 4

The Queue - Peek Method

- Returns the Object that's at the forefront of our Queue

"like" | "this" | "video" | "now"

exampleQueue

Size: 4

The Queue - Contains Method

- Returns whether or not the Queue contains an object

“like” “this” “video” “now”

exampleQueue

Size: 4

The Queue - Contains Method

- Returns whether or not the Queue contains an object

Contains(Object)

“like” “this” “video” “now”

exampleQueue

Size: 4

The Queue - Contains Method

- Returns whether or not the Queue **contains** an object

Contains(Object)

exampleQueue.contains("Queue")

"like" | "this" | "video" | "now"

exampleQueue

Size: 4

The Queue - Contains Method

- Returns whether or not the Queue **contains** an object

Contains(Object)

Returned

False

exampleQueue.contains("Queue")

"like" "this" "video" "now"

exampleQueue

Size: 4

The Queue - Contains Method

- Returns whether or not the Queue **contains** an object

The Queue - Contains Method

- Returns whether or not the Queue **contains** an object

Contains(Object)

Returned

False

exampleQueue.contains("Queue")

"like" "this" "video" "now"

exampleQueue

Size: 4

The Queue - Contains Method

- Returns whether or not the Queue **contains** an object

The Queue - Contains Method

- Returns whether or not the Queue **contains** an object

Contains(Object)

Returned

True

exampleQueue.contains("video")

"like" | "this" | "video" | "now"

exampleQueue

Size: 4

The Queue - Contains Method

- Returns whether or not the Queue **contains** an object

The Queue - Contains Method

- Returns whether or not the Queue **contains** an object

Contains(Object)

Returned

True

exampleQueue.contains("video")

"like" | "this" | "video" | "now"

exampleQueue

Size: 4

The Queue - Common Queue Methods

Queue Methods

Enqueue (Object)

Dequeue ()

Peek ()

Contains (Object)

The Queue - Time Complexity Equations

The Queue - Time Complexity Equations

Accessing

The Queue - Time Complexity Equations

Accessing

O(n)

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Element you Want

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Element you Want

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Element you Want

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Element you Want

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Element you Want

The Queue - Time Complexity Equations

Accessing

O(n)

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Queue - Time Complexity Equations

Accessing

O(n)

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Queue - Time Complexity Equations

Accessing

O(n)

The Queue - Time Complexity Equations

Accessing

O(n)

someQueue

The Queue - Time Complexity Equations

Accessing

O(n)

The Queue - Time Complexity Equations

Accessing

O(n)

Searching

The Queue - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The Queue - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

The Queue - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Deleting

The Queue - Time Complexity Equations

Accessing → O(n)

Searching → O(n)

Inserting → O(1)

Deleting → O(1)

The Queue - Time Complexity Equations

Accessing

O(n)

Deleting

O(1)

The Queue - Common Queue Uses

- Queues are used very often in programming, for a variety of functions

The Queue - Common Queue Uses

- Queues are used very often in programming, for a variety of functions

Job Scheduling

The Queue - Common Queue Uses

- Queues are used **very often** in programming, for a **variety** of functions

Job Scheduling

Printer Queueing

The Queue - Common Queue Uses

- Queues are used **very often** in programming, for a **variety** of functions

Job Scheduling

Printer Queueing

Modern Cameras

The Queue - Conclusion

The Queue - Conclusion

The Queue

The Queue - Conclusion

The Queue

Sequential Access Data Structure

The Queue - Conclusion

The Queue

Sequential Access Data Structure

Follows FIFO Principle (First In First Out)

The Queue - Conclusion

The Queue

Sequential Access Data Structure

Follows FIFO Principle (First In First Out)

Add to Back. Remove from Front

An Introduction to Data Structures

The LinkedList

The Linked List - LinkedList Basics

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

Salaries	10,000	12,500	8,750	15,000

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

Salaries	10,000	12,500	8,750	15,000

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

Salaries	10,000	12,500	8,750	15,000

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A **LinkedList** is a **sequential access** linear data structure in which every element is a separate **object** called a **Node**, which has 2 parts
  - The **data**
  - The **reference** (or pointer) which points to the next **Node** in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

Node

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

Node

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - LinkedList Basics

- A LinkedList is a sequential access linear data structure in which every element is a separate object called a Node, which has 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The Linked List - Linked List Visualization

Linked List

The Linked List - Linked List Visualization

Linked List

Head Node

The Linked List - Linked List Visualization

Linked List

Head Node

Data	Reference/ Pointer

The Linked List - Linked List Visualization

Linked List

Head Node

Data	Reference/ Pointer
1	

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Linked List Visualization

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

Data can only flow in and out of a few specified points

The Linked List - Adding and Removing Information

Data can only flow in and out of a few specified points

The Linked List - Adding and Removing Information

Data can only flow in and out of a few specified points

The Linked List - Adding and Removing Information

Data can only flow in and out of a few specified points

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

Add to Head

Remove from Head

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

Add to Head

Add to the Middle

Remove from Head

Remove from the Middle

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

Add to Head	Add to the Middle	Add to Tail
Remove from Head	Remove from the Middle	Remove from Tail

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

Data can flow in and out of any point of a LinkedList

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

Adding to the Head of a LinkedList

The Linked List - Adding and Removing Information

Adding to the Head of a LinkedList

Make that new Node's pointer point to the current Head of the LinkedList

The Linked List - Adding and Removing Information

Adding to the Head of a LinkedList

Make that new Node's pointer point to the current Head of the LinkedList

The Linked List - Adding and Removing Information

Adding to the Head of a LinkedList

Make that new Node's pointer point to the current Head of the LinkedList

The Linked List - Adding and Removing Information

Adding to the Head of a LinkedList

Make that new Node's pointer point to the current Head of the LinkedList

The Linked List - Adding and Removing Information

Adding to the Head of a LinkedList

Make that new Node's pointer point to the current Head of the LinkedList

The Linked List - Adding and Removing Information

Adding to the Head of a LinkedList

Make that new Node's pointer point to the current Head of the LinkedList

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

Removing from the Head of a LinkedList

The Linked List - Adding and Removing Information

Removing from the Head of a LinkedList

Set the Head Node's pointer to a null value

The Linked List - Adding and Removing Information

Removing from the Head of a LinkedList

Set the Head Node's pointer to a null value

The Linked List - Adding and Removing Information

Removing from the Head of a LinkedList

Set the Head Node's pointer to a null value

The Linked List - Adding and Removing Information

Removing from the Head of a LinkedList

Set the Head Node's pointer to a null value

The Linked List - Adding and Removing Information

Removing from the Head of a LinkedList

Set the Head Node's pointer to a null value

The Linked List - Adding and Removing Information

Removing from the Head of a LinkedList

Set the Head Node's pointer to a null value

The Linked List - Adding and Removing Information

Removing from the Head of a LinkedList

Set the Head Node's pointer to a null value

The Linked List - Adding and Removing Information

Removing from the Head of a LinkedList

Set the Head Node's pointer to a null value

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

Make the pointer of the new Node point to the Node after the location we want to insert at

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

Make the pointer of the new Node point to the Node after the location we want to insert at

Set the Node before the location we want to insert at to point towards the new Node

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

Make the pointer of the new Node point to the Node after the location we want to insert at

Set the Node before the location we want to insert at to point towards the new Node

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

Make the pointer of the new Node point to the Node after the location we want to insert at

Set the Node before the location we want to insert at to point towards the new Node

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

Make the pointer of the new Node point to the Node after the location we want to insert at

Set the Node before the location we want to insert at to point towards the new Node

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

Make the pointer of the new Node point to the Node after the location we want to insert at

Set the Node before the location we want to insert at to point towards the new Node

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

Make the pointer of the new Node point to the Node after the location we want to insert at

Set the Node before the location we want to insert at to point towards the new Node

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

Make the pointer of the new Node point to the Node after the location we want to insert at

Set the Node before the location we want to insert at to point towards the new Node

The Linked List - Adding and Removing Information

Adding a Node to the Middle of a LinkedList

Make the pointer of the new Node point to the Node after the location we want to insert at

Set the Node before the location we want to insert at to point towards the new Node

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

Removing a Node from the middle of a LinkedList

The Linked List - Adding and Removing Information

Removing a Node from the middle of a LinkedList

Make the pointer of the Node previous to the one we're removing, to now point to the Node after the one we're removing

The Linked List - Adding and Removing Information

Removing a Node from the middle of a LinkedList

Make the pointer of the Node previous to the one we're removing, to now point to the Node after the one we're removing

The Linked List - Adding and Removing Information

Removing a Node from the middle of a LinkedList

Make the pointer of the Node previous to the one we're removing, to now point to the Node after the one we're removing

The Linked List - Adding and Removing Information

Removing a Node from the middle of a LinkedList

Make the pointer of the Node previous to the one we're removing, to now point to the Node after the one we're removing

The Linked List - Adding and Removing Information

Removing a Node from the middle of a LinkedList

Make the pointer of the Node previous to the one we're removing, to now point to the Node after the one we're removing

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

Adding to the Tail of a LinkedList

The Linked List - Adding and Removing Information

Adding to the Tail of a LinkedList

Make the current tail point towards the new Node you want to add

The Linked List - Adding and Removing Information

Adding to the Tail of a LinkedList

Make the current tail point towards the new Node you want to add

The Linked List - Adding and Removing Information

Adding to the Tail of a LinkedList

Make the current tail point towards the new Node you want to add

The Linked List - Adding and Removing Information

Adding to the Tail of a LinkedList

Make the current tail point towards the new Node you want to add

The Linked List - Adding and Removing Information

Adding to the Tail of a LinkedList

Make the current tail point towards the new Node you want to add

The Linked List - Adding and Removing Information

The Linked List - Adding and Removing Information

Removing from the Tail of a LinkedList

The Linked List - Adding and Removing Information

Removing from the Tail of a LinkedList

Set the previous tail to point towards a null value instead of the current tail

The Linked List - Adding and Removing Information

Removing from the Tail of a LinkedList

Set the previous tail to point towards a null value instead of the current tail

The Linked List - Adding and Removing Information

Removing from the Tail of a LinkedList

Set the previous tail to point towards a null value instead of the current tail

The Linked List - Adding and Removing Information

Removing from the Tail of a LinkedList

Set the previous tail to point towards a null value instead of the current tail

The Linked List - Adding and Removing Information

Removing from the Tail of a LinkedList

Set the previous tail to point towards a null value instead of the current tail

The Linked List - Adding and Removing Information

Removing from the Tail of a LinkedList

Set the previous tail to point towards a null value instead of the current tail

The LinkedList - Time Complexity Equations

The LinkedList - Time Complexity Equations

Accessing

The LinkedList - Time Complexity Equations

Accessing

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Deleting

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Deleting

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Deleting

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Deleting

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Deleting

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Deleting

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

Deleting

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

O(n)

Deleting

O(n)

The LinkedList - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

O(n) O(1)

Deleting

O(n) O(1)

The LinkedList - Uses for LinkedLists

The LinkedList - Uses for LinkedLists

- LinkedLists can be used in the backing of other data structures
  - We can use LinkedLists to make Stacks, Queues, etc.

The LinkedList - Uses for LinkedLists

- LinkedLists can be used in the backing of other data structures
  - We can use LinkedLists to make Stacks, Queues, etc.

The LinkedList - Uses for LinkedLists

- LinkedLists can be used in the backing of other data structures
  - We can use LinkedLists to make Stacks, Queues, etc.

The LinkedList - Uses for LinkedLists

- LinkedLists can be used in the backing of other data structures
  - We can use LinkedLists to make Stacks, Queues, etc.

The LinkedList - Uses for LinkedLists

- LinkedLists can be used in the backing of other data structures
  - We can use LinkedLists to make Stacks, Queues, etc.

The LinkedList - Uses for LinkedLists

- LinkedLists can be used in the backing of other data structures
  - We can use LinkedLists to make Stacks, Queues, etc.

The LinkedList - Uses for LinkedLists

- LinkedLists can be used in the backing of other data structures
  - We can use LinkedLists to make Stacks, Queues, etc.

Article in The Description

The LinkedList - Uses for LinkedLists

The LinkedList - Uses for LinkedLists

NEXT TRACKS

#		SONG	ARTIST	ALBUM	
1	+	Don't Believe Her	Scorpions	Crazy World	4:56
2	+	To Be With You In Heaven	Scorpions	Crazy World	4:52
3	+	Wind Of Change	Scorpions	Crazy World	5:13
4	+	Restless Nights	Scorpions	Crazy World	5:48
5	+	Lust Or Love	Scorpions	Crazy World	4:23
6	+	Kicks After Six	Scorpions	Crazy World	3:50
7	+	Hit Between The Eyes	Scorpions	Crazy World	4:34
8	+	Money And Fame	Scorpions	Crazy World	5:07
9	+	Crazy World	Scorpions	Crazy World	5:09
10	+	Send Me An Angel	Scorpions	Crazy World	4:33

The LinkedList - Uses for LinkedLists

NEXT TRACKS

#		SONG	ARTIST	ALBUM	
1	+	Don't Believe Her	Scorpions	Crazy World	
2	+	To Be With You In Heaven	Scorpions	Crazy World	
3	+	Wind Of Change	Scorpions	Crazy World	
4	+	Restless Nights	Scorpions	Crazy World	5:48
5	+	Lust Or Love	Scorpions	Crazy World	4:23
6	+	Kicks After Six	Scorpions	Crazy World	3:50
7	+	Hit Between The Eyes	Scorpions	Crazy World	4:34
8	+	Money And Fame	Scorpions	Crazy World	5:07
9	+	Crazy World	Scorpions	Crazy World	5:09
10	+	Send Me An Angel	Scorpions	Crazy World	4:33

The LinkedList - Uses for LinkedLists

NEXT TRACKS

#		SONG	ARTIST	ALBUM	
1	+	Don't Believe Her	Scorpions	Crazy World	
2	+	To Be With You In Heaven	Scorpions	Crazy World	
	+	Wind Of Change	Scorpions	Crazy World	
	+	Restless Nights	Scorpions	Crazy World	5:48
	+	Lust Or Love	Scorpions	Crazy World	4:23
6	+	Kicks After Six	Scorpions	Crazy World	3:50
7	+	Hit Between The Eyes	Scorpions	Crazy World	4:34
8	+	Money And Fame	Scorpions	Crazy World	5:07
9	+	Crazy World	Scorpions	Crazy World	5:09
10	+	Send Me An Angel	Scorpions	Crazy World	4:33

Title

The LinkedList - Uses for LinkedLists

The LinkedList - Uses for LinkedLists

The LinkedList - Review + Conclusion

- A LinkedList is sequential access linear data structure in which every element is a separate object called a node, containing 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The LinkedList - Review + Conclusion

- A LinkedList is sequential access linear data structure in which every element is a separate object called a node, containing 2 parts
  - The data
  - The reference (or pointer) which points to the next Node in the List

The LinkedList - One Big Drawback

The LinkedList - One Big Drawback

The LinkedList - One Big Drawback

The LinkedList - One Big Drawback

The LinkedList - One Big Drawback

The LinkedList - One Big Drawback

An Introduction to Data Structures

The Doubly-LinkedList

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

Node

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Background Information

- **The Doubly-Linked List** is a **sequential access data structure** which stores data in the form of Nodes
  - Able to traverse both **forwards** and **backwards** using pointers

The Doubly-Linked List - Visualization

"Next" = That particular Nodes pointer which points to the next object in the List

The Doubly-Linked List - Visualization

"Next" = That particular Nodes pointer which points to the next object in the List

"Previous" = That particular Nodes pointer which points to the previous object in the List

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List

The Doubly-Linked List - Visualization

The Doubly-Linked List

Head Node

The Doubly-Linked List - Visualization

The Doubly-Linked List

Head Node

Pointer	Data	Pointer
•	1	•

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Visualization

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

Add to Head

Remove from Head

The Doubly-Linked List - Adding and Removing Information

Add to Head

Add to the Middle

Remove from Head

Remove from the Middle

The Doubly-Linked List - Adding and Removing Information

Add to Head	Add to the Middle	Add to Tail
Remove from Head	Remove from the Middle	Remove from Tail

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

Set the current head's previous to point towards this new Node

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

Set the current head's previous to point towards this new Node

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

Set the current head's previous to point towards this new Node

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

Set the current head's previous to point towards this new Node

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

Set the current head's previous to point towards this new Node

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

Set the current head's previous to point towards this new Node

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

Set the current head's previous to point towards this new Node

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

Set the current head's previous to point towards this new Node

The Doubly-Linked List - Adding and Removing Information

Adding to the Head of a Doubly-LinkedList

Set the new Nodes next to point towards the current head of the List

Take the new Node that we want to insert, and set it's previous to null

Set the current head's previous to point towards this new Node

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

Set the second Node's previous to also point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

Set the second Node's previous to also point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

Set the second Node's previous to also point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

Set the second Node's previous to also point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

Set the second Node's previous to also point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

Set the second Node's previous to also point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

Set the second Node's previous to also point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

Set the second Node's previous to also point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Head of a Doubly-LinkedList

Set the head Node's next to point towards a null value

Set the second Node's previous to also point towards a null value

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

Inserting into the Middle of a Doubly-LinkedList

Set the new Node's previous to point towards the Node previous to the position you want to insert at

Set the new Node's next to point towards the Node after the position you want to insert at

Set the next and previous on the Node's before and after the one you're inserting to point towards the new Node

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

Removing from the Middle of a Doubly-LinkedList

Set the Node before the one we want to remove's next to point towards the Node after the one we want to remove

Set the Node after the one we want to remove's previous to point towards the Node before the one we want to remove

Set both pointers of the Node we want to remove to point towards a null value

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

Make the new Node's next point towards a Null value

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

Make the new Node's next point towards a Null value

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

Make the new Node's next point towards a Null value

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

Make the new Node's next point towards a Null value

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

Make the new Node's next point towards a Null value

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

Make the new Node's next point towards a Null value

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

Make the new Node's next point towards a Null value

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

Make the new Node's next point towards a Null value

The Doubly-Linked List - Adding and Removing Information

Add to the Tail of a Doubly-LinkedList

Set the next pointer of the current tail to point towards the new Node we want to become the tail

Set the previous of the new Node that we're adding to be pointing towards the current tail of the List

Make the new Node's next point towards a Null value

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

Remove from the Tail of a Doubly-LinkedList

The Doubly-Linked List - Adding and Removing Information

Remove from the Tail of a Doubly-LinkedList

Set the tail Node's previous to point towards null

The Doubly-Linked List - Adding and Removing Information

Remove from the Tail of a Doubly-LinkedList

Set the tail Node's previous to point towards null

Set the second to last Node's next to also point towards null

The Doubly-Linked List - Adding and Removing Information

Remove from the Tail of a Doubly-LinkedList

Set the tail Node's previous to point towards null

Set the second to last Node's next to also point towards null

The Doubly-Linked List - Adding and Removing Information

Remove from the Tail of a Doubly-LinkedList

Set the tail Node's previous to point towards null

Set the second to last Node's next to also point towards null

The Doubly-Linked List - Adding and Removing Information

Remove from the Tail of a Doubly-LinkedList

Set the tail Node's previous to point towards null

Set the second to last Node's next to also point towards null

The Doubly-Linked List - Adding and Removing Information

Remove from the Tail of a Doubly-LinkedList

Set the tail Node's previous to point towards null

Set the second to last Node's next to also point towards null

The Doubly-Linked List - Adding and Removing Information

Remove from the Tail of a Doubly-LinkedList

Set the tail Node's previous to point towards null

Set the second to last Node's next to also point towards null

The Doubly-Linked List - Adding and Removing Information

Remove from the Tail of a Doubly-LinkedList

Set the tail Node's previous to point towards null

Set the second to last Node's next to also point towards null

The Doubly-Linked List - Adding and Removing Information

The Doubly-Linked List - Adding and Removing Information

- We only have to use this pseudocode to program these functions once, then we're able to use them an infinite amount of times

The Doubly-Linked List - Adding and Removing Information

- We only have to use this pseudocode to program these functions once, then we're able to use them an infinite amount of times

The Doubly-Linked List - Time Complexity Equations

The Doubly-Linked List - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

O(n) O(1)

Deleting

O(n) O(1)

The Doubly-Linked List - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

O(n) O(1)

Deleting

O(n) O(1)

The Doubly-Linked List - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

O(n) O(1)

Deleting

O(n) O(1)

The Doubly-Linked List - Time Complexity Equations

Accessing

O(n)

Searching

O(n)

Inserting

O(n) O(1)

Deleting

O(n) O(1)

The Doubly-Linked List - Uses of a Doubly-Linked List

- The back and forth functionality of a Doubly-Linked List lends itself to be implemented in a lot of Stack-like functionality

The Doubly-Linked List - Uses of a Doubly-Linked List

- The back and forth functionality of a Doubly-Linked List lends itself to be implemented in a lot of Stack-like functionality

The Doubly-Linked List - Uses of a Doubly-Linked List

- The back and forth functionality of a Doubly-Linked List lends itself to be implemented in a lot of Stack-like functionality

The Doubly-Linked List - Uses of a Doubly-Linked List

- The back and forth functionality of a Doubly-Linked List lends itself to be implemented in a lot of Stack-like functionality

The Doubly-Linked List - Conclusion

The Doubly-Linked List - Conclusion

The Doubly-Linked List - Conclusion

The Doubly-Linked List - Conclusion

The Doubly-Linked List - Conclusion

The Doubly-Linked List - Conclusion

An Introduction to Data Structures

Dictionaries

Dictionaries - Introduction

Dictionaries - Introduction

Dictionaries - Introduction

Dictionaries - Introduction

- Dictionaries are one of the most abstract data structures we'll cover
  -

Dictionaries - Introduction

- Dictionaries are one of the most abstract data structures we'll cover
  - They are also called Maps and Associative Arrays

Dictionaries - Introduction

- Dictionaries are one of the most abstract data structures we'll cover
  - They are also called Maps and Associative Arrays

Dictionaries

Dictionaries - Introduction

- Dictionaries are one of the most abstract data structures we'll cover
  - They are also called Maps and Associative Arrays

Dictionaries

Maps

Dictionaries - Introduction

- Dictionaries are one of the most abstract data structures we'll cover
  - They are also called Maps and Associative Arrays

Dictionaries

Maps

Associative Arrays

Dictionaries - Introduction

- Dictionaries are one of the most abstract data structures we'll cover
  - They are also called Maps and Associative Arrays

Dictionaries = Maps = Associative Arrays

Dictionaries - Introduction

- Dictionaries are one of the most abstract data structures we'll cover
  - They are also called Maps and Associative Arrays

Dictionaries = Maps = Associative Arrays

Dictionaries - Introduction

- Dictionaries are one of the most abstract data structures we'll cover
  - They are also called Maps and Associative Arrays

Dictionaries = Maps = Associative Arrays

Dictionaries - Dictionary Basics

- **Dictionary**
  - An abstract Data Structure which stores data in the form of **key/value pairs**

Dictionaries - Dictionary Basics

- **Dictionary**
  - An abstract Data Structure which stores data in the form of **key/value pairs**

Dictionaries - Dictionary Basics

- **Dictionary**
  - An abstract Data Structure which stores data in the form of **key/value pairs**

Value

Dictionaries - Dictionary Basics

- **Dictionary**
  - An abstract Data Structure which stores data in the form of **key/value pairs**

Dictionaries - Dictionary Basics

- **Dictionary**
  - An abstract Data Structure which stores data in the form of **key/value pairs**

Dictionaries - Dictionary Basics

- **Dictionary**
  - An abstract Data Structure which stores data in the form of **key/value pairs**

Dictionaries - Dictionary Basics

- Think of a key/value pair like a social security number

Dictionaries - Dictionary Basics

- Think of a key/value pair like a social security number

000-00-0000

Dictionaries - Dictionary Basics

- Think of a key/value pair like a social security number

000-00-0000

Dictionaries - Dictionary Basics

- Think of a key/value pair like a social security number

Value

000-00-0000

Dictionaries - Dictionary Basics

- Think of a key/value pair like a social security number

Value

000-00-0000

John Doe

Dictionaries - Dictionary Basics

- Think of a key/value pair like a social security number

Dictionaries - Indexing Dictionaries

- We index dictionaries using these keys instead of a numerical index

Dictionaries - Indexing Dictionaries

- We index dictionaries using these keys instead of a numerical index

Array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

Dictionaries - Indexing Dictionaries

- We index dictionaries using these keys instead of a numerical index

Array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

Dictionaries - Indexing Dictionaries

- We index dictionaries using these keys instead of a numerical index

Array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

Dictionaries - Indexing Dictionaries

- We index dictionaries using these keys instead of a numerical index

Array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

Dictionaries - Indexing Dictionaries

- We index dictionaries using these keys instead of a numerical index

Array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

Dictionary

Index										
Numbers	1	2	3	4	5	6	7	8	9	10

Dictionaries - Indexing Dictionaries

- We index dictionaries using these keys instead of a numerical index

Array

Index	0	1	2	3	4	5	6	7	8	9
Numbers	1	2	3	4	5	6	7	8	9	10

Dictionary

Index	key	key	key	key	key	key	key	key	key	key
Numbers	1	2	3	4	5	6	7	8	9	10

Dictionaries - Dictionary Properties

- The key's in a key/value pair can be pretty much any primitive data type you can think of

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values										

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values										

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values										

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values										

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values										

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values										

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values										

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values										

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"John"	"Sam"	"Earl"

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values										

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"John"	"Sam"	"Earl"

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values										

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"John"	"Sam"	"Earl"

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values	true	true	false	false	true	false	true	false	false	true

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"John"	"Sam"	"Earl"

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values	true	true	false	false	true	false	true	false	false	true

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values										

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"John"	"Sam"	"Earl"

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values	true	true	false	false	true	false	true	false	false	true

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"John"	"Sam"	"Earl"

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values	true	true	false	false	true	false	true	false	false	true

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary

Dictionaries - Dictionary Properties

- The **key's** in a **key/value pair** can be pretty much **any** primitive data type you can think of

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"John"	"Sam"	"Earl"

Keys	"Apple"	"Pear"	"Corn"	"Bread"	"Beans"	"Fries"	"Beef"	"Pork"	"Mint"	"Chips"
Values	true	true	false	false	true	false	true	false	false	true

Keys	4.5	6.7	1.3	3.8	9.0	10.1	4.4	7.8	5.4	2.3
Values	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary	Dictionary

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  -
  -

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  -

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Dictionaries - Dictionary Properties

- There are 2 extremely important **restrictions** when it comes to dictionaries
  - Every <u>key</u> can only appear **once** within the dictionary
  - Each <u>key</u> can only have **one value**

Dictionaries - Dictionary Properties

- There are 2 extremely important **restrictions** when it comes to dictionaries
  - Every <u>key</u> can only appear **once** within the dictionary
  - Each <u>key</u> can only have **one value**

Unique

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Memory

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Memory

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Dictionary

Dictionaries - Dictionary Properties

- There are 2 extremely important **restrictions** when it comes to dictionaries
  - Every <u>key</u> can only appear **once** within the dictionary
  - Each <u>key</u> can only have **one value**

Dictionary

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Dictionary

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Dictionaries - Dictionary Properties

- There are 2 extremely important **restrictions** when it comes to dictionaries
  - Every <u>key</u> can only appear **once** within the dictionary
  - Each <u>key</u> can only have **one value**

Dictionaries - Dictionary Properties

- There are 2 extremely important **restrictions** when it comes to dictionaries
  - Every <u>key</u> can only appear **once** within the dictionary
  - Each <u>key</u> can only have **one value**

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

- There can be duplicates of values within a dictionary

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

- There can be duplicates of values within a dictionary

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"Paul"	"Sam"	"Earl"

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

- There can be duplicates of values within a dictionary

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"Paul"	"Sam"	"Earl"

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

- There can be duplicates of values within a dictionary

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"Paul"	"Sam"	"Earl"

Dictionaries - Dictionary Properties

- There are 2 extremely important restrictions when it comes to dictionaries
  - Every key can only appear once within the dictionary
  - Each key can only have one value

- There can be duplicates of values within a dictionary

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"Paul"	"Sam"	"Earl"

Dictionaries - Time Complexity Equations?

