

Name: ANSWER KEY Section: ALL CM: N/A

CSSE 220—Object-Oriented Software Development

Final Exam – Part 1, Feb. 27, 2014

This exam consists of two parts. Part 1 is to be solved on these pages. You may use the back of a page if you need more room. Please indicate on the front if you do so. Part 2 is to be solved on your computer. You will need network access to download template code and upload your solution for part 2. Please disable IM, email, Lync, and other such communication programs before beginning the exam.

Resources for Part 1: One 8 1/2" by 11" double-sided sheet of notes, closed book, closed computer, closed electronic devices.

Resources for Part 2: Open book, notes, and computer. Limited network access. You may use the network only to access your own files, the course Piazza and Moodle sites, the course web pages, the textbook's site, and Oracle's Java website. Any communication with anyone other than the instructor or a TA during the exam may result in a failing grade for the course.

Part 1 is included in this document. You should read over all of the questions before beginning work, but...

You must turn in part 1 before accessing the resources for part 2.

Problem	Poss. Pts.	Earned
1	7	_____
2	6	_____
3	4	_____
4	3	_____
5	6	_____
6	6	_____
7	3	_____
Paper Part Subtotal	35	_____
Computer Part Subtotal	65	_____
Total	100	_____

Part 1—Paper Part

1. (7 points) Consider the following initial array configuration. In each question, assume we want to sort the array in **ascending** order (that is, from smallest to largest).

5	7	3	4	2	8	15	6	0	10
---	---	---	---	---	---	----	---	---	----

a. (3 points) Suppose the **insertion** sort algorithm from class is applied to the **initial array above**. Show the state of the array immediately following the first **three** executions of the outer loop. Note: the sorted part of the array initially contains 1 element.

5	7	3	4	2	8	15	6	0	10
---	---	---	---	---	---	----	---	---	----

↑

3	5	7	4	2	8	15	6	0	10
---	---	---	---	---	---	----	---	---	----

↑

3	4	5	7	2	8	15	6	0	10
---	---	---	---	---	---	----	---	---	----

↑

b. (3 points) Suppose the **selection** sort algorithm from class is applied to the **initial array above**. Show the state of the array immediately following the first **three** executions of the outer loop.

0	7	3	4	2	8	15	6	5	10
---	---	---	---	---	---	----	---	---	----

↑

0	2	3	4	7	8	15	6	5	10
---	---	---	---	---	---	----	---	---	----

↑

0	2	3	4	7	8	15	6	5	10
---	---	---	---	---	---	----	---	---	----

↑

c. (1 point) Suppose the **merge** sort algorithm from class is applied to the **initial array above**. Show the state of the two sub-arrays immediately before the final merge.

2	3	4	5	7	0	6	8	10	15
---	---	---	---	---	---	---	---	----	----

2. (6 points) Answer the questions below on the sorting algorithms that we discussed in class.

Consider a finite long list of random salaries that are to be sorted by a Big Data processing company. The company needs to sort the salaries as efficiently as it can, but its employees are only familiar with **InsertionSort**, **MergeSort**, and **SelectionSort**.

a. Which sorting algorithm should the company use?

Answer: MergeSort

b. Justify your answer using the **Big-O runtime** of your selected algorithm.

$O(n \log n)$ efficiency.
Efficiency of other algorithms is worse

c. Suppose that the list was mostly sorted with only a relatively small constant number of salaries that are unsorted. Is there another algorithm you would consider that you didn't consider before, given the list is mostly sorted?

Circle one. **Yes** or No.

If you answer Yes to the above question, which additional algorithm should the company consider?

Answer: InsertionSort

3. (4 points) Use the following classes to answer the question that follows on Fork-Join Parallelism.

```
1 import java.util.concurrent.RecursiveTask;
2 public class FibTask extends RecursiveTask<Long> {
3     private static final int THRESHOLD = 6;
4     private FibonacciProblem problem;
5     public long result;
6     public FibTask(FibonacciProblem problem) {
7         this.problem = problem;
8     }
9
10    @Override
11    public Long compute() {
12        if (problem.n < THRESHOLD) { // easy problem, don't bother with parallelism
13            result = problem.solve();
14        }
15        else { // create and execute 2 parallel tasks
16            FibTask worker1 = new FibTask(new FibonacciProblem(problem.n-1));
17            FibTask worker2 = new FibTask(new FibonacciProblem(problem.n-2));
18            worker1.fork();
19            long tempRes = worker1.join();
20            long tempResult = worker2.compute();
21            result = tempResult + tempRes;
22        }
23        return result;
24    }
25
26    public static void main(String[] args) {
27        FibTask task = new FibTask(new FibonacciProblem(10));
28
29        // Create a fork-join pool with as many tasks as there are processors
30        ForkJoinPool pool = new ForkJoinPool(8);
31
32        // Assign and execute the task by using the invoke method
33        pool.invoke(task);
34    }
35 }
```

```

36 public class FibonacciProblem {
37     public int n;
38     public FibonacciProblem(int n) {
39         this.n = n;
40     }
41
42     public long solve() {
43         return fibonacci(n);
44     }
45
46     private long fibonacci(int n) {
47         if (n <= 1)
48             return n;
49         else
50             return fibonacci(n-1) + fibonacci(n-2);
51     }
52 }

```

Consider the execution of the program above that uses the defined classes. The program correctly computes the desired results; however, no speedup in computation is seen even though the Fork-Join framework is used.

