

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

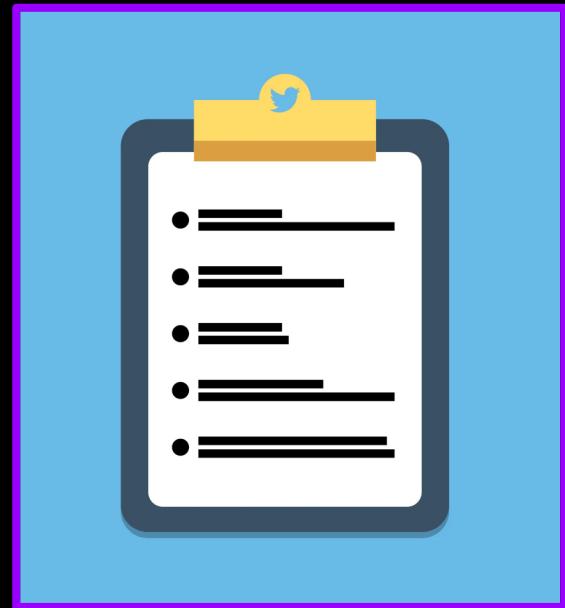
Time Complexity

Storage Methods

Check Description to skip ahead

The Array - Array Basics

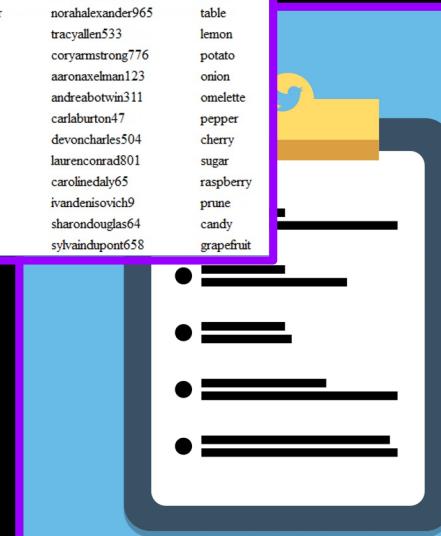
- An **array** is fundamentally a list of **similar** values
- Can be used to store **anything**
 - Usernames
 - High Scores
 - Prices
- Store values of the same **type**
 - Integer
 - String
 - Float



The Array - Array Basics

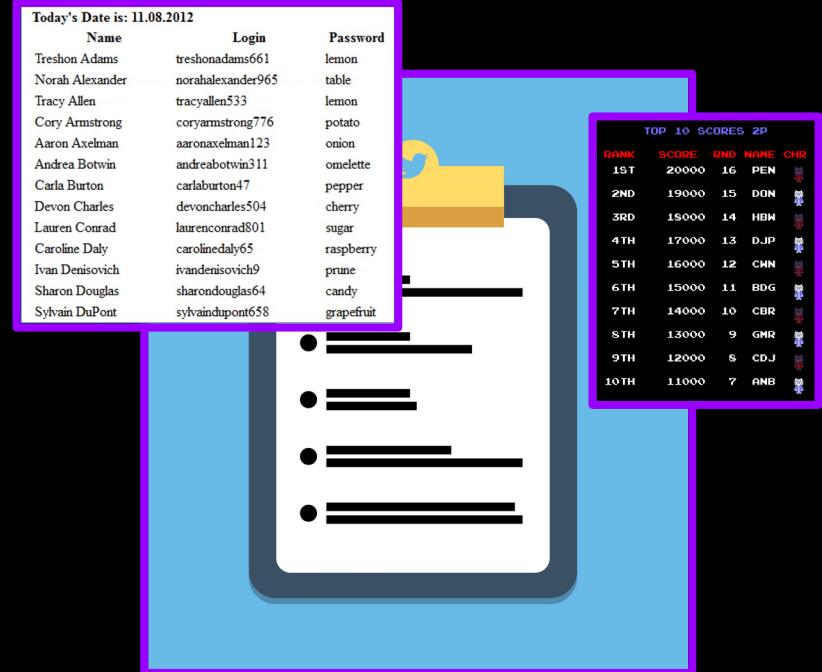
- An **array** is fundamentally a list of **similar** values
- Can be used to store **anything**
 - **Usernames**
 - **High Scores**
 - **Prices**
- Store values of the same **type**
 - **Integer**
 - **String**
 - **Float**

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	carolinadaly65	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

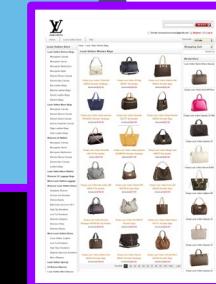
Name	Login	Password
Treshon Adams	treshonadams661	lemon
Norah Alexander	norahalexander965	table
Tracy Allen	tracyallen533	lemon
Cory Armstrong	coryarmstrong776	potato
Aaron Axelman	aaronaxelman123	onion
Andrea Botwin	andreibotwin311	omelette
Carla Burton	carlburton47	pepper
Devon Charles	devoncharles504	cherry
Lauren Conrad	laurenconrad801	sugar
Caroline Daly	carolinadaly65	raspberry
Ivan Denisovich	ivandenisovich9	prune
Sharon Douglas	sharondouglas64	candy
Sylvain DuPont	svlaindupont658	grapefruit

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

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	carlburton47	pepper
Devon Charles	devoncharles504	cherry
Lauren Conrad	laurenconrad801	sugar
Caroline Daly	carolinadaly65	raspberry
Ivan Denisovich	ivandenisovich9	prune
Sharon Douglas	sharondouglas64	candy
Sylvain DuPont	svlaintdupont658	grapefruit



