Arrays
Data Structures

Thus far, all of our data has been stored in variables

one variable holds one piece of data

*Data structures* enable our programs to organize our data in more efficient, sensible ways

- group related pieces of data together

We'll see three types of data structures this semester

- variables (all semester)
- arrays (this week)
- classes (in a few weeks)
Exercise: Storing Multiple Pieces of Data

Suppose we wanted to store the names of everyone in this class
What information do we need to know?
How can we store that information in a program?
What if the user was providing the names through the console? Could we adapt to changes to how many people are in the class? (e.g., 27 vs 33?)
What Is An Array?

variables

- **age** (int): 11
- **firstName** (String): "James"
- **temp** (double): 32.5

array

- **firstNames** (String[]): "James", "Amy", "Beth", "Harold", "Remus", "Eliza"
Array Properties

Arrays allow us to store a collection of data values together.

All data stored in an array must be of the same data type:
  e.g., all Strings, all ints, all booleans

Must predetermine the size of our array:
  e.g., if we say our array will hold 27 names, we cannot modify it to store 33 names
  however, we can always store less data (e.g., 15 names)

We refer to data by its variable name and index (i.e., position) in the array:
  indexes are zero-based, just like with Strings
  the length of the String is not zero-based
Setting Up An Array

Three steps:

*declaring* the array sets up the variable name and data type.

only change is the addition of square brackets, e.g., `[]`
Setting Up An Array

Three steps:

*declaring* the array sets up the variable name and data type

only change is the addition of square brackets, e.g., \[

*instantiating* the array sets up the size (i.e., length)

<table>
<thead>
<tr>
<th>names (String)</th>
<th>null</th>
<th>null</th>
<th>null</th>
<th>null</th>
<th>null</th>
<th>null</th>
</tr>
</thead>
</table>

0 1 2 3 4 5
What Is Null?

The absence of data

Keyword in Java to indicate that there is nothing (i.e., no data) referred to by this variable/spot in the array

Always (always always) initialize/instantiate variables/arrays!

   except for primitives, these are set to null until initialization/instantiation
NullPointerException

Java throws an exception when your program attempts to use a null value:

- accessing an array that has not been instantiated
- accessing a spot in the array that has not been initialized
- will see this other places too (e.g., classes)

Exception in thread "main" java.lang.NullPointerException
at Example.main(Example:8)
Setting Up An Array

Three steps:

* **declaring** the array sets up the variable name and data type
  
  only change is the addition of square brackets, e.g., `[]`

* **instantiating** the array sets up the size (i.e., length)
Setting Up An Array

Three steps:

*declaring* the array sets up the variable name and data type

  only change is the addition of square brackets, e.g., []

*instantiating* the array sets up the size (i.e., length)

*initializing* the array assigns initial values to each spot in the array

names (String)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;James&quot;</td>
<td>&quot;Amy&quot;</td>
<td>&quot;Beth&quot;</td>
<td>&quot;Harold&quot;</td>
<td>&quot;Remus&quot;</td>
<td>&quot;Eliza&quot;</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Definition: Declaring & Instantiating An Array

declare an array

```
<dataType>[] <identifier>;    // both of these lines do the same thing
<dataType> <identifier>[];
```

instantiate an array

```
<identifier> = new <dataType>[<length>];
```

declare and instantiate an array

```
<dataType>[] <identifier> = new <dataType>[<length>];
```
Example: Declaring & Instantiating An Array

declare an array of type String called names

```java
String[] names;  // both of these lines do the same thing
String names[];
```

instantiate an array of type String with length 6

```java
names = new String[6];  // notice we do not use the square brackets here
```

declare and instantiate an array of type String called names with length 6

```java
String[] names = new String[6];
```
Example: Array Initialization

initialize an array of type String called names

```java
> names[0] = "James";
> names[1] = "Amy";
> names[2] = "Beth";
> names[3] = "Harold";
> names[4] = "Remus";
> names[5] = "Eliza";
```
Example: Declaring, Instantiating, and Initializing

declare an array of type String called names

```java
String[] names;  // both of these lines do the same thing
String names[];
```

instantiate and initialize an array with our name Strings

```java
names = {"James", "Amy", "Beth", "Harold", "Remus", "Eliza"};
```

declare, instantiate and initialize an array with our name Strings

```java
String[] names = {"James", "Amy", "Beth", "Harold", "Remus", "Eliza"};
```
Example: Array Access

access each value in the array and print it out

```java
System.out.println(names[0]);
System.out.println(names[1]);
System.out.println(names[2]);
System.out.println(names[3]);
System.out.println(names[4]);
System.out.println(names[5]);
```

`names (String [])`

<table>
<thead>
<tr>
<th></th>
<th>&quot;James&quot;</th>
<th>&quot;Amy&quot;</th>
<th>&quot;Beth&quot;</th>
<th>&quot;Harold&quot;</th>
<th>&quot;Remus&quot;</th>
<th>&quot;Eliza&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Definition: Array Length

Like Strings, can often be helpful to know the length of an array.

