16. [10 marks] You are given the following Student class.

class Student {  
private int labID; private String tutID;

Student(int labID, String tutID) { this.labID = labID; this.tutID = tutID;

}

public int getLabID() { return this.labID;

}

public String getTutID() { return this.tutID;

}

@Override  
public String toString() {

return labID + " : " + tutID; }

}

We would like to design a class management application where students may be sorted by their lab group only, their tutorial group only, lab followed by tutorial groups, or tutorial followed by lab groups. We can achieve this by storing the comparators (one or more) in a list, and process the comparators one by one when sorting students.

In the following questions, assume that we are sorting students in the order of lab groups first, followed by tutorial groups.

(a) [3 marks] Define two student comparators, comp1 that compares students by lab group only, and comp2 that compares students by tutorial group only.

ANSWER:

page2image84778512

(b) [2marks]Define a List of comparators comparatorList to store the comparators in question 16a.

ANSWER:

(c) [3 marks] Design the StudentComparator class that implements the interface java.util.Comparator to perform the sorting method as represented in the list of comparators in question 16b.

ANSWER:

(d) [2 marks] Write a program fragment to show how the following list of four stu- dents can be sorted by lab group, then by tutorial group, and printed out.

• Student in lab group 1, and tutorial group a2 • Student in lab group 3, and tutorial group b1 • Student in lab group 3, and tutorial group a2 • Student in lab group 1, and tutorial group b1

ANSWER:

page4image84900064

17. [6 marks] By completing the following IntStream pipeline, write an addition tutor that takes in two integer values i and j and prints instructions on how long addition is performed.

As an example the numbers 2089 and 65 when added gives the following output

9 + 5 = 4 carry 1  
8 + 6 + (1) = 5 carry 1 0 + 0 + (1) = 1 2+0=2

2 10 18 9 +65 2154

Another example showing 1 + 9999 gives the following output

1 + 9 = 0 carry 1

1 + 1 19 19 19 9

10000

0 + 9 + (1) = 0

0 + 9 + (1) = 0

0 + 9 + (1) = 0

0 + 0 + (1) = 1

ANSWER:

carry 1

carry 1

carry 1

CS2030

page5image84915824page5image84907296page5image84910416

public static void main(String[] args) { Scanner sc = new Scanner(System.in); int i = sc.nextInt();  
int j = sc.nextInt();

IntStream // continue the stream pipeline below .

18. [9 marks] Many ways have been devised to multiply two large integers. One of these ways is attributed to Anatoly Karatsuba in 1960 and is described below using the example 1234 × 567 = 699678.

Step 1. Step 2.

Step 3. Step 4. Step5. Step 6. Step 7.

If necessary, pad the smaller number with leading zeros to make two numbers of the same length L, i.e. 1234 and 0567.

Divide the two numbers into equal left and right portions and label them

a,b,c,d

a=12 b=34 c=05 d=67

Calculate ac = 12×5 = 60  
Calculate bd = 34 × 67 = 2278  
Calculate(a+b)(c+d)=46×72=3312  
Calculate the result of step (5) - step (4) - step (3) = 3312 − 2278 − 60 = 974 Add the partial results with zero padding

600000 from step (3) by padding L trailing zeroes  
2278 from step (4) with no additional padding of trailing zeroes

97400 from step (6) by padding L/2 trailing zeroes 699678 page6image84790736page6image84790944page6image84791152page6image84791360page6image84791568

Notice that multiplying two large numbers require three smaller multiplications which can be done independently in steps (3), (4) and (5).

Your task is to define a Task class that extends RecursiveTask<BigInteger> and computes the multiplication in parallel. The following methods from the BigInteger class may be useful to you.

* public BigInteger(String val)  
  Translates the decimal String representation of a BigInteger into a BigInteger.
* public BigInteger add(BigInteger val) Returns a BigInteger whose value is this + val.
* public BigInteger subtract(BigInteger val) Returns a BigInteger whose value is this - val.
* public BigInteger multiply(BigInteger val)  
  Returns a BigInteger whose value is this \* val. Use this when the numbers to be multiplied are of length less than 2.
* public BigInteger pow(int exponent)  
  Returns a BigInteger whose value is this raised to the power of exponent.
* public String toString() Returns the String representation of this BigInteger. You may also use other methods from the Java API.

ANSWER: