Jeffrey Cruz

10/9/17

CS 361

Homework 2



**Jeffrey Cruz**

**CS 361**

**Homework #2**

**Question 1: History of programming languages**

Put the following programming languages on a chronological timeline. The year must be provided. **In addition,** indicate the name of the designer of the programming language, where it was created (company, national lab, higher education institution etc.), and the country.

* Fortran

Year: 1957

Designer: John Bacus

Location: IBM

* Lisp
  + Year: 1958
  + Designer: John McCarthy
  + Location: Massachusetts Institute of Technology
* Cobol
  + Year: 1959
  + Designer: Team (Design was inspired by work of Grace Hopper)
  + Location: CODASYL
* ISETL
  + Year: 1969
  + Designer: Jacob T. Schwartz
  + Location: Courant Institute of Mathematical Sciences
* PASCAL
  + Year: 1970
  + Designer: Niklaus Wirth
  + Location: N/A
* C
  + Year: 1972
  + Designer: Dennis Ritchie
  + Location: AT&T Bell Laboratories
* Prolog
  + Year: 1972
  + Designer: Alain Colmerauer
  + Location: France
* SML
  + Year: 1973
  + Designer: Robin Milner
  + Location: University of Edinburgh
* ADA
  + Year: 1980
  + Designer: Jean Ichbiah
  + Location: United States Department of Defense
* C++
  + Year: 1983
  + Designer: Bjarne Stroustrup
  + Location: AT&T Bell Labs
* EIFFEL
  + Year: 1986
  + Designer: Bertrand Meyer
  + Location: Eiffel Software
* Perl
  + Year: 1987
  + Designer: Larry Wall
  + Location: Unisys
* Python
  + Year: 1991
  + Designer: Guido van Rossum
  + Location: Python Software Foundation
* Java
  + Year: 1995
  + Designer: James Gosling
  + Location: Sun Microsystems
* Ruby
  + Year: 1995
  + Designer: Yukihiro Matsumoto
  + Location: Japan
* Kotlin
  + Year: 2011
  + Designer: JetBrains
  + Location: Jetbrains

**Question 2:**

Consider the following code. Each *draw* method has a number.

public class Circle{

public double center\_x, center\_y;

public double radius;

public void draw() {

// **(1)** method to draw circle on the screen

}

public void draw(Color color) {

// **(2)** method to draw circle on the screen with a

// given color

}

}

public class ColoredCircle extends Circle{

public int color;

public void draw() {

// **(3)** method to draw the colored circle

}

}

1. Explain polymorphism on the code above.

The polymorphism in the above code is shown in the ColoredCircle class. A ColoredCircle is a ColoredCircle, but it is also a Circle. Meaning any object of the ColoredCircle class is also an object of the Circle class.

1. c is of type Circle and d is of type ColoredCircle. Can we write d = c;? Why?

We cannot do this because a Circle is not necessarily a ColoredCircle. There are things a ColorCircled has, that a Circle does not have, meaning we cannot assign a circle object to a coloredcircle object variable. The polymorphism does not work in this direction

1. c is of type Circle and d is of type ColoredCircle. Can we write c = d;? Why? What happens if we execute the code below? What method called *draw* is called? Why?

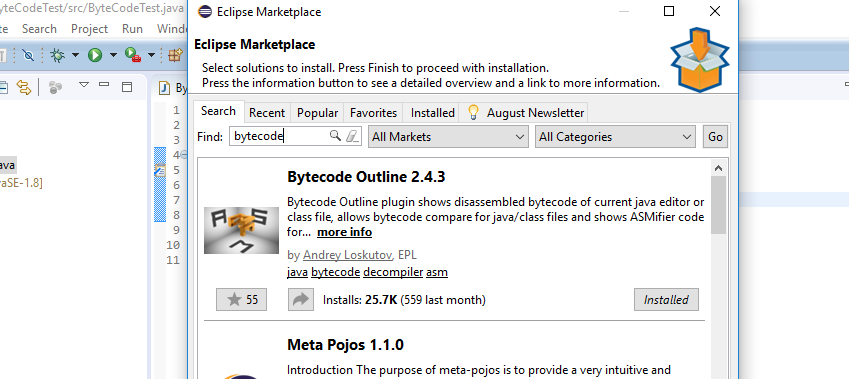
Yes we can do this because a ColoredCircle is a circle, meaning we can assign a ColorCircled to a Circle object variable. If we execute the code below then the variable c will refer to an object of the ColoredCircle class.

