

Name: _____

CSSE 220—Object-Oriented Software Development

Final Exam, Nov. 17, 2008

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 using your computer. You will need network access to download template code and upload your solution for part 2. Please disable IM, email, and other such communication programs before beginning the exam.

Resources for Part 1: Open book and notes, closed computer.

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 ANGEL site and web pages, the textbook's site, Sun's Java website, and Logan Library's Safari Tech Books Online. Any communication with anyone other than the instructor or a TA during the exam may result in a failing grade for the course.

Parts 1 and 2 are 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	4	_____
	2	8	_____
	3	18	_____
	4	24	_____
	5	20	_____
	6	7	_____
	7	4	_____
	8	14	_____
	9	8	_____
	Paper Part Subtotal	107	_____
C1. recursion.Power (1 point per unit test)		8	_____
C2. DoublyLinkedList reverse() (1 point per unit test)		5	_____
C2. DoublyLinkedList removeFirst() (2 points per unit test + 2 for remove)		12	_____
C2. DoublyLinkedList addInOrder() (2 points per unit test)		10	_____
C3. list.Matchers (2 points per unit test, except first one)		8	_____
	Computer Part Subtotal	43	_____
	Total	150	_____

Part 1—Paper Part

1. (4 points) What is wrong with the recursive algorithm below? (You do not have to fix the problem, just identify what it is.)

```
/**
 * @param arr
 * @return the sum of the elements in the given array
 */
public static int sumElements(int[] arr) {
    return sumElements(arr, 0);
}

// Helper method
private static int sumElements(int[] arr, int i) {
    return arr[i] + sumElements(arr, i + 1);
}
```

Answer:

2. (8 points) For each of the scenarios below, indicate which of the *sorting algorithms* we studied—**selection sort, insertion sort, merge sort, or quicksort**—would be best. Justify your answer.

a. Implementing a course catalog system where the data is kept sorted by course number.

- Algorithm:
- Justification:

b. Implementing a music program where the user can choose to order the songs by song title, artist, album, or musical style.

- Algorithm:
- Justification:

3. (18 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).

3	6	5	8	4	2	7	1
---	---	---	---	---	---	---	---

- a. 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.

--	--	--	--	--	--	--	--

- b. Suppose the **selection** sort algorithm from class is applied to the **initial array above**. Show the state of the array immediately following the first **two** executions of the outer loop.

--	--	--	--	--	--	--	--

- c. 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.

--	--	--	--	--	--	--	--

4. (24 points) For each of the scenarios below, indicate which of the *data structures* we studied—**array list, linked list, stack, queue, hash set, tree set, hash map, or tree map**—would be best. Justify your answer.

- a. Modeling the waiting list to get into a section of a course that is over-capacity.

Data Structure: _____

Justification:

- b. Tracking the point values for each letter in the game of Scrabble. (In Scrabble common letters like E, S, and T are worth 1 point each. Less common letters are worth more points, with maximum points for Q and Z, which are worth 10 points each.)

Data Structure: _____

Justification:

- c. Simulating a paper in-box, where new items are placed on the top of the pile and work items are taken from the top of the pile.

Data Structure: _____

Justification:

- d. Tracking all the courses taken by each student in a university.

Data Structure: _____

Justification:

5. (20 points) Predict the output of each of the following code blocks:

a.

```
Stack<Integer> s = new Stack<Integer>();
s.push(1);
s.push(2);
System.out.print(s.pop());
s.push(3);
System.out.print(s.pop());
s.push(4);
s.push(5);
System.out.print(s.pop());
System.out.print(s.pop());
System.out.print(s.pop());
```

Answer: _____

b.

```
HashMap<Character,String> map =
    new HashMap<Character, String>();
map.put('a', "Turkey");
map.put('b', "Pumpkin Pie");
map.put('a', "Tofurkey");
System.out.print(map.get('b'));
System.out.print(map.get('a'));
```

Answer: _____

c.

```
HashMap<Character,Set<Integer>> map =
    new HashMap<Character, Set<Integer>>();

Set<Integer> one = new TreeSet<Integer>();
one.add(1);

Set<Integer> two = new TreeSet<Integer>();
two.add(2);

map.put('x', one);
map.put('y', two);
map.put('z', one);
map.get('z').add(3);

System.out.print(map.get('x'));
System.out.print(map.get('y'));
System.out.print(map.get('z'));
```

Answer: _____

6. (7 points) Below are two algorithms for **removing duplicates** from an array. Study them, then answer the questions that follow.

- For each item in the array, count how many times it appears in the full array. If the count is greater than 1, then remove the item.
- First sort the array. Next, for each item in the array, if it is the same as its neighbor, remove the item.