Dictionaries - Time Complexity Equations?

Dictionaries - Time Complexity Equations?

Dictionaries - Time Complexity Equations?

Dictionaries - Time Complexity Equations?

Dictionaries - Time Complexity Equations?

Keys	0	1	2	3	4	5	6	7	8	9
Values	“Abe”	“Carl”	“Paul”	“Gabe”	“Zack”	“Jack”	“Sean”	“Paul”	“Sam”	“Earl”

Dictionaries - Time Complexity Equations?

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"Paul"	"Sam"	"Earl"

Dictionaries - Time Complexity Equations?

Keys	0	1	2	3	4	5	6	7	8	9
Values	"Abe"	"Carl"	"Paul"	"Gabe"	"Zack"	"Jack"	"Sean"	"Paul"	"Sam"	"Earl"

Dictionaries - Hash Table Mini-Lesson

Dictionaries - Hash Table Mini-Lesson

Index	Key

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	0
1	
2	2
3	
4	4
5	
6	
7	7
8	8
9	

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	0
1	
2	2
3	
4	4
5	
6	
7	7
8	8
9	

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	
1	
2	2
3	
4	4
5	
6	
7	7
8	8
9	

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	0
1	
2	2
3	
4	4
5	
6	
7	7
8	8
9	

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	0
1	
2	2
3	
4	4
5	
6	
7	7
8	8
9	

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	0
1	
2	2
3	
4	
5	
6	
7	7
8	8
9	

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	0
1	
2	2
3	
4	4
5	
6	
7	7
8	8
9	

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	0
1	
2	2
3	
4	4
5	
6	
7	7
8	8
9	

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	0
1	nil
2	2
3	nil
4	4
5	nil
6	nil
7	7
8	8
9	nil

Dictionaries - Hash Table Mini-Lesson

Index	Key
0	0
1	nil
2	2
3	nil
4	4
5	nil
6	nil
7	7
8	8
9	nil

Dictionaries - Hash Table Mini-Lesson

WHY DOESN'T THIS WORK?

Index	Key
0	0
1	nil
2	2
3	nil
4	4
5	nil
6	nil
7	7
8	8
9	nil

Dictionaries - Hash Table Mini-Lesson

WHY DOESN'T THIS WORK?

- This doesn't work because it's based upon the simple assumption that the size of the array is not too large

Index	Key
0	0
1	nil
2	2
3	nil
4	4
5	nil
6	nil
7	7
8	8
9	nil

Dictionaries - Hash Table Mini-Lesson

WHY DOESN'T THIS WORK?

- This doesn't work because it's based upon the simple assumption that the size of the array is not too large

Index	Key
0	0
1	nil
2	2
3	nil
4	4
5	nil
6	nil
7	7
8	8
9	nil

Dictionaries - Hash Table Mini-Lesson

WHY DOESN'T THIS WORK?

- This doesn't work because it's based upon the simple assumption that the size of the array is not too large

Dictionaries - Hash Table Mini-Lesson

WHY DOESN'T THIS WORK?

- This doesn't work because it's based upon the simple assumption that the size of the array is not too large

Index	Key
0	1
10	10
100	100
1,000	1,000
10,000	10,000
100K	100K
1 Million	1 Million
10 Million	10 Million
100 Million	100 Million
1 Billion	1 Billion

Dictionaries - Hash Table Mini-Lesson

WHY DOESN'T THIS WORK?

- This doesn't work because it's based upon the simple assumption that the size of the array is not too large

9,999,990 Nil Values

Index	Key
0	1
10	10
100	100
1,000	1,000
10,000	10,000
100K	100K
1 Million	1 Million
10 Million	10 Million
100 Million	100 Million
1 Billion	1 Billion

Dictionaries - Hash Table Mini-Lesson

Dictionaries - Hash Table Mini-Lesson

- Hash tables are fundamentally a way to store this information in such a way that we're able to cut down the amount of nil values while also allowing for the information to be stored in a way that is easily accessible

Dictionaries - Hash Table Mini-Lesson

- Hash tables are fundamentally a way to **store** this information in such a way that we're able to cut down the amount of **nil values** while also allowing for the information to be stored in a way that is **easily accessible**

Index										
Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion

Dictionaries - Hash Table Mini-Lesson

- Hash tables are fundamentally a way to store this information in such a way that we're able to cut down the amount of nil values while also allowing for the information to be stored in a way that is easily accessible

Index	0	1	2	3	4	5	6	7	8	9
Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion

Dictionaries - Hash Table Mini-Lesson

- Hash tables are fundamentally a way to **store** this information in such a way that we're able to cut down the amount of **nil values** while also allowing for th[illegible]tion to be stored in a way that is **easily accessible**

Hash Function

Index	0	1	2	3	4	5	[illegible]	7	8	9
Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion

Dictionaries - Hash Table Mini-Lesson

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Hash Function

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

- A **Hash function** will take all the keys for a given dictionary and **strategically map** them to certain index locations in an array so that they can eventually be **retrieved easily**

- The goal of a **good hashing function** is to take in a key and **reliably** place it somewhere in the table so that it can be **accessed later** by the computer

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Dictionaries - Hash Table Mini-Lesson

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Hash Function

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

(1,000,000

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

(1,000,000/1,000,000)

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

(1,000,000/1,000,000) x

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

(1,000,000/1,000,000) x (7-1)

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index										

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

(1,000,000/1,000,000) x (7-1) = 6

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index							6			

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

(1,000,000/1,000,000) x (7-1) = 6

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index							6			

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

Hash Function

Take the key, divide it by itself, then multiply the result by the number of digits in the key minus 1

↓

We have consolidated the 10 keys from 1 to a billion into 10 index slots instead of 1 billion

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

Hash Function

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

Hash Function

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

1,000,000

Hash Function

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

1,000,000 → Hash Function

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

1,000,000 → Hash Function

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

1,000,000 → Hash Function →

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

1,000,000 → Hash Function → 6

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

1,000,000 → Hash Function → 6

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

1,000,000 → Hash Function → 6

Dictionaries - Hash Table Mini-Lesson

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	1	2	3	4	5	6	7	8	9

1,000,000 → Hash Function → 6

- **Dictionaries** are built upon these **Hash Tables**, and the key's in our key/value pairs are stored in memory **IN** these hash tables at indexes which are **determined** by a **hash function**

Dictionaries - Hash Table Mini-Lesson

Dictionaries - Hash Table Mini-Lesson

What happens when we run two different dictionary keys into a hash function, and the computer tells us to store them at the same index location?

Dictionaries - Hash Table Mini-Lesson

What happens when we run two different dictionary keys into a hash function, and the computer tells us to store them at the same index location?

"Steven"

Dictionaries - Hash Table Mini-Lesson

What happens when we run two different dictionary keys into a hash function, and the computer tells us to store them at the same index location?

"Steven"

"Sean"

Dictionaries - Hash Table Mini-Lesson

What happens when we run two different dictionary keys into a hash function, and the computer tells us to store them at the same index location?

"Steven"

↓

Hash Function

"Sean"

↓

Hash Function

Dictionaries - Hash Table Mini-Lesson

What happens when we run two different dictionary keys into a hash function, and the computer tells us to store them at the same index location?

Dictionaries - Hash Table Mini-Lesson

What happens when we run two different dictionary keys into a hash function, and the computer tells us to store them at the same index location?

Dictionaries - Hash Table Mini-Lesson

What happens when we run two different dictionary keys into a hash function, and the computer tells us to store them at the same index location?