If we then run c.draw() then it will run the draw method from the ColoredCircle class. It does this because even though c is a circle reference, it is referring to a ColoredCircle object. Therefore the draw method from the circle class is overridden by the draw method from the ColoredCircle class.

c.draw();

**Question 3:**

Install the following Eclipse Bytecode Outline plugin from: <http://asm.objectweb.org/eclipse/index.html> or from the Eclipse MarketPlace.



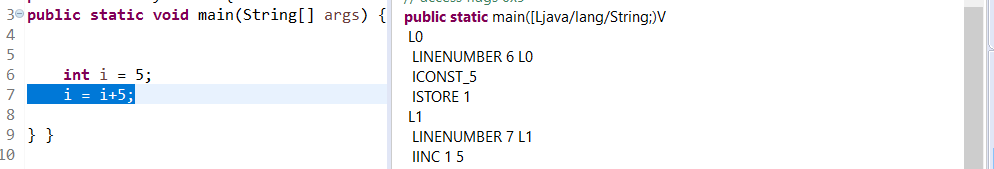
*[Dr. Scharff tested with the Neon version of Eclipse and with Eclipse Marketplace Byte Outline 2.4.3 plugin and it works! ]*

1. What Eclipse version are you using? – Version: Oxygen.1 Release (4.7.1)
2. What Java version are you using?- java version "1.8.0\_144"
3. What is the Bytecode generated by the following statements?

int i = 5;

i = i+5;

Explain the syntax of the Bytecode. Provide a screenshot to support your work.



The bytecode begins with the first line in the main method (LINENUMBER 6 L0) which is taking the number 5 (ICONST\_5) and storing it in variable i which is variable #1 in the bytecode(ISTORE 1). Then on the next line (LINENUMBER 7 L1) it takes whatever is stored in variable #1 and increments it by 5 (IINC 1 5).

1. Compare the Bytecode generated by the 2 functions below and write down your conclusions.

Provide screenshots to support your work.

**public** **static** **int** sum\_for(**int** n) {

**int** i = 0, sum = 0;

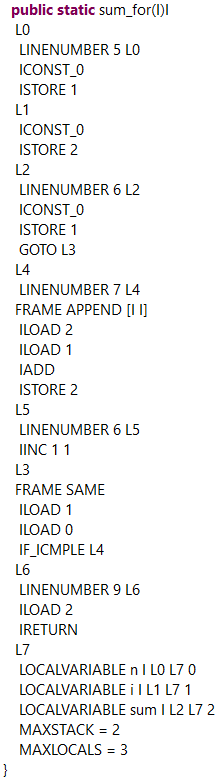
**for** (i = 0; i <= n; i++) {

sum += i;

}

**return** sum;

}



**public** **static** **int** sum\_while(**int** n) {

**int** i = 0, sum = 0;

**while** (i <= n) {

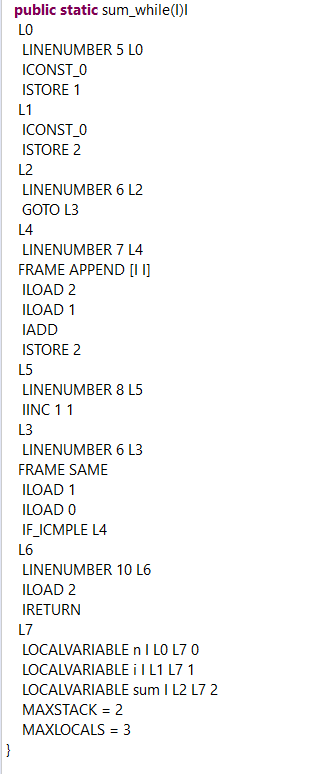
sum += i;

i++;

}

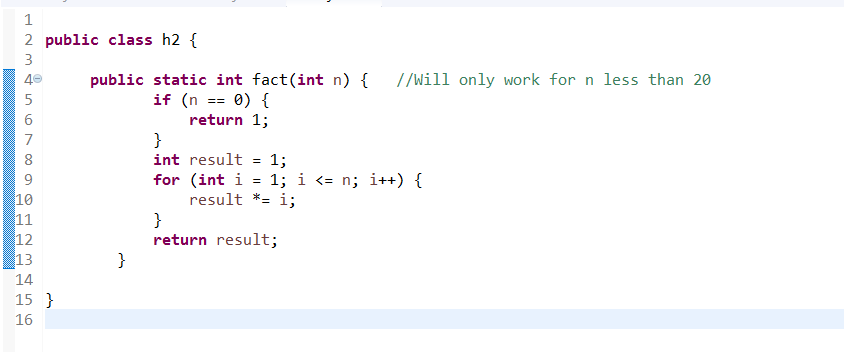
**return** sum;