- a. Which algorithm is faster? (circle one) the first the second
- b. Explain your answer to part *a*:

7. (4 points) Suppose an algorithm takes 3 seconds for an input of size 10,000. If the algorithm is $O(n^3)$, approximately how many seconds does it take for an input of size 20,000? (Circle one.)

1.5 3 4.5 6 9 12 18 24 30

8. (14 points) Consider the following code:

```
for (int i = 0; i < N - 1; i++) {  
    for (int j = 0; j < N - 1; j++) {  
        foo();  
    }  
}
```

a. Exactly how many times is `foo()` called, in terms of N ? _____

b. Fill in the blank: Your answer to part *a* is $O(\text{_____})$.

c. Suppose the condition on the inner loop were changed to $j < i - 1$.

Now exactly how many times would `foo()` be called, in terms of N ? _____

d. Fill in the blank: Your answer to part *c* is $O(\text{_____})$.

9. (8 points) Suppose the input file to the Markov Chain algorithm contains only the following line:

x xo om x xo m x xo om m x xo x

Let the prefix length be 2. For each of the following, indicate whether or not it would be a valid output of the algorithm:

x xo x valid invalid

x xo m x xo x valid invalid

x om x valid invalid

x xo om x xo x xo x valid invalid

Part 2—Computer Part

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 ANGEL site and web pages, the textbook's site, Sun's Java website, and Logan Library's Safari Tech Books Online. Any communication with anyone other than the instructor or a TA during the exam may result in a failing grade for the course.

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

Instructions. You must actually get these problems working on your computer. Almost all of the credit for this problem will be for code that actually works. There are several different small methods to write, so you can get a lot of partial credit by getting some of them to work. If you get every part working, comments are not required. If you do not get a method to work, comments may help me to understand enough so I can give you (a small amount of) partial credit.

After you have handed in part 1, **begin part 2 by checking out the project named *FinalExam* from your course SVN repository.** (Ask for help immediately if you are unable to do this.)

When you have finished the problems, and more frequently if you wish, you should **submit your code by committing it to your SVN repository.** We will check commit logs, so you must be careful not to commit anything after the end of the exam. For grading, we will ensure that the included JUnit tests have not been changed.

Problem Descriptions

C1. (8 points) In `recursion.Power`, complete a **recursive** implementation of the method specified in the javadoc below. Your solution must be recursive to earn any credit. JUnit tests are provided in `recursion.PowerTest`.

```
/**
 * Calculates x raised to the nth power using recursion.
 *
 * @param x
 * @param n must be non-negative
 * @return x^n
 */
public static int power(int x, int n) { ... }
```


C2. (27 points) In the list package you will find a version of doubly linked list like the one from class. For this problem, your task is to add some additional methods. The javadoc specifications for those methods are given below.

JUnit tests are provided in list.DoublyLinkedListTest.

```
/**
 * Returns a NEW doubly linked list consisting of the elements of this list
 * in reverse order. Leaves the original list unchanged.
 *
 * @return a new, reversed list
 */
public DoublyLinkedList<E> reverse() {
    // TODO: implement reverse()
}

/**
 * Removes the first element from this list.
 *
 * @return the first element of this list
 * @throws NoSuchElementException if this list is empty
 */
public E removeFirst() throws NoSuchElementException {
    // TODO: implement removeFirst(), For full credit you must add and use a
    // "remove()" method in the Node inner class.
}

/**
 * Adds the given element to the list in the first position such that the
 * subsequent list element, if any, is larger than the new one and all
 * preceding list elements are smaller than the new one. If the list is
 * originally empty, then this is equivalent to calling addFirst(element).
 *
 * If all additions to a list are made using addInOrder(), then the resulting
 * list will be sorted in ascending order.
 *
 * @param element to be added to the list
 */
public void addInOrder(E element) {
    // TODO: implement addInOrder()
}
```

C3. (8 points) The provided `list.DoublyLinkedList` code includes a method `countMatches()` that takes a `ListItemMatcher` instance and returns a count of the number of matching items in the list. The `ListItemMatcher` interface is given in Figure 1 below.

Complete the TODO item in `list.Matchers` by writing an implementation of `ListItemMatcher` that matches strings with more than 3 characters.

JUnit tests are provided in `list.MatchersTest`.

```
/**
 * This is the type of a function object for matching items in a list.
 *
 * @param <E> the type of elements in the list
 */
public interface ListItemMatcher<E extends Comparable<E>> {
    /**
     * @param element the element to be tested
     * @return true if the given element is a match
     */
    boolean isMatch(E element);
}
```

Figure 1: The `ListItemMatcher` interface.