Hash Collision

Dictionaries - Hash Table Mini-Lesson

Hash Collision

Open Addressing

Dictionaries - Hash Table Mini-Lesson

Hash Collision

Open Addressing

Closed Addressing

Dictionaries - Hash Table Mini-Lesson

Open Addressing

Open Addressing

Put the key in some other index location separate from the one returned to us by the hash function

Open Addressing

Put the key in some other index location separate from the one returned to us by the hash function

8	9	10
“Selena”	nil	nil

Open Addressing

Put the key in some other index location separate from the one returned to us by the hash function

8	9	10
“Selena”	“Steven”	nil

Open Addressing

Put the key in some other index location separate from the one returned to us by the hash function

8	9	10
“Selena”	“Steven”	“Sean”

Dictionaries - Hash Table Mini-Lesson

Closed Addressing

Closed Addressing

Uses Linked Lists to chain together keys which result in the same has value

Closed Addressing

Uses Linked Lists to chain together keys which result in the same has value

8	9	10
“Selena”	“Steven”	nil
	“Sean”	

Dictionaries - Hash Table Mini-Lesson

Closed Addressing

Uses Linked Lists to chain together keys which result in the same has value

8	9	10
“Selena”	“Steven”	nil
	“Sean”	

Dictionaries - Hash Table Mini-Lesson

Closed Addressing

Uses Linked Lists to chain together keys which result in the same has value

8	9	10
“Selena”	“Steven”	nil
	“Sean”	

Closed Addressing

Uses Linked Lists to chain together keys which result in the same has value

8	9	10
“Selena”	“Steven”	nil
	“Sean”	

Dictionaries - Time Complexity Equations

Dictionaries - Time Complexity Equations

We generally measure a data structure based on it's worst-case scenario

Dictionaries - Time Complexity Equations

We generally measure a data structure based on it's worst-case scenario

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	0	0	0	0	0	0	0	0	0

Dictionaries - Time Complexity Equations

We generally measure a data structure based on it's worst-case scenario

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	0	0	0	0	0	0	0	0	0

Accessing

Dictionaries - Time Complexity Equations

We generally measure a data structure based on it's worst-case scenario

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	0	0	0	0	0	0	0	0	0

Accessing

Searching

Dictionaries - Time Complexity Equations

We generally measure a data structure based on it's worst-case scenario

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	0	0	0	0	0	0	0	0	0

Accessing

Searching

Inserting

Dictionaries - Time Complexity Equations

We generally measure a data structure based on it's worst-case scenario

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	0	0	0	0	0	0	0	0	0

Accessing **Searching** **Inserting** **Deleting**

Dictionaries - Time Complexity Equations

We generally measure a data structure based on it's worst-case scenario

Key	1	10	100	1,000	10K	100K	1 Million	10 Million	100 Million	1 Billion
Index	0	0	0	0	0	0	0	0	0	0

Accessing → O(n)

Searching → O(n)

Inserting → O(n)

Deleting → O(n)

Dictionaries - Time Complexity Equations

Accessing	Searching	Inserting	Deleting
O(n)	O(n)	O(n)	O(n)

Dictionaries - Time Complexity Equations

Dictionaries - Time Complexity Equations

- To access, search for, insert, or delete a key/value pair from our dictionary, all we need to do is run that key throughout hash function and it'll tell is what index in the hash table to go to in order to perform that operation

Dictionaries - Conclusion

- Dictionaries are a very useful data structure when it comes to Computer Science for a few reasons...
  -
  -
  -

Dictionaries - Conclusion

- Dictionaries are a very useful data structure when it comes to Computer Science for a few reasons...
  - The option for non-numerical indexes
  -
  -

Dictionaries - Conclusion

- Dictionaries are a very useful data structure when it comes to Computer Science for a few reasons...
  - The option for non-numerical indexes
  - The flexibility when it comes to making keys
  -

Dictionaries - Conclusion

- Dictionaries are a very useful data structure when it comes to Computer Science for a few reasons...
  - The option for non-numerical indexes
  - The flexibility when it comes to making keys
  - The speed which comes with the hash table implementation

An Introduction to Data Structures

Trees

Trees - Introduction

Trees - Introduction

Trees - Introduction

Trees - Introduction to Hierarchical Data

Trees - Introduction to Hierarchical Data

- Every Data Structure we've covered up until this point has been stored linearly...

Trees - Introduction to Hierarchical Data

- Every Data Structure we've covered up until this point has been stored linearly...

Trees - Introduction to Hierarchical Data

- Every Data Structure we've covered up until this point has been stored linearly...

Trees - Introduction to Hierarchical Data

- Every **Data Structure** we've covered up until this point has been stored **linearly...**

Trees - Introduction to Hierarchical Data

- Every **Data Structure** we've covered up until this point has been stored **linearly...**

Trees - Introduction to Hierarchical Data

- Every **Data Structure** we've covered up until this point has been stored **linearly...**

Trees - Introduction to Hierarchical Data

- Every Data Structure we've covered up until this point has been stored linearly...

Trees - Introduction to Hierarchical Data

- Trees store data hierarchically as opposed to linearly

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

Family Trees

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

Family Trees

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

Family Trees

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

Family Trees

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

Family Trees

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

Family Trees

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

Family Trees

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

Family Trees

Trees - Introduction to Hierarchical Data

- Trees store data hierarchically as opposed to linearly

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

File Structure

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

File Structure

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

File Structure

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

File Structure

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

File Structure

Trees - Introduction to Hierarchical Data

- Trees store data **hierarchically** as opposed to linearly

File Structure

Trees - Formal Background on the Tree

- A Tree...
  - An abstract Data Structure which contains a series of linked nodes connected together to form a hierarchical representation of information
- Like a LinkedList where each Node has the option of pointing towards multiple Nodes

Trees - Formal Background on the Tree

- A Tree...
  - An abstract Data Structure which contains a series of linked nodes connected together to form a hierarchical representation of information
- Like a LinkedList where each Node has the option of pointing towards multiple Nodes

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Tree

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Tree

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Tree

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Tree

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Tree

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Tree

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Tree

Trees - Formal Background on the Tree

- A Tree...
  - An abstract Data Structure which contains a series of linked nodes connected together to form a hierarchical representation of information
- Like a LinkedList where each Node has the option of pointing towards multiple Nodes

Tree

Edges

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Tree

Trees - Formal Background on the Tree

- A Tree...
  - An abstract Data Structure which contains a series of linked nodes connected together to form a hierarchical representation of information
- Like a LinkedList where each Node has the option of pointing towards multiple Nodes

Trees - Formal Background on the Tree

- A Tree...
  - An abstract Data Structure which contains a series of linked nodes connected together to form a hierarchical representation of information
- Like a LinkedList where each Node has the option of pointing towards multiple Nodes

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Trees - Formal Background on the Tree

- **A Tree...**
  - An **abstract** Data Structure which contains a series of linked nodes connected together to form a **hierarchical representation** of information
- Like a **LinkedList** where each Node has the option of pointing towards **multiple Nodes**

Trees - Tree Terminology and Visualization

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node -

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node -

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node -

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Vertice - A certain Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to another Node one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Trees - Tree Terminology and Visualization

Trees - Tree Terminology and Visualization

Height -

Depth -

Trees - Tree Terminology and Visualization

Property of the Tree

Height -

Depth -

Trees - Tree Terminology and Visualization

Property of the Tree

Height -

Property of a Node

Depth -

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth -

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth -

Trees - Tree Terminology and Visualization

Height = 3

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth -

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth -

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Vertice - A Node in a Tree

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Vertice - A Node in a Tree

Edge - A connection between Nodes

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Vertice - A Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Vertice - A Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to one level above itself

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Vertice - A Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Vertice - A Node in a Tree

Edge - A connection between Nodes

Root Node - Topmost Node of a Tree

Child Node - A certain Node which has an edge connecting it to one level above itself

Parent Node - Any Node which has 1 or more child Nodes

Leaf Node - A Node in a tree which does not have any child Nodes

Property of the Tree

Height - Number of edges on the longest possible path down towards a leaf

Property of a Node

Depth - Number of edges required to get from that particular node to the root Node

Trees - Tree Terminology and Visualization

Trees - Tree Terminology and Visualization

Trees - Tree Terminology and Visualization

Trees - Tree Terminology and Visualization

Trees - Tree Terminology and Visualization

Trees - Tree Terminology and Visualization

Trees - Tree Terminology and Visualization

1950

Trees - Tree Terminology and Visualization

1950

1960

Trees - Tree Terminology and Visualization

1950

1954

1960

Trees - Tree Terminology and Visualization

Trees - Tree Terminology and Visualization

Trees - Different types of Trees

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Rules

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Rules

Restrictions

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Binary Search Tree

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Binary Search Tree

AVL Tree

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Binary Search Tree

Red-Black Tree

AVL Tree

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Binary Search Tree

Red-Black Tree

AVL Tree

N-ary Tree

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

Trees - Different types of Trees

- Regular trees are great for storing hierarchical data, but their power can really be heightened when you start messing around with how the data is actually stored within them

An Introduction to Data Structures

Tries

An Introduction to Data Structures

Graphs

An Introduction to Data Structures

Heaps

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children
  - For any given **parent Node**, the child to the **left** has a value **less** than or equal to itself, and the child to the **right** has a value **greater** than or equal to itself

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children
  - For any given **parent Node**, the child to the **left** has a value **less** than or equal to itself, and the child to the **right** has a value **greater** than or equal to itself

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children
  - For any given **parent Node**, the child to the **left** has a value **less** than or equal to itself, and the child to the **right** has a value **greater** than or equal to itself

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children
  - For any given **parent Node**, the child to the **left** has a value **less** than or equal to itself, and the child to the **right** has a value **greater** than or equal to itself

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children
  - For any given **parent Node**, the child to the **left** has a value **less** than or equal to itself, and the child to the **right** has a value **greater** than or equal to itself