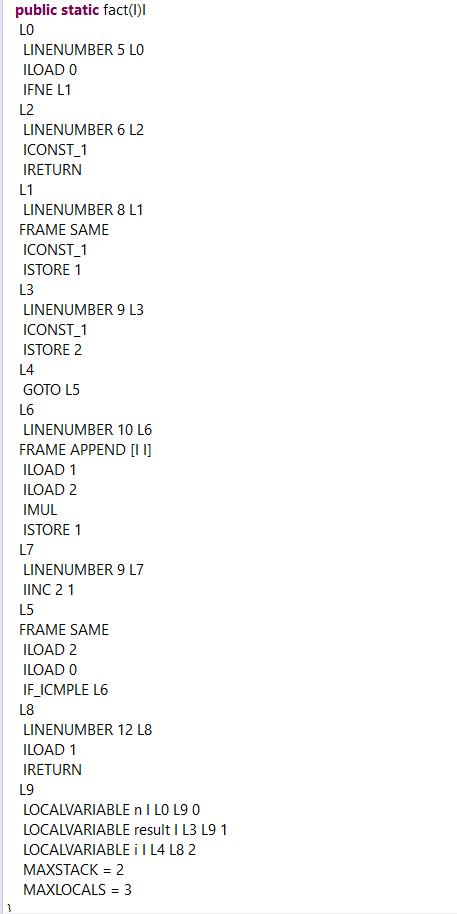
}



Comparing these two bytecodes we can see that a for loop and a while loop in java are implemented the exact same way in the Java Bytecode. This shows us that internally, there is no difference between a while loop and a for loop in Java. It is merely a difference of how it is written in the higher-level language, but with respect to the bytecode, they are both the same.

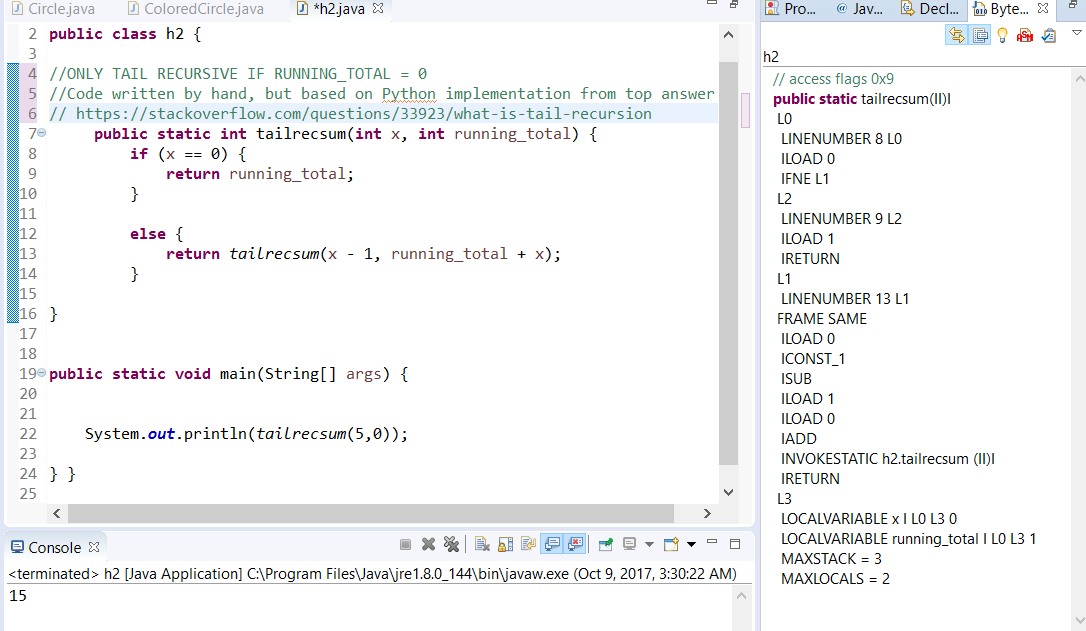
1. Write the factorial function (with the profile: public static fact(int n)) and describe the bytecode generated by this function.