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

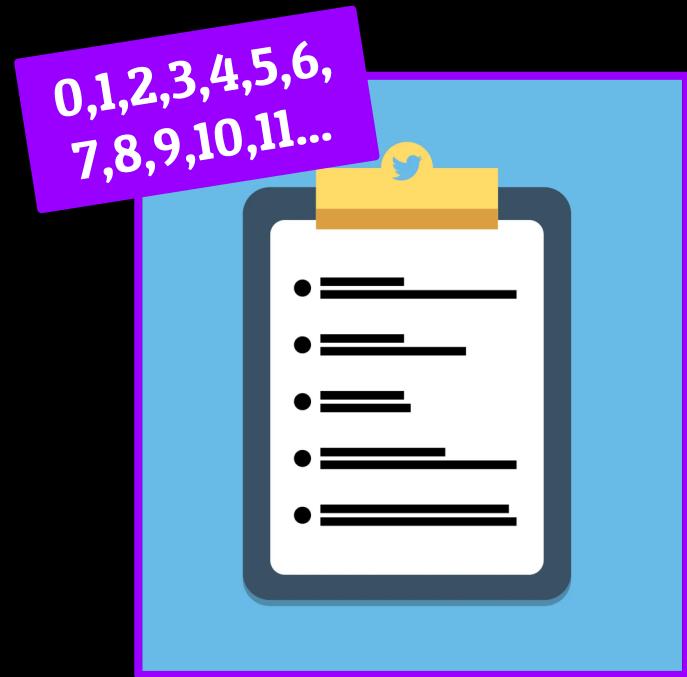
The Array - Array Basics

- 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



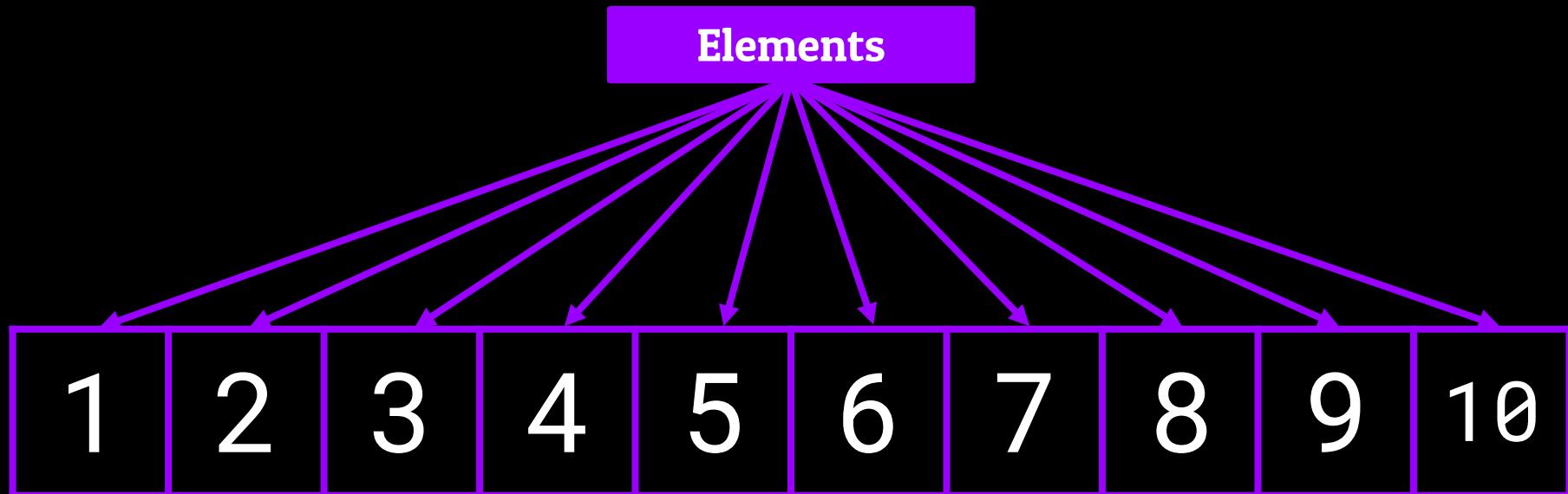
The Array - Array Basics

- 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
---	---	---	---	---	---	---	---	---	----

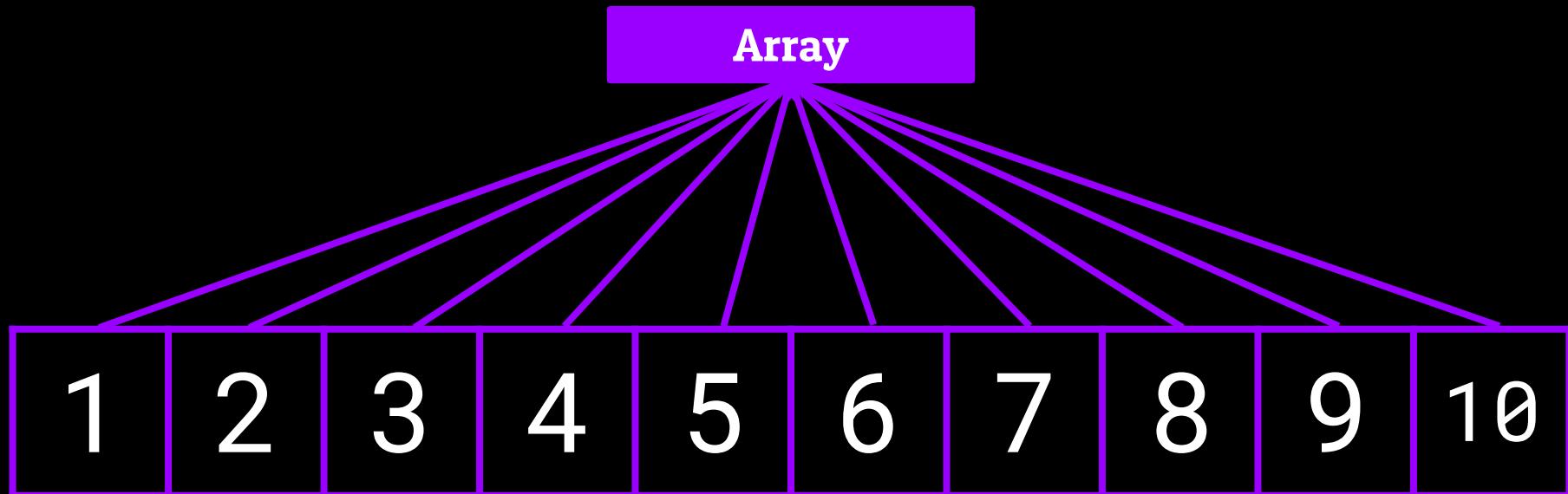
The Array - Array Basics

- 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”