Unlike Strings, we use `.length` notice no parentheses!

access the length of an array.

```javascript
<identifier>.length;
names.length;
```
Example: Array Access

access each value in the array and print it out

```java
for (int i = 0; i < names.length; ++i) {
    System.out.println(names[i]);
}
```

<table>
<thead>
<tr>
<th>names (String[])</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;James&quot;</td>
</tr>
<tr>
<td>&quot;Amy&quot;</td>
</tr>
<tr>
<td>&quot;Beth&quot;</td>
</tr>
<tr>
<td>&quot;Harold&quot;</td>
</tr>
<tr>
<td>&quot;Remus&quot;</td>
</tr>
<tr>
<td>&quot;Eliza&quot;</td>
</tr>
</tbody>
</table>
ArrayIndexOutOfBoundsException

Java throws an exception when your program attempts to access a value beyond the length of the array.

similar to attempting to access a character index not available in a String

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
   at Example.main(Example:8)
Definition: String Methods

toCharArray: converts a String to an array of char values

```java
str.toCharArray();
```

equals: checks for equality between one String and another (case sensitive!)

```java
str.equals(str2);
```

`==`: checks to see if two String values point to the same memory location

```java
str == str2;
```
toCharArray

arguments: nothing

returns: a char array containing each character in the String, in order

```
<String>.toCharArray();

> String exampleStr = "Hi!";
> char[] arr = exampleStr.toCharArray();

memory

exampleStr (String)
"Hi!"

names (char[])
'H' 'i' '!'
equals

arguments: a String to compare to

returns: a boolean value; true if the two Strings are the same, false if not

```java
<String>.equals(<String>);

String exampleStr = "Hi!";
boolean same = exampleStr.equals("Hi");
```

memory

<table>
<thead>
<tr>
<th>exampleStr (String)</th>
<th>&quot;Hi!&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>same (boolean)</td>
<td>true</td>
</tr>
</tbody>
</table>
equals

arguments: a String to compare to

returns: a boolean value; true if the two Strings are the same, false if not

```
<String>.equals(<String>);

String exampleStr = "Hi!";

boolean same = exampleStr.equals("hi!");
```
arguments: two String values to compare

returns: a boolean value; true if the Strings are at the same memory location

```java
<String> == <String>;

String str1 = "Hi!", str2 = "Hi!";
boolean same = str1 == str2;
```

memory

```
str1 (String)          str2 (String)
"Hi!"                  "Hi!"