This bytecode is basically a more technical description of the factorial function that I wrote. It begins by with by loading the number 0 and then does the if statement with IFNE (if not equal) meaning if n is not equal to 0, then we go to L1 (which is line number 8, meaning we avoid whats inside the if statement), otherwise we go from L0 to L2 which loads the constant 1 and returns it. The rest of the bytecode describes how the for loop is implemented in java, it loads all the variables, uses GOTO statements to simulate the looping, and checks to make sure that the variable i is still less than or equal to n (IF \_ICMPLE). Then the actual body of the for loop which is in L6 just loads the two variables result and I, multiples them, then stores them back into the result variable. This bytecode would look different had I chosen to write this function recursively instead of iterively. One of the biggest differences would be an INVOKESTATIC with the name of the class that contains the fact function, which would be the bytecode that recursively calls back the fact function.

1. Choose a tail recursive function and describe the bytecode generated by this function. Compare with the code generated for a recursive function obtained in c).



This code is identical to the code generated for a recursive function (Aside from the differences between the functions themselves of course). This is because Java does not have optimization for tail recursive functions yet. Apparently this is because that tail optimization would change the number of frames on the stack, and this would break some security-sensitive methods that rely on counting stacks between frames between jdk library code. It has been said by java developers though that it will eventually get done.

**References**

* The Java Virtual Machine Specification <https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf> (Java 8 SE)
* Java Bytecode Basics <http://www.javaworld.com/javaworld/jw-09-1996/jw-09-bytecodes.html> (1996)
* <http://www.beyondjava.net/blog/java-programmers-guide-java-byte-code/> (2015)

**Question 4:**

1. Write a PROLOG program that describes the British family until nowadays. Kate, William and their children should be cited in the facts. Your program will start with the facts available in the slides (slide 31) and ends with Kate, William and their children.

Let P(x,y) be the relationship that if P(x,y) then X is the parent of y

Let G(x,y) be the relationship that if G(x,y) then X is the grandparent of y

P(Edward VII, George V).

P(Victoria, Edward VII).

P(Alexandra, George V).

P(George VI, Elizabeth II).

P(George V, George VI).

P(Elizabeth II, Charles)

P(Charles,William)

P(William,George)

P(Kate,George)

P(William,Charlotte)

P(Kate,Charlotte)

G(x,y):-P(x,z),P(z,y).

1. Write a **rule** that describes the father predicate. *Father(X,Y)* means that *X* is the father of *Y*.

Father(X,Y):-M(X),P(X,Y)

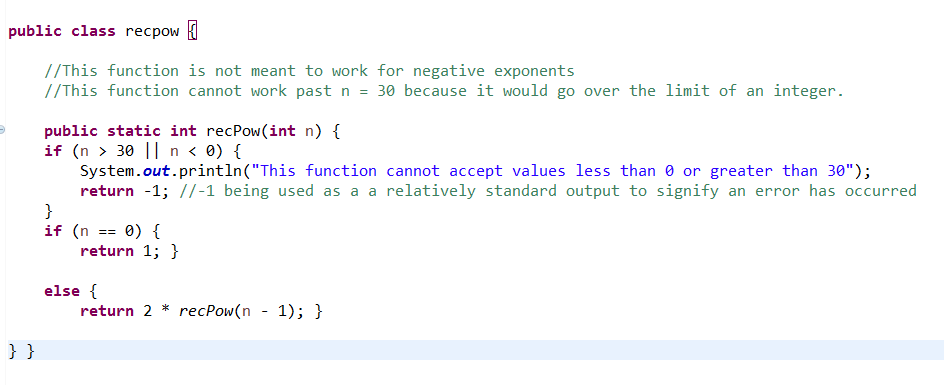
Let M(X) be the function that if M(X) then X is male

**Question 5:**

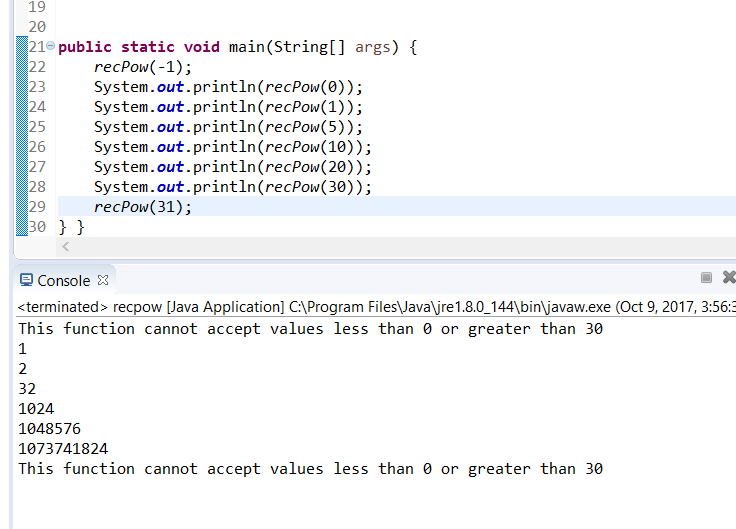
Write a **recursive** function *recPow* that computes 2n for n >= 0 in Java. The function will have the following profile:

public static int recPow(int n)

The function must consider all cases and be tested exhaustively. Show your testing!



TESTING:



**Question 6:**

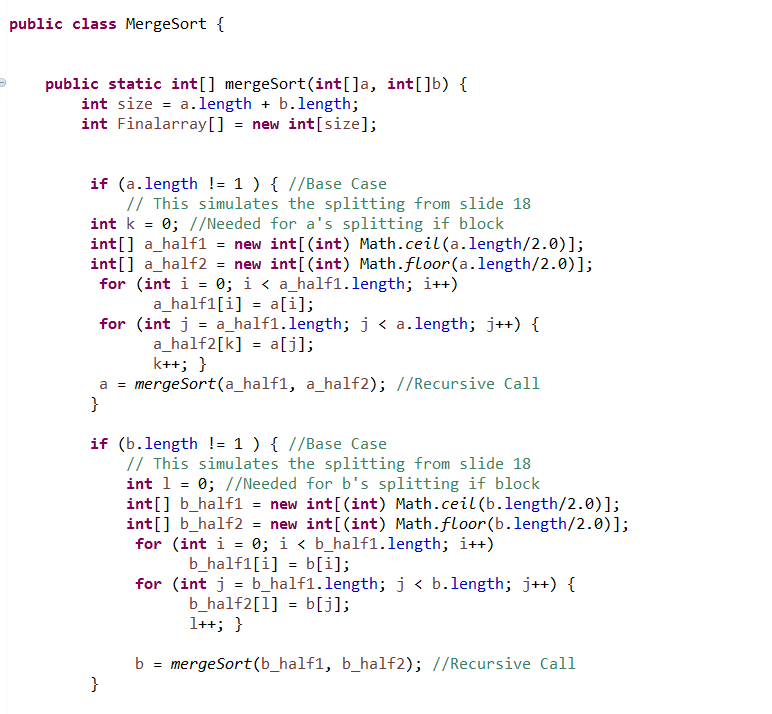
Write a **recursive** function merge that merges 2 arrays in Java. . The function will have the following profile:

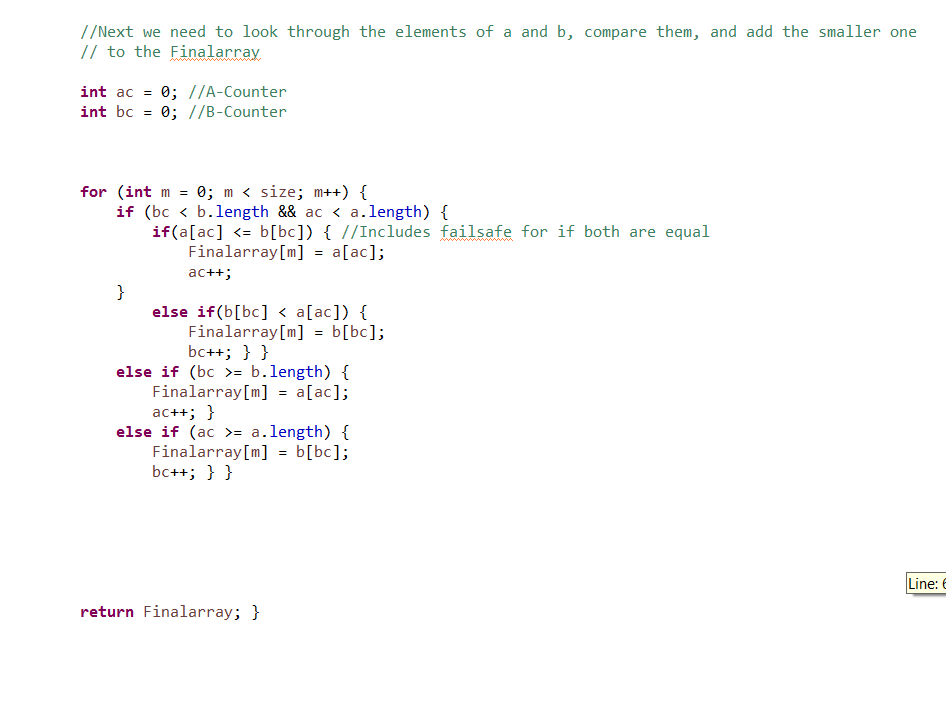
public static int[] mergeSort(int[] a, int[] b)

You will use the split function of slide 18 (odd and even positions).