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”

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”

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”

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



Grocery List

Eggs

Milk

Meat

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

Grocery
List

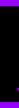
Eggs

Milk

Meat

Populate Later

Grocery
List



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

Grocery List

Eggs

Milk

Meat

Populate Later

Grocery List

Eggs

Milk

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

Grocery List

Eggs

Milk

Meat

Populate Later

Grocery List

Eggs

Milk

Meat

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

Grocery List

Eggs

Milk

Meat

Populate Later

Grocery List

Eggs

Milk

Meat

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

Salaries



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

~~prime numbers~~

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**

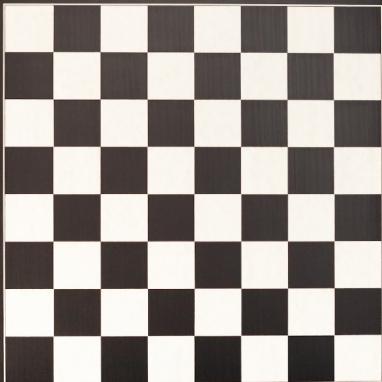
Index	0	1	2	3	4
0					
1					
2					
3					
4					

The Array - 2-Dimensional Arrays

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

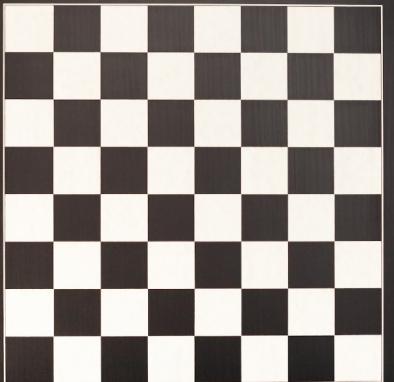
The Array - 2-Dimensional Arrays

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



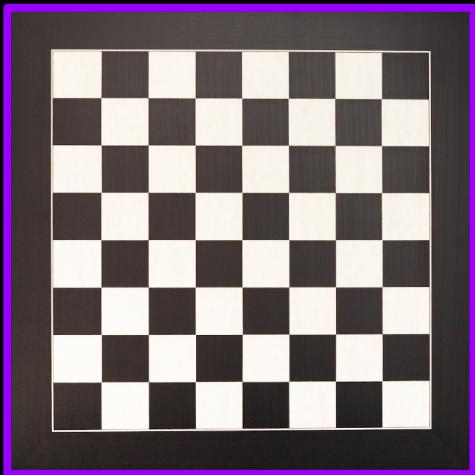
The Array - 2-Dimensional Arrays

- 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”

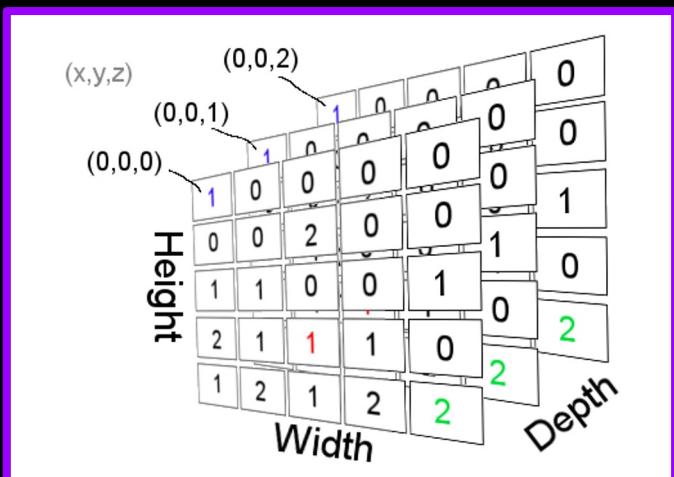
The Array - 2-Dimensional Arrays

- 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**

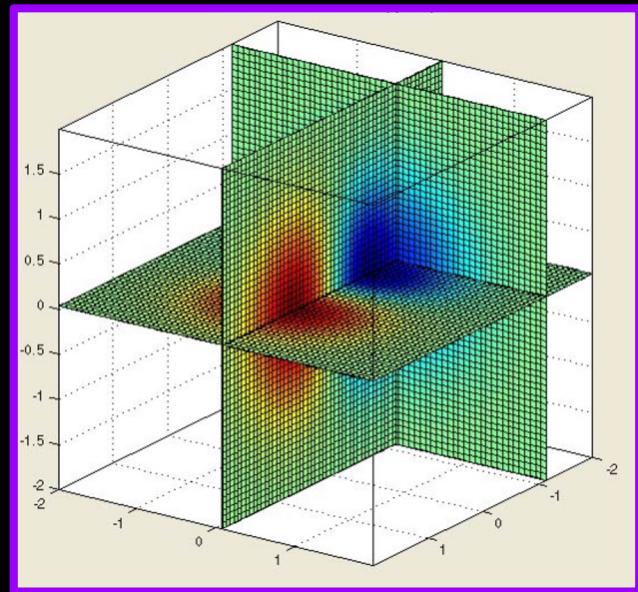
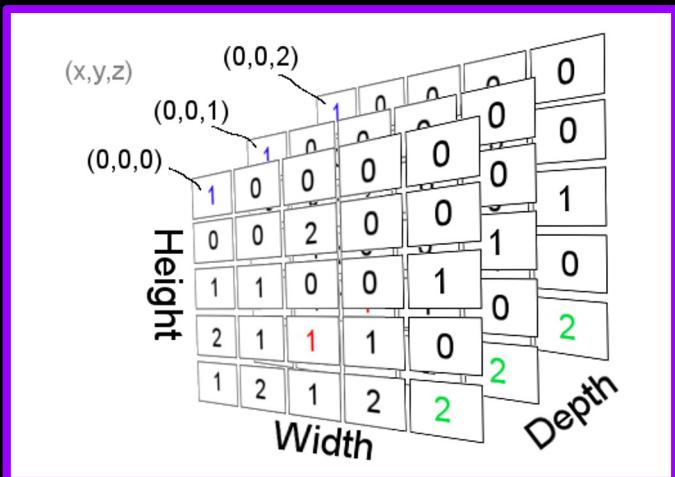
Index	0	1	2	3
0	“Steven”	“Alex”	“Dave”	“Jake”
1	“Adam”	“Lucas”	“Quinten”	“John”
2	“Sean”	“Marcus”	“Carl”	“Jackson”
3	“Peter”	“Cam”	“Anthony”	“Ethan”