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children
  - For any given **parent Node**, the child to the **left** has a value **less** than or equal to itself, and the child to the **right** has a value **greater** than or equal to itself
  - No 2 Nodes can contain the **same value**

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children
  - For any given **parent Node**, the child to the **left** has a value **less** than or equal to itself, and the child to the **right** has a value **greater** than or equal to itself
  - No 2 Nodes can contain the **same value**

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children
  - For any given **parent Node**, the child to the **left** has a value **less** than or equal to itself, and the child to the **right** has a value **greater** than or equal to itself
  - No 2 Nodes can contain the **same value**

Trees - Different types of Trees

- A **Binary Search Tree** is a simple variation on the standard tree which has **three restrictions** on it to help organize the data
  - A Node can have at most 2 children
  - For any given **parent Node**, the child to the **left** has a value **less** than or equal to itself, and the child to the **right** has a value **greater** than or equal to itself
  - No 2 Nodes can contain the **same value**

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Go left if the value we're searching for is less than the current Node

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Go left if the value we're searching for is less than the current Node

Go right if the value we're searching for is greater than the current Node

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Go left if the value we're searching for is less than the current Node

Go right if the value we're searching for is greater than the current Node

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Go left if the value we're searching for is less than the current Node

Go right if the value we're searching for is greater than the current Node

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Go left if the value we're searching for is less than the current Node

Go right if the value we're searching for is greater than the current Node

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Go left if the value we're searching for is less than the current Node

Go right if the value we're searching for is greater than the current Node

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Go left if the value we're searching for is less than the current Node

Go right if the value we're searching for is greater than the current Node

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Go left if the value we're searching for is less than the current Node

Go right if the value we're searching for is greater than the current Node

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

Binary Search Tree

Trees - Different types of Trees

- The biggest advantage of Binary Search Trees is that we're able to search through them in Logarithmic Time

- Makes them extremely popular for storing large quantities of data that need to be easily searchable
  - Also translates to accessing, inserting, and deleting Nodes

Binary Search Tree

Trees - Uses for the Tree

Trees - Uses for the Tree

Hierarchical Data

Trees - Uses for the Tree

Hierarchical Data

Trees - Uses for the Tree

Hierarchical Data

Trees - Uses for the Tree

Hierarchical Data

Trees - Uses for the Tree

Trees - Uses for the Tree

An Introduction to Data Structures

Tries

Tries - Introduction

Tries - Introduction

Binary Search Tree

Tries - Introduction

Trees can become even more useful once you start setting restrictions on how and where data can be stored within them

Binary Search Tree

Tries - Introduction

Trees can become even more useful once you start setting restrictions on how and where data can be stored within them

Trie

Tries - Introduction

The Trie is often **overlooked** since it is only used in **specific situations**

Trie

Tries - Introduction

Tries - Introduction

What are those restrictions?

Tries - Introduction

What are those restrictions?

How can they help us?

Tries - Trie Basics

- A Trie is a tree-like data structure whose nodes store letters of an alphabet in the form of characters

Tries - Trie Basics

- A Trie is a tree-like data structure whose nodes store letters of an alphabet in the form of characters

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

Tries - Trie Basics

- A Trie is a tree-like data structure whose nodes store letters of an alphabet in the form of characters

Tries - Trie Basics

- A Trie is a tree-like data structure whose nodes store letters of an alphabet in the form of characters

- We can carefully construct this tree of characters in such a way which allows us to quickly retrieve words in the form of Strings by traversing down a path of the trie in a certain way

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

- We can **carefully construct** this tree of characters in such a way which allows us to quickly **retrieve words** in the form of Strings by traversing down a path of the trie in a certain way

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

- We can **carefully construct** this tree of characters in such a way which allows us to quickly **retrieve words** in the form of Strings by traversing down a path of the trie in a certain way

Deal

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

- We can **carefully construct** this tree of characters in such a way which allows us to quickly **retrieve words** in the form of Strings by traversing down a path of the trie in a certain way

Dear

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

- We can **carefully construct** this tree of characters in such a way which allows us to quickly **retrieve words** in the form of Strings by traversing down a path of the trie in a certain way

Do

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

- We can **carefully construct** this tree of characters in such a way which allows us to quickly **retrieve words** in the form of Strings by traversing down a path of the trie in a certain way

He

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

- We can **carefully construct** this tree of characters in such a way which allows us to quickly **retrieve words** in the form of Strings by traversing down a path of the trie in a certain way

Heat

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

- We can **carefully construct** this tree of characters in such a way which allows us to quickly **retrieve words** in the form of Strings by traversing down a path of the trie in a certain way

Hen

Tries - Trie Basics

- A **Trie** is a **tree-like** data structure whose nodes store **letters** of an alphabet in the form of **characters**

- We can **carefully construct** this tree of characters in such a way which allows us to quickly **retrieve words** in the form of Strings by traversing down a path of the trie in a certain way

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Root Node

With References

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

References that point towards all characters in the English alphabet which serve as the first two characters of a word in the English dictionary

Tries - Trie Visualization

Tries - Trie Visualization

Dad

Tries - Trie Visualization

Dad

Dao

Tries - Trie Visualization

Dad

Dao

Dab

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Trie Visualization

Tries - Flagging

Flagging - Marking the end of a word by having it also point towards a “flag” to let the computer know that the end of a word has occurred

Tries - Flagging

Flagging - Marking the end of a word by having it also point towards a “flag” to let the computer know that the end of a word has occurred

Tries - Flagging

Flagging - Marking the end of a word by having it also point towards a "flag" to let the computer know that the end of a word has occurred

Tries - Flagging

Flagging - Marking the end of a word by having it also point towards a "flag" to let the computer know that the end of a word has occurred

Tries - Flagging

Flagging - Marking the end of a word by having it also point towards a "flag" to let the computer know that the end of a word has occurred

Tries - Uses for Tries

Tries - Uses for Tries

Tries - Uses for Tries

Tries - Uses for Tries

- Big programs like IOS or google docs don't just store tries containing a few words or even all words starting with a certain letter. They're storing the entire English dictionary

Tries - Uses for Tries

- Big programs like IOS or google docs don't just store tries containing a few words or even all words starting with a certain letter. They're storing the entire English dictionary

Tries - Uses for Tries

- Big programs like IOS or google docs don't just store tries containing a few words or even all words starting with a certain letter. They're storing the entire English dictionary

Tries - Uses for Tries

Tries - Uses for Tries

S

Tries - Uses for Tries

N

S

Tries - Uses for Tries

Tries - Uses for Tries

N

S

A

O

Tries - Uses for Tries

Tries - Uses for Tries

Tries - Uses for Tries

S

Tries - Uses for Tries

Su

Tries - Uses for Tries

Sub

Tries - Uses for Tries

Subs

Tries - Uses for Tries

Tries - Uses for Tries

Tries - Uses for Tries

Subscribs

Tries - Uses for Tries

Subscribs

Tries - Uses for Tries

Tries - Uses for Tries

Chocolata

Tries - Uses for Tries

Chocolata

Tries - Conclusion

An Introduction to Data Structures

Heaps

Heaps - Introduction

- A **Binary Search Tree**
  -
  -
  -
  -

Binary Search Tree

Heaps - Introduction

- A **Binary Search Tree**
  - 2 Children Max
  -
  -
  -

Binary Search Tree

Heaps - Introduction

- A **Binary Search Tree**
  - 2 Children Max
  - Child to the **left** has a value **less** than or equal to itself
  -
  -

Binary Search Tree

Heaps - Introduction

- A **Binary Search Tree**
  - 2 Children Max
  - Child to the **left** has a value **less** than or equal to itself
  - Child to the **right** has a value **greater** than or equal to itself
  -

Binary Search Tree

Heaps - Introduction

- A **Binary Search Tree**
  - 2 Children Max
  - Child to the **left** has a value **less** than or equal to itself
  - Child to the **right** has a value **greater** than or equal to itself
  - No 2 Nodes can contain the **same value**

Binary Search Tree

Heaps - What is a Heap?