- a. Write the line number(s) from the code above that prevent the program from computing the results as quickly as expected (*i.e.*, by taking advantage of parallelism).

Answer: 19 & 20

- b. What changes would you make in the code to overcome that lack of parallelism?

Answer: Swap lines 19 & 20

4. (3 points) Predict the output for the code snippet below.

```

int[] data = {1, 2, 3, 7, 8, 9};
int[] result = funct(data, 0, data.length/2);
for(int i = 0; i < result.length - 1; i++) {
    System.out.print(result[i] + ", ");
}
System.out.print(result[result.length - 1]);

// elsewhere...
public static int[] funct(int[] data, int f, int b){
    if(b == data.length) {return data;}
    int temp = data[b];
    data[b] = data[f];
    data[f] = temp;
    return funct(data, f+1, b+1);
}

```

Output: 7, 8, 9, 1, 2, 3

5. (6 points) Give the Big-O runtime for each of the code snippet below.

a.

```
public void funct1(int n) {
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += i * i * i;
        for (int j = 0; j < 20; j++) {
            sum += j * i;
        }
    }
}
```

Answer:

$O(n)$

b.

```
public static void funct2(int n) {
    for (int i = 0; i < n; i++) {
        System.out.println("Break!");
    }
    for (int i = 0; i < n; i++) {
        for (int j = i; j < n; j++) {
            System.out.println("Howdy");
        }
    }
}
```

Answer:

$O(n^2)$

c.

```
// Assume data is a square 2d array.
// If data is a 10 * 10 array, then n == 10.
public void funct3(int[][] data) {
    for (int i = 0; i < data.length; i++) {
        selectionSort(data[i]);
    }
}
```

Answer:

$O(n^3)$

6. (6 points) Consider the following **four** related declarations:

```
interface Type1 {
    public void move();
}

abstract class Type2 implements Type1 {
    public void move() {
        System.out.print("ok ");
        this.sing();
    }

    public abstract void sing();
}
```

```
class Type3 extends Type2 {
    public void sing() {
        System.out.print("meow ");
    }
}

class Type4 extends Type2 {
    public void move() {
        super.move();
        this.sing();
    }

    public void sing() {
        System.out.print("howl ");
    }

    public void speak() {
        System.out.print("woof ");
    }
}
```

Consider each of the following code snippets independently. (That is, errors on one don't affect the others.) For each, write the output, or if it is an error write the type (compile-time or run-time).

a. `Type2 a = new Type3();` ✓
`((Type4) a).speak();` ✗

runtime error

b. `Type1 b = new Type3();`
`b.move();`

ok meow

c. `Type4 c = new Type2();`
`c.move();`

compile-time error

d. `Type4 d = new Type4();`
`d.speak();`

woof

e. `Type3 e = new Type4();` ✗
`e.sing();`

compile-time error

f. `Type2 f = new Type3();`
`f.sing();`

meow

7. (3 points) Use the **Team** class, declared below, to answer questions on the appropriate use of visibility modifiers.

```
package basketballTeams;

public class Team {
    private String name;
    protected int rank;
    public int wins;

    public Team(String name, int rank, int wins){
        this.name = name;
        this.rank = rank;
        this.wins = wins;
    }

    // The rest of this class is elided.
}
```

```
1 package basketballTeams;
2
3 public class BestTeam extends Team{
4     public BestTeam(String name, int rank, int wins, int points) {
5         super(name, rank, wins);
6     }
7
8     public static void main(String[] args) {
9         Team team1 = new Team("Florida", 1, 25);
10        System.out.println("The best team is " + team1.name);
11        System.out.println("It's ranking is" + team1.rank);
12        System.out.println("The best team has won" + team1.wins + " games");
13    }
14 }
```

a. The **BestTeam** class above creates and uses a **Team** object to query information about an NCAA basketball team. However, the use of visibility modifiers in the **BestTeam** code prevents the code from compiling. Circle **True** or **False** as your answer for each of the statements below.

Line 10 produces a compilation error.

True

or

False

Line 11 produces a compilation error.

True

or

False

Line 12 produces a compilation error.

True

or

False

```

1 package badteams;
2
3 import basketballTeams.Team;
4
5 public class WorstTop25Team {
6
7     public static void main(String[] args) {
8         Team team2 = new Team("New Mexico", 25, 21);
9         System.out.println("The worst team in the top 25 is " + team2.name);
10        System.out.println("It's ranking is" + team2.rank);
11        System.out.println("This team has won" + team2.wins + " games");
12    }
13
14 }

```

b. The **WorstTop25Team** class does not compile also because of inaccurate use of visibility modifiers. Circle **True** or **False** as your answer for each of the statements below.

Line 9 produces a compilation error. **True** or **False**

Line 10 produces a compilation error. **True** or **False**

Line 11 produces a compilation error. **True** or **False**

You must turn in your solutions to part 1
before accessing resources for part 2.