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

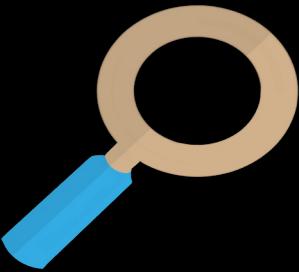


Accessing

The Array - Arrays as a Data Structure



Accessing

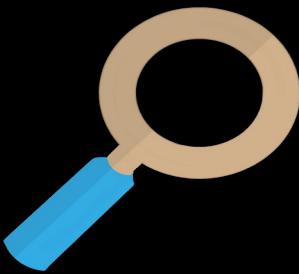


Searching

The Array - Arrays as a Data Structure



Accessing



Searching

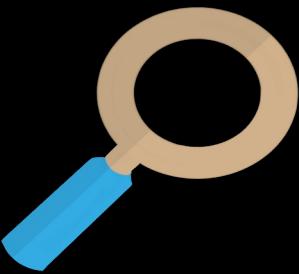


Inserting

The Array - Arrays as a Data Structure



Accessing



Searching

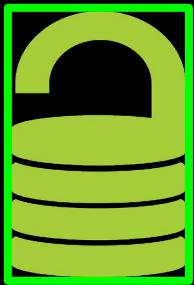


Inserting

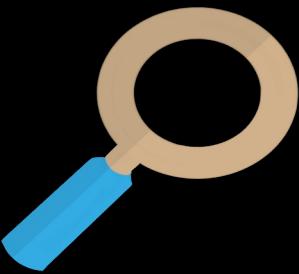


Deleting

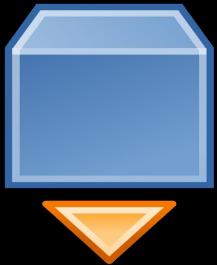
The Array - Arrays as a Data Structure



Accessing



Searching



Inserting

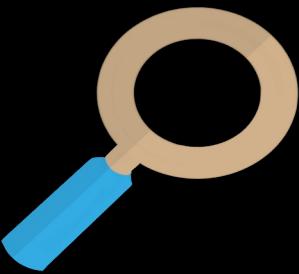


Deleting

The Array - Arrays as a Data Structure



Accessing



Searching



Inserting

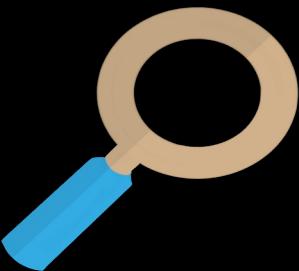


Deleting

The Array - Arrays as a Data Structure



Accessing



Searching



Inserting



Deleting

$O(1)$

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

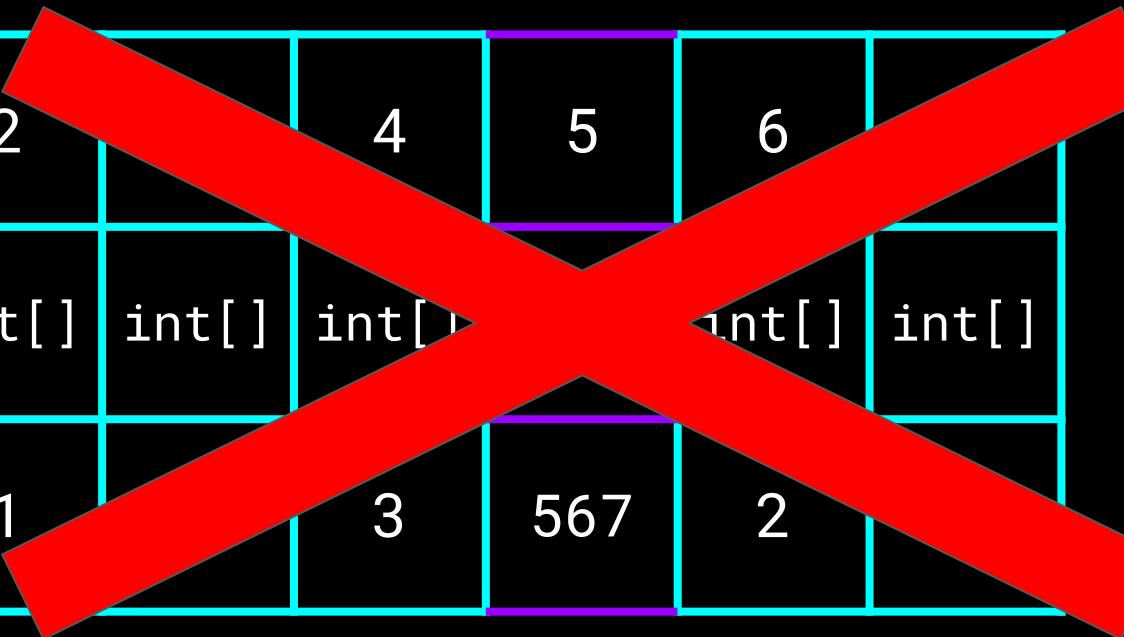
Memory

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

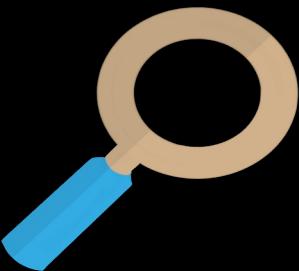
Spot in Memory	0	1	2	4	5	6
Memory Type	int	String	int[]	int[]	int[]	int[]
Stored Value	78	"Hey"	1	3	567	2



