

# CSCI 132:

# Basic Data Structures and Algorithms

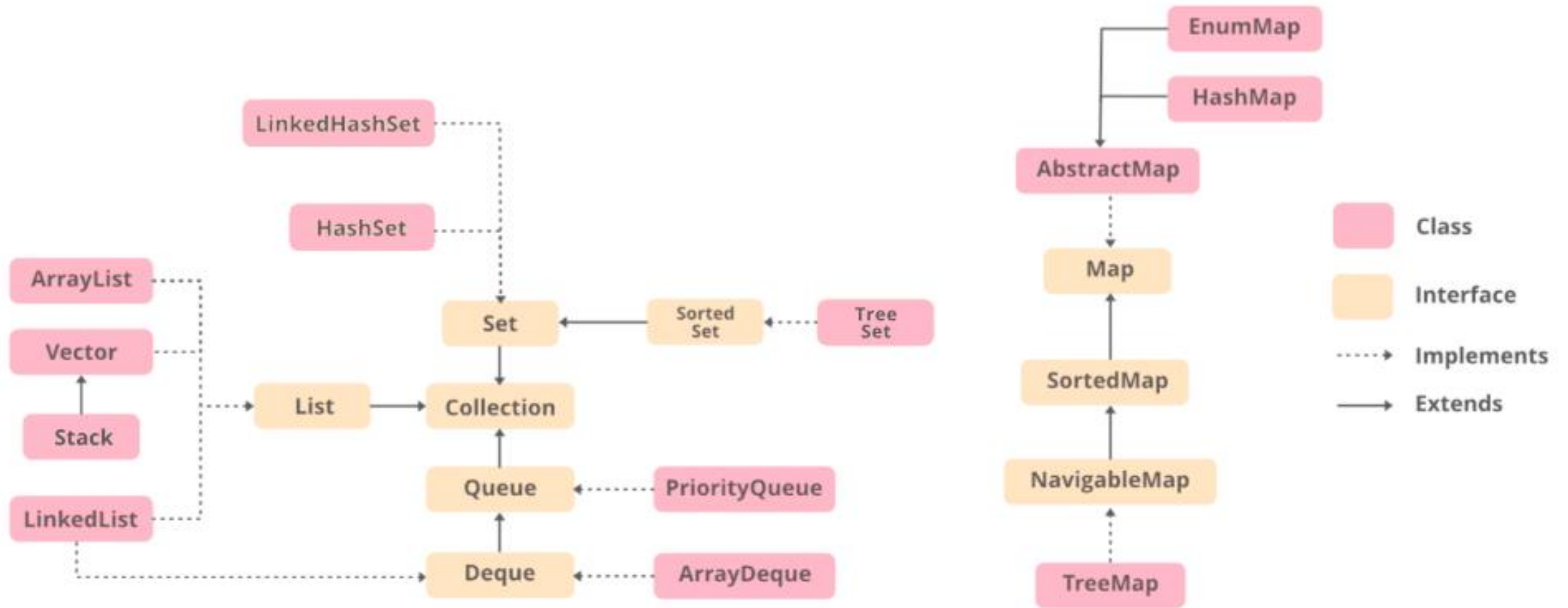
More Java Constructs

Reese Pearsall  
Spring 2024

# Announcements

- Lab 13- Course evaluation (easy)
- Program 5 due **Sunday (5/5)**
- No lecture on Wednesday (Program 5 help session)
- Take some time this week to double check your grades
- Final Exam  
**Wednesday May 8<sup>th</sup>**  
**2:00 PM – 3:50 PM**
- Rubber Duck Extra credit screenshot due by Friday
- Fill out the course evaluation





Instead of writing many `if/else` statements, you can use the `switch` statement

The `switch` statement selects one of many code blocks to be executed

```
int day = 4;
switch (day) {
    case 1:
        System.out.println("Monday");
        break;
    case 2:
        System.out.println("Tuesday");
        break;
    case 3:
        System.out.println("Wednesday");
        break;
    default:
        System.out.println("???");
}
```

These can be efficient when working with many possible conditions. They serve the same purpose as `if` statements, but are *slightly* more efficient

Java has a built-in `sort` method for Arrays

*What sorting algorithm does it use?*

<https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>

# Java has a built-in `sort` method for Arrays

*What sorting algorithm does it use?*

## Method Detail

### `sort`

```
public static void sort(int[] a)
```

Sorts the specified array into ascending numerical order.

Implementation note: The sorting algorithm is a **Dual-Pivot Quicksort** by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers  $O(n \log(n))$  performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Parameters:

`a` - the array to be sorted

<https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html>

# Java has a built-in `sort` method for Arrays

*What sorting algorithm does it use?*

## Method Detail

sort

```
public static void sort(int[] a)
```

Sorts the specified array into ascending numerical order.