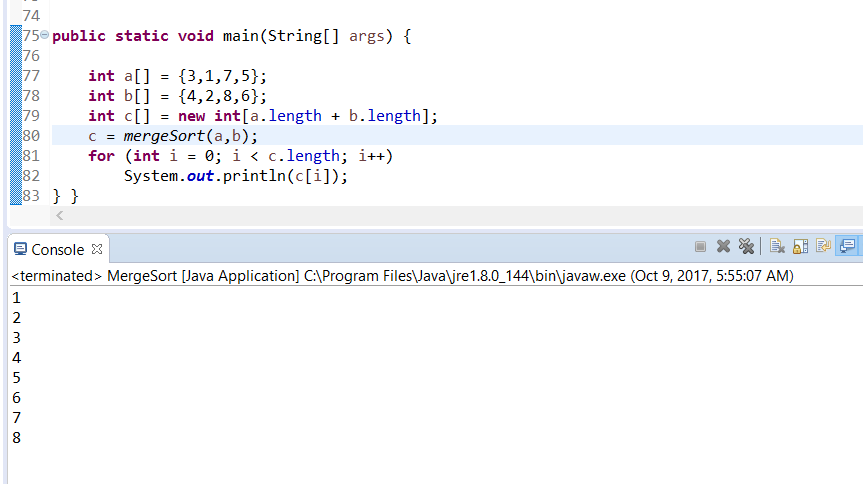
The function must be tested exhaustively. Show your testing!

If you use code online, you will need to cite your sources.

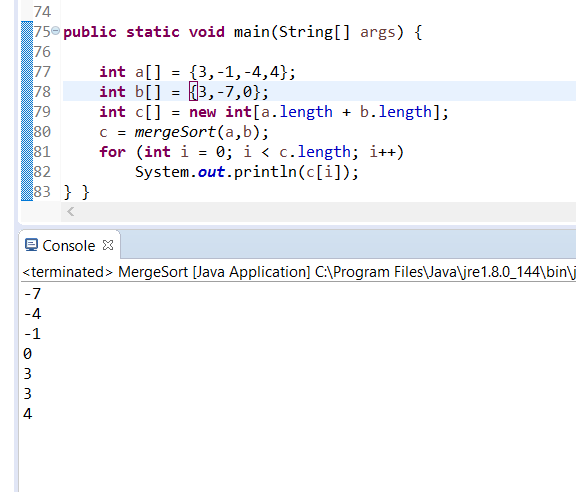




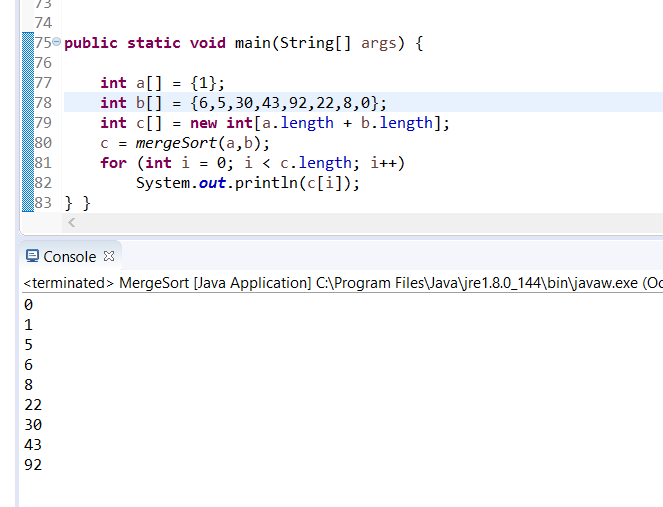
TEST CASE 1 : Two Arrays equal length



TEST CASE 2 : Negative numbers, different array lengths



Test Case 3 : One array with length 1



Test Case 4 : Two arrays with length 1