The Array - Arrays as a Data Structure



Accessing



Searching



Inserting



Deleting

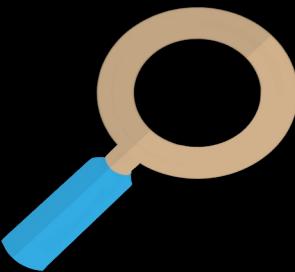
$O(1)$

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

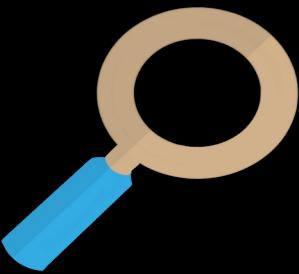
Search For: “Gary”

The Array - Arrays as a Data Structure



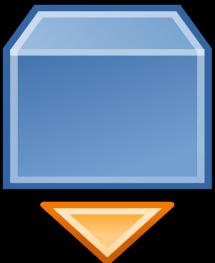
Accessing

$O(1)$



Searching

$O(n)$



Inserting



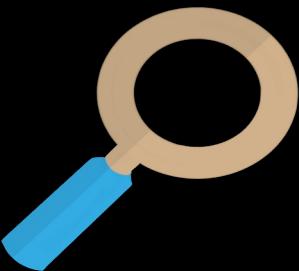
Deleting

The Array - Arrays as a Data Structure



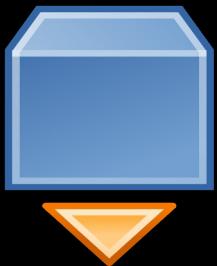
Accessing

$O(1)$



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**



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;

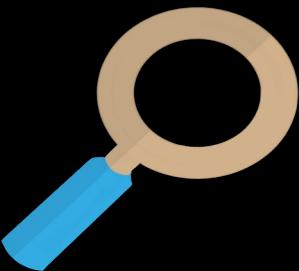
Index	0	1	2	3	4
Numbers	2	3	4	5	

The Array - Arrays as a Data Structure



Accessing

$O(1)$



Searching

$O(n)$



Inserting

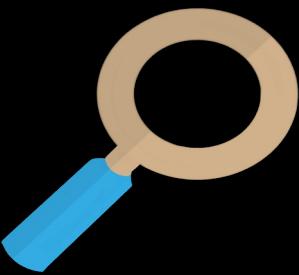
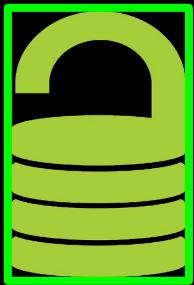
$O(n)$



Deleting

$O(n)$

The Array - Arrays as a Data Structure



Accessing

$O(1)$

Searching

$O(n)$

Inserting

$O(n)$

Deleting

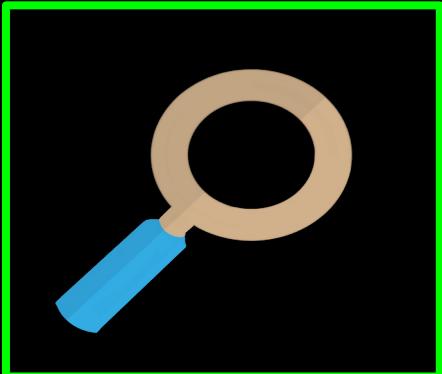
$O(n)$

The Array - Arrays as a Data Structure



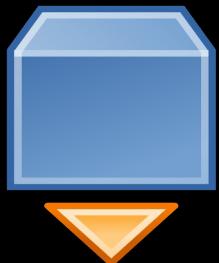
Accessing

$O(1)$



Searching

$O(n)$



Inserting

$O(n)$



Deleting

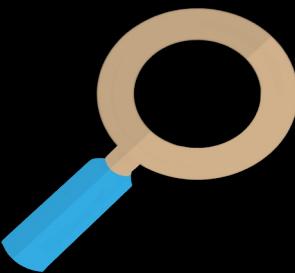
$O(n)$

The Array - Arrays as a Data Structure



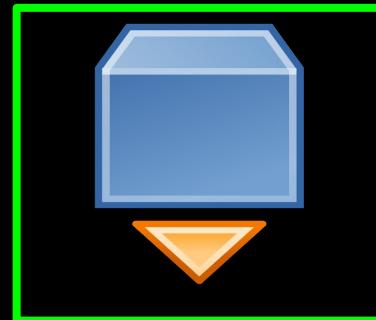
Accessing

$O(1)$



Searching

$O(n)$



Inserting

$O(n)$



Deleting

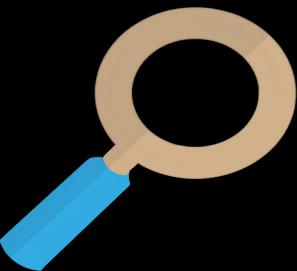
$O(n)$

The Array - Arrays as a Data Structure



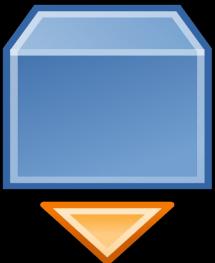
Accessing

$O(1)$



Searching

$O(n)$



Inserting

$O(n)$



Deleting

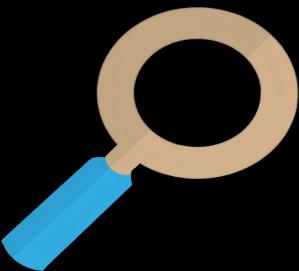
$O(n)$

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

Good for storing similar contiguous data

Cons

O(1) Accessing Power

Very basic. Easy to learn and master

The Array - Pros and Cons

Pros

Good for storing similar contiguous data

Cons

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

Good for storing similar contiguous data

O(1) Accessing Power

Very basic. Easy to learn and master

Cons

Size of the array cannot be changed once initialized

Inserting and Deleting are not efficient

The Array - Pros and Cons

Pros

Good for storing similar contiguous data

O(1) Accessing Power

Very basic. Easy to learn and master

Cons

Size of the array cannot be changed once initialized

Inserting and Deleting are not efficient

Can be wasting storage space

The Array - Pros and Cons

Pros

Good for storing similar contiguous data

O(1) Accessing Power

Very basic. Easy to learn and master

Cons

Size of the array cannot be changed once initialized

Inserting and Deleting are not efficient

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 **arrayList**, conceptually, 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
t