Implementation note: The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers  $O(n \log(n))$  performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Parameters:

a - the array to be sorted

Python's `.sort()` function uses a hybrid of merge sort and insertion sort, called **Timsort**

Timsort

11 languages

Article Talk

Read Edit View history Tools

From Wikipedia, the free encyclopedia

**Timsort** is a [hybrid](#), [stable sorting algorithm](#), derived from [merge sort](#) and [insertion sort](#), designed to perform well on many kinds of real-world data. It was implemented by [Tim Peters](#) in 2002 for use in the [Python programming language](#). The algorithm finds subsequences of the data that are already ordered (runs) and uses them to sort the remainder more efficiently. This is done by merging runs until certain criteria are fulfilled. Timsort has been Python's standard sorting algorithm since version 2.3. It is also used to sort arrays of non-primitive type in [Java SE 7](#),<sup>[4]</sup> on the [Android platform](#),<sup>[5]</sup> in [GNU Octave](#),<sup>[6]</sup> on [V8](#),<sup>[7]</sup> [Swift](#),<sup>[8]</sup> and [Rust](#).<sup>[9]</sup>

It uses techniques from Peter McIlroy's 1993 paper "Optimistic Sorting and Information Theoretic Complexity".<sup>[10]</sup>

Timsort	
Class	Sorting algorithm
Data structure	Array
Worst-case performance	$O(n \log n)$ <sup>[1][2]</sup>
Best-case performance	$O(n)$ <sup>[3]</sup>
Average performance	$O(n \log n)$
Worst-case space complexity	$O(n)$

*Let's go back to when we were writing our own Linked List and Node class*

For example, this Linked List could only hold Strings

```
public class Node {  
  
    private String name;  
    private Node next;  
  
    public Node(String c) {  
        this.name = c;  
        this.next = null  
    }  
    ...  
}
```



## *Let's go back to when we were writing our own Linked List and Node class*

For example, this Linked List could only hold Strings

```
public class Node {  
  
    private String name;  
    private Node next;  
  
    public Node(String c) {  
        this.name = c;  
        this.next = null  
    }  
    ...  
}
```

If we wanted to have Linked List hold Doubles, we would need to modify parts of the Node and LinkedList class

```
public class Node {  
  
    private double value;  
    private Node next;  
  
    public Node(double c) {  
        this.value = c;  
        this.next = null  
    }  
    ...  
}
```

## *Let's go back to when we were writing our own Linked List and Node class*

For example, this Linked List could only hold Strings

```
public class Node {  
  
    private String name;  
    private Node next;  
  
    public Node(String c) {  
        this.name = c;  
        this.next = null  
    }  
    ...  
}
```

If we wanted to have Linked List hold Doubles, we would need to modify parts of the Node and LinkedList class

```
public class Node {  
  
    private double value;  
    private Node next;  
  
    public Node(double c) {  
        this.value = c;  
        this.next = null  
    }  
    ...  
}
```

It would be nice if we could allow our Linked List to hold **any type of data** without needing to modify the source code of our classes

## *Let's go back to when we were writing our own Linked List and Node class*

For example, this Linked List could only hold Strings

```
public class Node {  
  
    private String name;  
    private Node next;  
  
    public Node(String c) {  
        this.name = c;  
        this.next = null  
    }  
    ...  
}
```

If we wanted to have Linked List hold Doubles, we would need to modify parts of the Node and LinkedList class

```
public class Node {  
  
    private double value;  
    private Node next;  
  
    public Node(String c) {  
        this.name = c;  
        this.next = null  
    }  
    ...  
}
```

It would be nice if we could allow our Linked List to hold **any type of data** without needing to modify the source code of our classes → We can achieve this using **Java generics**

We can **embed** a class within another class (although I don't recommend doing this unless the class is very small and/or the classes are strongly related to each other)

```
public class GenericLinkedList {
```

```
    public class Node<E>{
```

```
        E data;
```



The data can be  
*any object*

```
        Node<E> next;
```

```
        public Node(E data){
            this.data = data;
            this.next = null;
        }
```

When we create a Node  
object, we will give it  
some data type

```
        public E getData() {
            return this.data;
        }
```

getData() will now return  
some generic object E

```
        public Node getNext() {
            return this.next;
        }
```

```
    }
```

```
    private Node head;
    private int size;
```

Start of Linked List class

```
    public GenericLinkedList() {
        this.head = null;
        this.size = 0;
    }
```

```
    public <E> void add(E newData) {
```

<E> is used to indicate that this  
Node class will hold a **Generic  
object**. It can be *any* object

This is very helpful for cases when  
we might not know what data type  
we will be working with

<T> is also a value used to indicate a generic object