Heaps - Heap Basics

- **A Heap**
  - A special tree where all parent Nodes compare to their children Node's in some specific way by being more or less extreme
    - Either greater than or less than
    - Determines where the data is stored
    - Usually dependent on the parent Node's value

Heaps - Heap Basics

- **A Heap**
  - A special tree where all parent Nodes compare to their children Node's in some specific way by being more or less extreme
    - Either greater than or less than
    - Determines where the data is stored
    - Usually dependent on the parent Node's value

Heaps - Heap Basics

- **A Heap**
  - A special tree where all parent Nodes compare to their children Node's in some specific way by being more or less extreme
    - Either greater than or less than
    - Determines where the data is stored
    - Usually dependent on the parent Node's value

Heaps - Heap Basics

- **A Heap**
  - A special tree where all parent Nodes compare to their children Node's in some specific way by being more or less extreme
    - Either greater than or less than
    - Determines where the data is stored
    - Usually dependent on the parent Node's value

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Min-Heaps

- **Min-Heap**
  - The value at the **Root Node** of the tree must be the **minimum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Max-Heaps

- **Max-Heaps**
  - The value at the **Root Node** of the tree must be the **maximum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Max-Heaps

- **Max-Heaps**
  - The value at the **Root Node** of the tree must be the **maximum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Max-Heaps

- **Max-Heaps**
  - The value at the **Root Node** of the tree must be the **maximum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Max-Heaps

- **Max-Heaps**
  - The value at the **Root Node** of the tree must be the **maximum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Max-Heaps

- **Max-Heaps**
  - The value at the **Root Node** of the tree must be the **maximum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Max-Heaps

- **Max-Heaps**
  - The value at the **Root Node** of the tree must be the **maximum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Max-Heaps

- **Max-Heaps**
  - The value at the **Root Node** of the tree must be the **maximum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Max-Heaps

- **Max-Heaps**
  - The value at the **Root Node** of the tree must be the **maximum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Max-Heaps

- **Max-Heaps**
  - The value at the **Root Node** of the tree must be the **maximum** amongst all of its children
  - This fact must be the same **recursively** for all parent Node's contained within the Heap

Heaps - Building Heaps

Heaps - Building Heaps

Max Heap

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

- Recursively go up tree and swap Nodes if necessary
  - Either greater than or less than the node above it

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

- Recursively go up tree and swap Nodes if necessary
  - Either greater than or less than the node above it

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Index	Value
0	70
1	4
2	90
3	45
4	23
5	76
6	100

Heaps - Building Heaps

Max Heap

Heaps - Deleting From Heaps

Max Heap

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1.

2.

3.

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2.
3.

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2.
3.

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3.

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3.

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3.

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3. “Heapify” the Heap by comparing parent Node’s to their children and swapping if necessary

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3. “Heapify” the Heap by comparing parent Node’s to their children and swapping if necessary

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3. “Heapify” the Heap by comparing parent Node’s to their children and swapping if necessary

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3. “Heapify” the Heap by comparing parent Node’s to their children and swapping if necessary

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3. “Heapify” the Heap by comparing parent Node’s to their children and swapping if necessary

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3. “Heapify” the Heap by comparing parent Node’s to their children and swapping if necessary

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3. “Heapify” the Heap by comparing parent Node’s to their children and swapping if necessary

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3. “Heapify” the Heap by comparing parent Node’s to their children and swapping if necessary

Heaps - Deleting From Heaps

Max Heap

Deleting From the Root Node

1. Remove the root node from our Heap
2. Replace it with the Node furthest to the right
3. “Heapify” the Heap by comparing parent Node’s to their children and swapping if necessary

Heaps - Heap Implementations

- Heaps are most commonly used in the implementation of HeapSort
  - A sorting algorithm which takes in a list of elements, builds them into a min or max heap, and the removes the root Node continuously to make a sorted list

Heaps - Heap Implementations

- Heaps are most commonly used in the implementation of **HeapSort**
  - A **sorting algorithm** which takes in a list of elements, builds them into a min or max heap, and the **removes** the root Node **continuously** to make a sorted list

Index	Value
0	65
1	23
2	57
3	14
4	89
5	73
6	2

Heaps - Heap Implementations

- Heaps are most commonly used in the implementation of **HeapSort**
  - A **sorting algorithm** which takes in a list of elements, builds them into a min or max heap, and the **removes** the root Node **continuously** to make a sorted list

Index	Value
0	65
1	23
2	57
3	14
4	89
5	73
6	2

Heaps - Heap Implementations

- Heaps are most commonly used in the implementation of **HeapSort**
  - A **sorting algorithm** which takes in a list of elements, builds them into a min or max heap, and the **removes** the root Node **continuously** to make a sorted list

Index	Value
0	65
1	23
2	57
3	14
4	89
5	73
6	2

Index	Value
0	89
1	73
2	65
3	57
4	23
5	14
6	2

Heaps - Heap Implementations

- Heaps are most commonly used in the implementation of **HeapSort**
  - A **sorting algorithm** which takes in a list of elements, builds them into a min or max heap, and the **removes** the root Node **continuously** to make a sorted list

Index	Value
0	65
1	23
2	57
3	14
4	89
5	73
6	2

Index	Value
0	89
1	73
2	65
3	57
4	23
5	14
6	2

Heaps - Heap Implementations

- Heaps are most commonly used in the implementation of **HeapSort**
  - A **sorting algorithm** which takes in a list of elements, builds them into a min or max heap, and the **removes** the root Node **continuously** to make a sorted list

Index	Value
0	65
1	23
2	57
3	14
4	89
5	73
6	2

Index	Value
0	89
1	73
2	65
3	57
4	23
5	14
6	2

Heaps - Heap Implementations

- Heaps are most commonly used in the implementation of **HeapSort**
  - A **sorting algorithm** which takes in a list of elements, builds them into a min or max heap, and the **removes** the root Node **continuously** to make a sorted list

Index	Value
0	65
1	23
2	57
3	14
4	89
5	73
6	2

Index	Value
0	89
1	73
2	65
3	57
4	23
5	14
6	2

Heaps - Heap Implementations

- Heaps are most commonly used in the implementation of **HeapSort**
  - A **sorting algorithm** which takes in a list of elements, builds them into a min or max heap, and the **removes** the root Node **continuously** to make a sorted list

Index	Value
0	65
1	23
2	57
3	14
4	89
5	73
6	2

Index	Value
0	89
1	73
2	65
3	57
4	23
5	14
6	2

Heaps - Heap Implementations

- Heaps are most commonly used in the implementation of **HeapSort**
  - A **sorting algorithm** which takes in a list of elements, builds them into a min or max heap, and the **removes** the root Node **continuously** to make a sorted list

Heaps - Heap Implementations

- **Priority Queues**
  - An advanced data structure which your computer uses to **designate tasks** and assign computer power based on how **urgent** the matter is

Heaps - Heap Implementations

- **Priority Queues**
  - An advanced data structure which your computer uses to **designate tasks** and assign computer power based on how **urgent** the matter is

Heaps - Heap Implementations

- **Priority Queues**
  - An advanced data structure which your computer uses to **designate tasks** and assign computer power based on how **urgent** the matter is

Heaps - Heap Implementations

- **Priority Queues**
  - An advanced data structure which your computer uses to **designate tasks** and assign computer power based on how **urgent** the matter is

Heaps - Heap Implementations

- **Priority Queues**
  - An advanced data structure which your computer uses to **designate tasks** and assign computer power based on how **urgent** the matter is

Heaps - Heap Implementations

- **Priority Queues**
  - An advanced data structure which your computer uses to **designate tasks** and assign computer power based on how **urgent** the matter is

Heaps - Heap Implementations

- **Priority Queues**
  - An advanced data structure which your computer uses to **designate tasks** and assign computer power based on how **urgent** the matter is

Heaps - Review

- **Heaps**
  - A special tree in which each level contains Node's with values more extreme, either greater than or less than, the Node's on the level above it

An Introduction to Data Structures

Graphs

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs - Graph Visualization

Graphs

Graphs - Graph Visualization

Graphs

Pieces of Information

Graphs - Graph Visualization

Graphs

Pieces of Information

The Paths that run between them

Graphs - Graph Basics

- **A Graph**
  - A **nonlinear** Data Structure consisting of **Nodes** and **Edges**
    - Finite set of Nodes (**Vertices**)
    - Nodes are **connected** by the edges

Graphs - Graph Basics

- **A Graph**
  - A nonlinear Data Structure consisting of Nodes and Edges
    - Finite set of Nodes (Vertices)
    - Nodes are connected by the edges

Nodes

Graphs - Graph Basics

- **A Graph**
  - A **nonlinear** Data Structure consisting of **Nodes** and **Edges**
    - Finite set of Nodes (**Vertices**)
    - Nodes are **connected** by the edges

Nodes

Edges

Graphs - Graph Basics

- **A Graph**
  - A **nonlinear** Data Structure consisting of **Nodes** and **Edges**
    - Finite set of Nodes (**Vertices**)
    - Nodes are **connected** by the edges

Graphs - Graph Basics

- **A Graph**
  - A **nonlinear** Data Structure consisting of **Nodes** and **Edges**
    - Finite set of Nodes (**Vertices**)
    - Nodes are **connected** by the edges

Graphs - Graph Basics

- **A Graph**
  - A nonlinear Data Structure consisting of Nodes and Edges
    - Finite set of Nodes (Vertices)
    - Nodes are connected by the edges

Specific Starting Point

Graphs - Graph Basics

- **A Graph**
  - A nonlinear Data Structure consisting of Nodes and Edges
    - Finite set of Nodes (Vertices)
    - Nodes are connected by the edges

Specific Starting Point

Multiple Branches

Graphs - Graph Basics

- **A Graph**
  - A **nonlinear** Data Structure consisting of **Nodes** and **Edges**
    - Finite set of Nodes (**Vertices**)
    - Nodes are **connected** by the edges

Graphs - Graph Basics

- **A Graph**
  - A **nonlinear** Data Structure consisting of **Nodes** and **Edges**
    - Finite set of Nodes (**Vertices**)
    - Nodes are **connected** by the edges

Graphs - Graph Basics

- **A Graph**
  - A **nonlinear** Data Structure consisting of **Nodes** and **Edges**
    - Finite set of Nodes (**Vertices**)
    - Nodes are **connected** by the edges