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

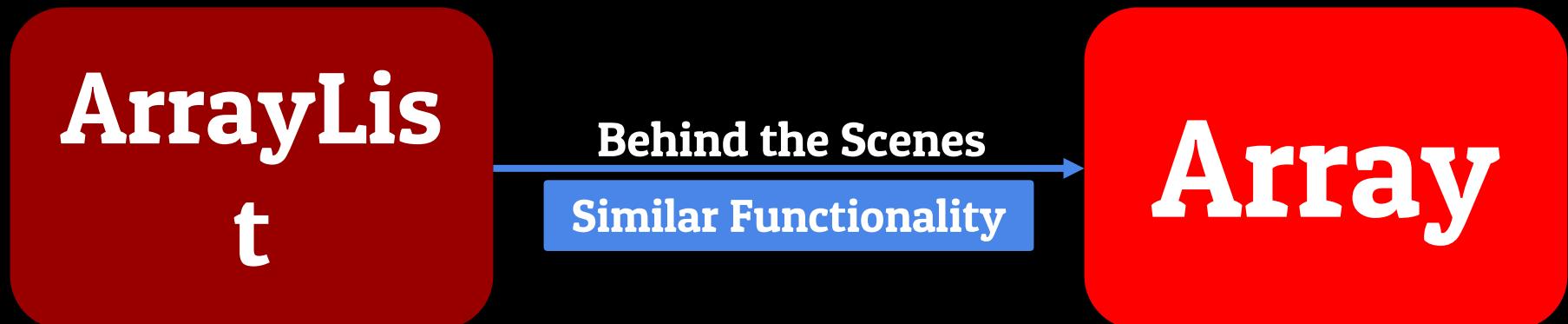
ArrayList
t

Behind the Scenes

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 arrayList are **frankensteined** together into a single data structure called **Lists**
 - Take some functionality from **both data structures**

The ArrayList - Initializing an ArrayList

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

Python Lists



Arrays

The ArrayList - Initializing an ArrayList

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

Python Lists

Arrays

ArrayLists

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

vList<Integer>(1, 2, 3);

ArrayList<Integer> arrList = new ArrayList<Integer>(1, 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>();
```

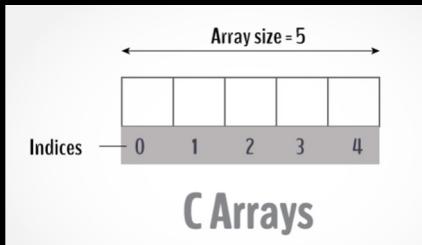
```
ArrayList arrayList = new ArrayList();
```

The ArrayList - ArrayList Functionality

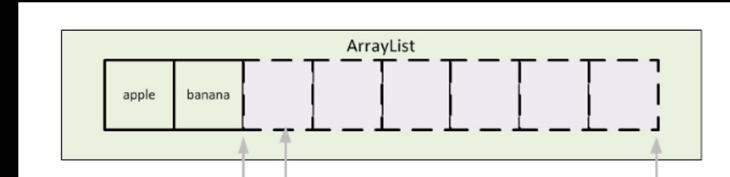
- An arrayList can be thought of as an **evolved array**
 - Beefier
 - More **Functionality**
 - More Powerful

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

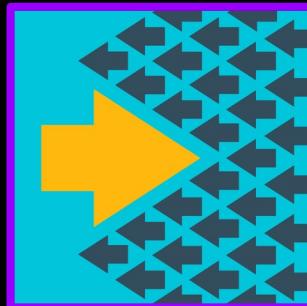
The ArrayList - ArrayList Functionality

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



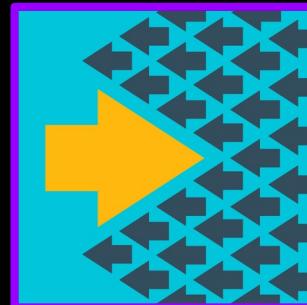
The ArrayList - ArrayList Functionality

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



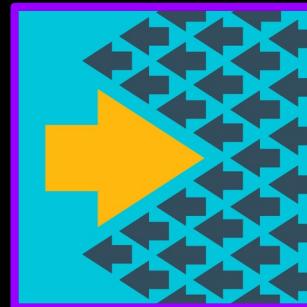
The ArrayList - ArrayList Functionality

- 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

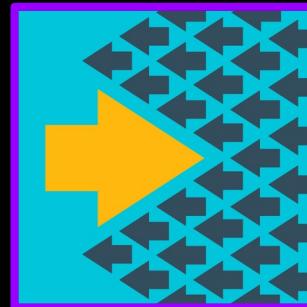


The ArrayList - ArrayList Functionality

- 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

```
og").val();
= 0; c < a.length; c++) {
    b += " " + a[c];
}
0; c < a.length; c++) {
    function(a) {
        b = " " + a[c];
        if (0 == a.length) {
            return " ";
        }
        var a = "#inp-stats-unique";
        a.html(liczenie().unique);
        a = "#use";
        a.val();
        a = "#User_logged";
        a = a.split("");
        for (var a = "", b = "", c = 0; c < a.length; c++) {
            a += use_array[a[c]];
            b += use_array[b[c]];
        }
        a = "#use_unique";
        a.function(count_use_unique(a)) {
            for (var a = "", b = "", c = 0; c < a.length; c++) {
                a += use_array[a[c]];
                b += use_array[b[c]];
            }
            a = "#inp_use";
            a.function(count_use_unique(a)) {
                for (var a = "", b = "", c = 0; c < a.length; c++) {
                    a += use_array[a[c]];
                    b += use_array[b[c]];
                }
                a = "#use";
                a.function(count_use_unique(a)) {
                    for (var a = "", b = "", c = 0; c < a.length; c++) {
                        a += use_array[a[c]];
                        b += use_array[b[c]];
                    }
                    a = "#User_logged";
                    a.function(count_use_unique(a)) {
                        for (var a = "", b = "", c = 0; c < a.length; c++) {
                            a += use_array[a[c]];
                            b += use_array[b[c]];
                        }
                    }
                }
            }
        }
    }
}
```

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

```
exampleArrayList.add(2);
```

Index	0	1	2	3
ExampleArrayList				

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

```
exampleArrayList.add(2);
```

Index	0	1	2	3
ExampleArrayList				

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