same (boolean)         false
```
== vs equals()

Primitive data types (boolean, char, int, double, ...)

- always use ==
- will check to see if the two are the same value
- .equals() does not exist for primitive data types

Class data types (String, ...)

- will almost always use .equals()
  - will check to see if the content of the two objects is the same
  - we can define what equality means!
- == will check if the memory location of the two objects is the same
Searching & Sorting

Data structures can contain multiple pieces of information in a single place. Often want to manipulate these:

- searching
- sorting
Searching An Array

Examine each index until we find what we are looking for
Searching Modifications

Know there are one vs many occurrences

one: can stop after it’s found
many: must continue until the end of the loop

Searching for first vs all occurrences

one: can stop after the first is found
many: must continue until the end of the loop
Sorting An Array

Numerous sorting algorithms available

many algorithms + their efficiency (i.e., complexity) will be discussed in 340

In this class

selection sort

insertion sort
Selection Sort

Considered one of the classic sorting algorithms

Very simple, but very inefficient

will do the job for this class

Basic premise:

scans through the array multiple times, looking for the next smallest element each time

moves the smallest element to the front of the array
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)
- initially, everything is unsorted

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

Repeat
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

8 3 2 5 9 7

smallestIndex = 1
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

\[
\text{smallestIndex} = 2
\]
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 2
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 2
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

8 3 2 5 9 7

smallestIndex = 2
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

8 3 2 5 9 7

smallestIndex = 2
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

\[
\text{smallestIndex} = 2
\]
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part).

Scan through the unsorted part for the smallest element.

Swap the smallest element with the leftmost unsorted value.

Length of sorted part increases by one, length of unsorted part decreases by one.

smallestIndex = 1

2 3 8 5 9 7
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 1
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 1
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 1
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

`smallestIndex = 1`
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 2
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 3
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 3
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 3
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 3
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 3
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 3
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 3
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 5
Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

\[ \text{smallestIndex} = 5 \]
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part).

Scan through the unsorted part for the smallest element.

Swap the smallest element with the leftmost unsorted value.

Length of sorted part increases by one, length of unsorted part decreases by one.

smallestIndex = 5
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 5
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 5
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one

smallestIndex = 5
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Selection Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Scan through the unsorted part for the smallest element

Swap the smallest element with the leftmost unsorted value

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Considered one of the classic sorting algorithms

Very simple, but very inefficient

- will do the job for this class

Basic premise:

- scans through the array multiple times, looking at the next unsorted element
- moves that unsorted element into a sorted place in the final list
**Insertion Sort**

Array is divided into two parts: sorted (left part) and unsorted (right part)

- Initially, first element is sorted, everything else is unsorted

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

Repeat
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

3 8 2 5 9 7
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one

2 3 8 5 9 7
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part).

Look at the leftmost unsorted value.

Move it down the sorted list until it is in the correct place.

Length of sorted part increases by one, length of unsorted part decreases by one.
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
Insertion Sort

Array is divided into two parts: sorted (left part) and unsorted (right part)

Look at the leftmost unsorted value

Move it down the sorted list until it is in the correct place

Length of sorted part increases by one, length of unsorted part decreases by one
chars have a strict ordering, just like numbers
comes from underlying numeric representations every char has

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Dec</th>
<th>Hex</th>
<th>Dec</th>
<th>Hex</th>
<th>Dec</th>
<th>Hex</th>
<th>Dec</th>
<th>Hex</th>
<th>Dec</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>16</td>
<td>10</td>
<td>DLE</td>
<td>32</td>
<td>20</td>
<td>48</td>
<td>30</td>
<td>64</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>SOH</td>
<td>17</td>
<td>11</td>
<td>DC1</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>49</td>
<td>31</td>
<td>65</td>
</tr>
<tr>
<td>2</td>
<td>02</td>
<td>STX</td>
<td>18</td>
<td>12</td>
<td>DC2</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>50</td>
<td>32</td>
<td>66</td>
</tr>
<tr>
<td>3</td>
<td>03</td>
<td>ETX</td>
<td>19</td>
<td>13</td>
<td>DC3</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>51</td>
<td>33</td>
<td>67</td>
</tr>
<tr>
<td>4</td>
<td>04</td>
<td>EOT</td>
<td>20</td>
<td>14</td>
<td>DC4</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>52</td>
<td>34</td>
<td>68</td>
</tr>
<tr>
<td>5</td>
<td>05</td>
<td>ENQ</td>
<td>21</td>
<td>15</td>
<td>NAK</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>53</td>
<td>35</td>
<td>69</td>
</tr>
<tr>
<td>6</td>
<td>06</td>
<td>ACK</td>
<td>22</td>
<td>16</td>
<td>SYN</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>54</td>
<td>36</td>
<td>70</td>
</tr>
<tr>
<td>7</td>
<td>07</td>
<td>BEL</td>
<td>23</td>
<td>17</td>
<td>ETB</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>55</td>
<td>37</td>
<td>71</td>
</tr>
<tr>
<td>8</td>
<td>08</td>
<td>BS</td>
<td>24</td>
<td>18</td>
<td>CAN</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>56</td>
<td>38</td>
<td>72</td>
</tr>
<tr>
<td>9</td>
<td>09</td>
<td>HT</td>
<td>25</td>
<td>19</td>
<td>EM</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>57</td>
<td>39</td>
<td>73</td>
</tr>
<tr>
<td>10</td>
<td>0A</td>
<td>LF</td>
<td>26</td>
<td>1A</td>
<td>SUB</td>
<td>42</td>
<td>2A</td>
<td>*</td>
<td>58</td>
<td>3A</td>
<td>:</td>
</tr>
<tr>
<td>11</td>
<td>0B</td>
<td>VT</td>
<td>27</td>
<td>1B</td>
<td>ESC</td>
<td>43</td>
<td>2B</td>
<td>;</td>
<td>59</td>
<td>3B</td>
<td>:</td>
</tr>
<tr>
<td>12</td>
<td>0C</td>
<td>FF</td>
<td>28</td>
<td>1C</td>
<td>FS</td>
<td>44</td>
<td>2C</td>
<td>`</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
</tr>
<tr>
<td>13</td>
<td>0D</td>
<td>CR</td>
<td>29</td>
<td>1D</td>
<td>GS</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>61</td>
<td>3D</td>
<td>=</td>
</tr>
<tr>
<td>14</td>
<td>0E</td>
<td>SO</td>
<td>30</td>
<td>1E</td>
<td>RS</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
</tr>
<tr>
<td>15</td>
<td>0F</td>
<td>SI</td>
<td>31</td>
<td>1F</td>
<td>US</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>63</td>
<td>3F</td>
<td>?</td>
</tr>
</tbody>
</table>