Multiple Starting Points

Graphs - Graph Basics

- **A Graph**
  - A nonlinear Data Structure consisting of Nodes and Edges
    - Finite set of Nodes (Vertices)
    - Nodes are connected by the edges

Multiple Starting Points

Multiple Branches

Graphs - Graph Basics

Graphs - Graph Basics

Visually

Graphs - Graph Basics

Notationally

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Notationally

{ 1,2,3,4,5,6 }

{(6,4),(4,5),(4,3),(3,2),(5,2),(2,1),(5,1)}

Visually

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Graph Basics

Graphs - Directed vs. Undirected Graphs

- **An Undirected Graph**
  - A Graph in which the **direction** you traverse the Nodes **ISN'T** important
    - Usually indicated by a **lack of arrows**

Graphs - Directed vs. Undirected Graphs

- **An Undirected Graph**
  - A Graph in which the direction you traverse the Nodes ISN'T important
    - Usually indicated by a lack of arrows

Graphs - Directed vs. Undirected Graphs

- **An Undirected Graph**
  - A Graph in which the **direction** you traverse the Nodes **ISN'T** important
    - Usually indicated by a **lack of arrows**

Graphs - Directed vs. Undirected Graphs

- **An Undirected Graph**
  - A Graph in which the **direction** you traverse the Nodes **ISN'T** important
    - Usually indicated by a **lack of arrows**

Graphs - Directed vs. Undirected Graphs

- **An Undirected Graph**
  - A Graph in which the direction you traverse the Nodes ISN'T important
    - Usually indicated by a lack of arrows

Graphs - Directed vs. Undirected Graphs

- **An Undirected Graph**
  - A Graph in which the direction you traverse the Nodes ISN'T important
    - Usually indicated by a lack of arrows

Graphs - Directed vs. Undirected Graphs

- **An Undirected Graph**
  - A Graph in which the direction you traverse the Nodes ISN'T important
    - Usually indicated by a lack of arrows

Graphs - Directed vs. Undirected Graphs

- **An Directed Graph**
  - A Graph in which the **direction** you traverse the Nodes **IS** important
    - Usually indicated by a **arrows representing direction**

Graphs - Directed vs. Undirected Graphs

- **An Directed Graph**
  - A Graph in which the **direction** you traverse the Nodes **IS** important
    - Usually indicated by a **arrows representing direction**

Graphs - Directed vs. Undirected Graphs

- **An Directed Graph**
  - A Graph in which the **direction** you traverse the Nodes **IS** important
    - Usually indicated by a **arrows representing direction**

Graphs - Directed vs. Undirected Graphs

- **An Directed Graph**
  - A Graph in which the **direction** you traverse the Nodes **IS** important
    - Usually indicated by a **arrows representing direction**

Graphs - Directed vs. Undirected Graphs

- **An Directed Graph**
  - A Graph in which the **direction** you traverse the Nodes **IS** important
    - Usually indicated by a **arrows representing direction**

Graphs - Directed vs. Undirected Graphs

- **An Directed Graph**
  - A Graph in which the **direction** you traverse the Nodes **IS** important
    - Usually indicated by a **arrows representing direction**

Graphs - Directed vs. Undirected Graphs

- **An Directed Graph**
  - A Graph in which the **direction** you traverse the Nodes **IS** important
    - Usually indicated by a **arrows representing direction**

Graphs - Directed vs. Undirected Graphs

- **An Directed Graph**
  - A Graph in which the **direction** you traverse the Nodes **IS** important
    - Usually indicated by a **arrows representing direction**

Graphs - Directed vs. Undirected Graphs

Graphs - Directed vs. Undirected Graphs

Undirected

Graphs - Directed vs. Undirected Graphs

Undirected

Directed

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

All Undirected Graphs are Cyclical

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

All Undirected Graphs are Cyclical

Graphs - Cyclic vs. Acyclic Graphs

- **A Cyclic Graph**
  - One which contains a **path** from at least one Node back to itself

All Undirected Graphs are Cyclical

Graphs - Cyclic vs. Acyclic Graphs

- **An Acyclic Graph**
  - One which contains **no path** from any one Node which leads back in on **itself**

Graphs - Cyclic vs. Acyclic Graphs

- **An Acyclic Graph**
  - One which contains **no path** from any one Node which leads back in on **itself**

There are **NO** paths which lead back to itself

Graphs - Weighted Graphs

- **Weighted Graphs**
  - Associating a **numerical value** with each edge (**Cost**)
  - Each weight represents some **property** of the information you're trying to convey

Graphs - Weighted Graphs

- **Weighted Graphs**
  - Associating a numerical value with each edge (Cost)
  - Each weight represents some property of the information you're trying to convey

Graphs - Weighted Graphs

- **Weighted Graphs**
  - Associating a numerical value with each edge (Cost)
  - Each weight represents some property of the information you're trying to convey

A Good Weight

Graphs - Weighted Graphs

- **Weighted Graphs**
  - Associating a numerical value with each edge (Cost)
  - Each weight represents some property of the information you're trying to convey

A Good Weight

↓

Distance between Nodes

Graphs - Types of Graphs

Graphs - Types of Graphs

Directed

Undirected

Graphs - Types of Graphs

Directed

Cyclic

Undirected

Acyclic

Graphs - Types of Graphs

Directed

Cyclic

Weighted Edges

Undirected

Acyclic

Unweighted Edges

Graphs - Types of Graphs

Directed

Cyclic

Weighted Edges

Undirected

Acyclic

Unweighted Edges

Graphs - Types of Graphs

Directed

Cyclic

Weighted Edges

Undirected

Acyclic

Unweighted Edges

Graphs - Types of Graphs

Directed

Cyclic

Weighted Edges

Undirected

Acyclic

Unweighted Edges

Graphs - Types of Graphs

Directed

Undirected

Cyclic

Acyclic

Weighted Edges

Unweighted Edges

Graphs - Types of Graphs

- **Undirected Cyclical Heaps with weighted Edges** can be used through **Dijkstra's shortest path algorithm**
  - Compiles a **list** of the shortest possible paths from that source vertex to all other Nodes within the Graph

Graphs - Types of Graphs

- **Undirected Cyclical Heaps with weighted Edges** can be used through **Dijkstra's shortest path algorithm**
  - Compiles a **list** of the shortest possible paths from that source vertex to all other Nodes within the Graph

Graphs - Types of Graphs

- **Undirected Cyclical Heaps with weighted Edges** can be used through **Dijkstra's shortest path algorithm**
  - Compiles a **list** of the shortest possible paths from that source vertex to all other Nodes within the Graph

Graphs - Types of Graphs

- **Undirected Cyclical Heaps with weighted Edges** can be used through **Dijkstra's shortest path algorithm**
  - Compiles a **list** of the shortest possible paths from that source vertex to all other Nodes within the Graph

Graphs - Types of Graphs

- **Unweighted Cyclical Graphs (Undirected and Directed)** are used in the **follower system** of a majority of social media websites
  - Facebook, Snapchat, Instagram, Twitter, etc.

Graphs - Types of Graphs

- **Unweighted Cyclical Graphs (Undirected and Directed)** are used in the **follower system** of a majority of social media websites
  - Facebook, Snapchat, Instagram, Twitter, etc.

Graphs - Types of Graphs

- **Unweighted Cyclical Graphs (Undirected and Directed)** are used in the **follower system** of a majority of social media websites
  - Facebook, Snapchat, Instagram, Twitter, etc.

Graphs - Types of Graphs

- **Unweighted Cyclical Graphs (Undirected and Directed)** are used in the **follower system** of a majority of social media websites
  - Facebook, Snapchat, Instagram, Twitter, etc.

Graphs - Types of Graphs

- **Unweighted Cyclical Graphs (Undirected and Directed)** are used in the **follower system** of a majority of social media websites
  - Facebook, Snapchat, Instagram, Twitter, etc.

Graphs - Review

- **Graph**
  - A **Data Structure** which is represented by a set of **Nodes** and **Edges**
    - Come together to form a **visualization** of data

Graphs - Review

- **Graph**
  - A **Data Structure** which is represented by a set of **Nodes** and **Edges**
    - Come together to form a **visualization** of data

Graphs - Review

- **Graph**
  - A **Data Structure** which is represented by a set of **Nodes** and **Edges**
    - Come together to form a **visualization** of data

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Conclusion

Arrays

An Introduction to Data Structures

NullPointerException

Introduction to Data Structures - Conclusion

Arrays

Hash-Tables

An Introduction to Data Structures

NullPointerException

Introduction to Data Structures - Conclusion

Arrays

Hash-Tables

Dictionaries

An Introduction to Data Structures

NullPointerException

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Conclusion

Introduction to Data Structures - Shameless Plug

NullPointer Exception
812 subscribers

Introduction to Data Structures - Shameless Plug

NullPointer Exception
812 subscribers

Introduction to Data Structures - Shameless Plug

NullPointer Exception
812 subscribers

*I'm the cute one on the channel

Introduction to Data Structures - Shameless Plug

NullPointer Exception
812 subscribers

Introduction to Data Structures - Shameless Plug

NullPointer Exception
812 subscribers

Check out the link in the description below