```
exampleArrayList.add(2);
```

Index	0	1	2	3
ExampleArrayList				

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

`exampleArrayList.add(2);`

2

Index	0	1	2	3
ExampleArrayList				

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

```
exampleArrayList.add(2);
```

2

Integer
Object (2)

Index	0	1	2	3
ExampleArrayList				

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

exampleArrayList.add(2);

2

Integer
Object (2)

Index	0	1	2	3
ExampleArrayList				

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

```
exampleArrayList.add(2);
```

Index	0	1	2	3
ExampleArrayList				

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

```
exampleArrayList.add(2);
```

Index	0	1	2	3
ExampleArrayList	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

```
exampleArrayList.add(5);
```

Index	0	1	2	3
ExampleArrayList	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

```
exampleArrayList.add(5);
```

Index	0	1	2	3
ExampleArrayList	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

```
exampleArrayList.add(5);
```

Index	0	1	2	3
ExampleArrayList	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

```
exampleArrayList.add(2);
```

Index	0	1	2	3
ExampleArrayList	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

```
exampleArrayList.add(2);
```

Index	0	1	2	3
ExampleArrayList	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)

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

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

Index	0	1	2	3
ExampleArrayList	2	5		

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

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

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

Index	0	1	2	3
ExampleArrayList		2	5	

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

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

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

Index	0	1	2	3
ExampleArrayList	1	2	5	

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

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

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

Index	0	1	2	3
ExampleArrayList	1	2	5	

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

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

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

Index	0	1	2	3
ExampleArrayList	1	2		5

The ArrayList - Add Method

- The add method has 2 different types...

Add(Object, Index)

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

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

Index	0	1	2	3
ExampleArrayList	1	2	3	5

The ArrayList - Add Method

- The add method has 2 different types...

Integer

Add(Object, Index)

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

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

Index	0	1	2	3
ExampleArrayList	1	2	3	5

The ArrayList - Add Method

- The add method has 2 different types...

Integer

Add(Object, Index)

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

Index

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

Index	0	1	2	3
ExampleArrayList	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)

Remove(Object)

- 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)

- 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...

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)

```
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)

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

True

- 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	

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

```
exampleArrayList.get(0);
```

Index	0	1	2	3
ExampleArrayList	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

exampleArrayList.get(0);

1

Index	0	1	2	3
ExampleArrayList	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

```
exampleArrayList.get(2);
```

Index	0	1	2	3
ExampleArrayList	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

```
exampleArrayList.get(2);
```

Index	0	1	2	3
ExampleArrayList	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

exampleArrayList.get(2);

3

Index	0	1	2	3
ExampleArrayList	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

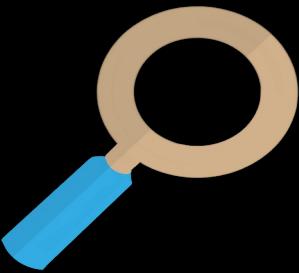


Accessing

The ArrayList - ArrayList as a Data Structure



Accessing

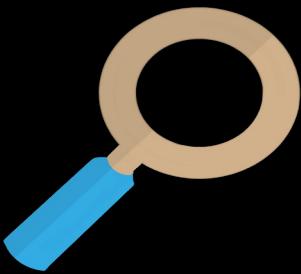


Searching

The ArrayList - ArrayList as a Data Structure



Accessing



Searching

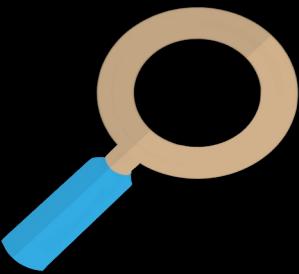


Inserting

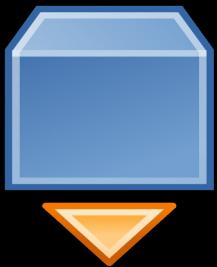
The ArrayList - ArrayList as a Data Structure



Accessing



Searching



Inserting

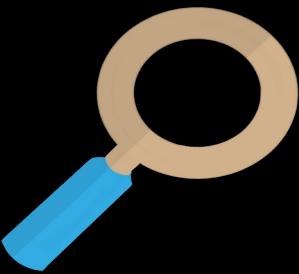


Deleting

The ArrayList - ArrayList as a Data Structure



Accessing



Searching



Inserting



Deleting

$O(1)$

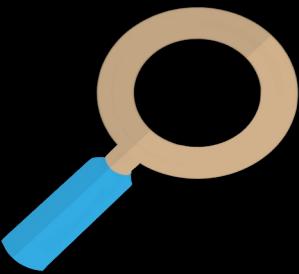
The ArrayList - ArrayList as a Data Structure



Accessing

Get
Method

O(1)



Searching



Inserting



Deleting

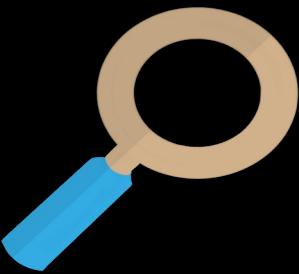
The ArrayList - ArrayList as a Data Structure



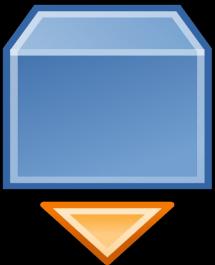
Accessing

Get
Method

Q(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
Stored Value	4	...	1	"hey"

The ArrayList - ArrayList as a Data Structure

Memory				
Index	0	1	2	3
Example	87	91	100	42
Spot in Memory	42	...	87	88
Stored Value	4	...	1	"hey"

The ArrayList - ArrayList as a Data Structure

Memory				
Index	0	1	2	3
Example	87	91	100	42
Spot in Memory	42	...	87	88
Stored Value	4	...	1	"hey"

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
Stored Value	4	...	1	"hey"

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

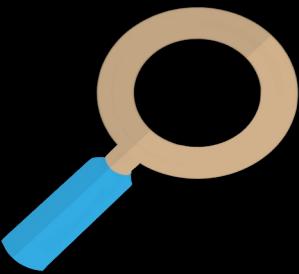
The ArrayList - ArrayList as a Data Structure



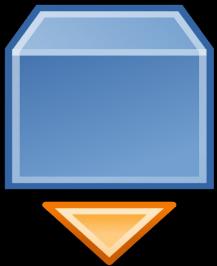
Accessing

Get
Method

O(1)



Searching



Inserting



Deleting

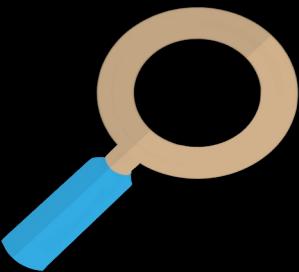
The ArrayList - ArrayList as a Data Structure



Accessing

Get
Method

O(1)



Searching

O(n)



Inserting



Deleting

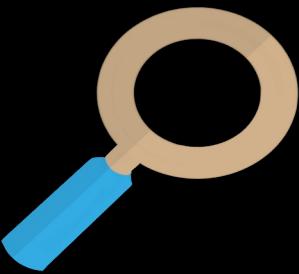
The ArrayList - ArrayList as a Data Structure



Accessing

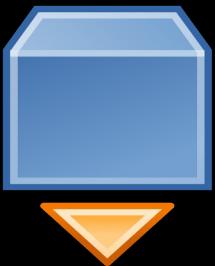
Get
Method

$O(1)$



Searching

$O(n)$



Inserting

Add(Object,
Index)
Add(Object
)

$O(n)$



Deleting

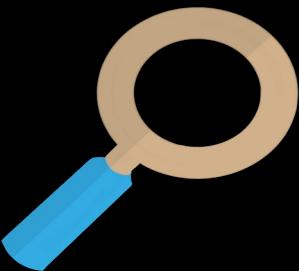
The ArrayList - ArrayList as a Data Structure



Accessing

Get
Method

$O(1)$



Searching

$O(n)$



Inserting

Add(Object,
Index) Add(Object
)

$O(n)$



Deleting

Remove
(Object) Remove
(Index)

$O(n)$

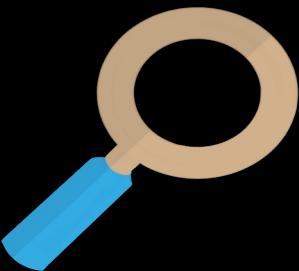
The ArrayList - ArrayList as a Data Structure



Accessing

Get
Method

$O(1)$



Searching

$O(n)$



Inserting

Add(Object,
Index) Add(Object
)

$O(n)$



Deleting

Remove
(Object) Remove
(Index)

$O(n)$

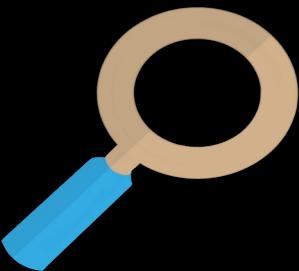
The ArrayList - ArrayList as a Data Structure



Accessing

Get
Method

$O(1)$



Searching

$O(n)$



Inserting

Add(Object,
Index) Add(Object
)

$O(n)$



Deleting

Remove
(Object) Remove
(Index)

$O(n)$

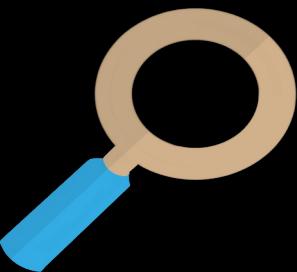
The ArrayList - ArrayList as a Data Structure



Accessing

Get
Method

$O(1)$



Searching

$O(n)$



Inserting

Add(Object,
Index) Add(Object
)

$O(n)$



Deleting

Remove
(Object) Remove
(Index)

$O(n)$

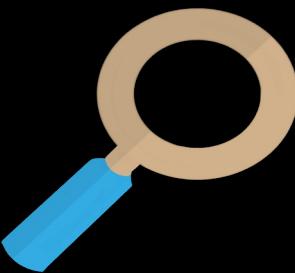
The ArrayList - ArrayList as a Data Structure



Accessing

Get
Method

$O(1)$



Searching

$O(n)$



Inserting

Add(Object,
Index) Add(Object
)

$O(n)$



Deleting

Remove
(Object) Remove
(Index)

$O(n)$

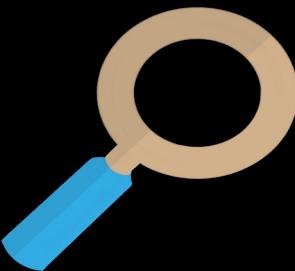
The ArrayList - ArrayList as a Data Structure



Accessing

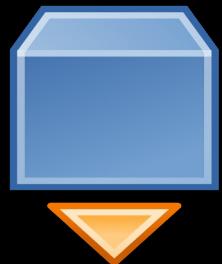
Get
Method

$O(1)$



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

$O(n)$

Add
(Index)

$O(n)$

Deleting

Move
(Object) 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

Fixed Size

ArrayLists

Dynamic Size

The ArrayList - Comparing and Contrasting with Arrays

Arrays

Fixed Size

Can store all data types

ArrayLists

Dynamic Size

The ArrayList - Comparing and Contrasting with Arrays

Arrays

Fixed Size

Can store all data types

ArrayLists

Dynamic Size

Can only store Objects

The ArrayList - Comparing and Contrasting with Arrays

Arrays

Fixed Size

Can store all data types

Methods need to be created

ArrayLists

Dynamic Size

Can only store Objects

The ArrayList - Comparing and Contrasting with Arrays

Arrays

Fixed Size

Can store all data types

Methods need to be created

ArrayLists

Dynamic Size

Can only store Objects

Methods are created for you

The ArrayList - Comparing and Contrasting with Arrays

Arrays

Fixed Size

Can store all data types

Methods need to be created

Doesn't require much memory use or upkeep

ArrayLists

Dynamic Size

Can only store Objects

Methods are created for you

The ArrayList - Comparing and Contrasting with Arrays

Arrays

Fixed Size

Can store all data types

Methods need to be created

Doesn't require much memory use or upkeep

ArrayLists

Dynamic Size

Can only store Objects

Methods are created for you

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

Smaller tasks, where you won't be interacting or changing the data that often

ArrayLists

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