Name:_____ Student ID:_____

CSS 443: Mid Term Exam

Feb 19, 2003
3:30pm – 5:35pm

- This exam contains **4 problems** on **6** pages. You have 125 minutes to earn 100 points.
- This exam is open book.
- Do not spend too much time on any problem. Read them all through first and attack them in the order that allows you to make the most progress.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat!
- Please show all the details of your work and justify your answers. As in all cases in this class, correct answers without justifications will **not** earn you full credit.

Problem 1 (25%)	
1a (5%)	
1b (5%)	
1c (15%)	
Problem 2 (30%)	
2a (5%)	
2b (8%)	
2c (2%)	
2d (6%)	
2e (6%)	
2f (3%)	
Problem 3 (15%)	
Problem 4 (30%)	
4a (10%)	
4b (10%)	
3c (10%)	

Problem 1 (25%). Study our homework assignment #1 closely here are my new solution to the thread termination sequencing problem. Here is my new and improved TsyncObject:

```
class TsyncObject {
        private int fID;
        private int fCount;  // for keeping track of the number of treads

        // constructor
        TsyncObject()  {    fID = -1;
                            fCount = 0;          }

        // counter methods
        synchronized public void IncrementCount() {      fCount++; }
        synchronized public int GetCount() {    return fCount; }

        // id comparison methods
        synchronized public void SetID(int id) {          fID = id;  }
        synchronized public boolean IsID(int id) {
                boolean found = (fID == id);
                if (found) {
                        if (fID == 4)
                                fID = 0;
                        else
                                fID++;
                }
                return found;
        }

};
```

This is what my new TcomputeThread looks like:

```
class TcomputeThread extends Thread {

        // class variable:     All threads share this!!
        static private TsyncObject sfSyncObject;                    // for synchronization ...

        // class method
        static void SetClassVariables(TsyncObject syncObj) { sfSyncObject = syncObj; }

        // instance variable
        private    int                  fThreadID;

        // constructor
        TcomputeThread(int id) {     fThreadID = id; }

        public void run() {
                while ( !sfSyncObject.IsID(fThreadID) );
                System.err.println("Thread[" + fThreadID + "]: dieing ... ");
                sfSyncObject.IncrementCount();
        }

};
```

Where my simpleThread() looks like:

```
public class TsimpleThread {
        //  Driver program for TsimpleThread.
        public static void main(String args[]) {
                int numThreads = 5;
                TsyncObject syncObj = new TsyncObject();
                TcomputeThread.SetClassVariables(syncObj);
                TcomputeThread myThreads[] = new TcomputeThread[numThreads];

                // now initialize and start all the threads
                for (int i=0; i<numThreads; i++) {
                        myThreads[i] = new TcomputeThread(i);
                        myThreads[i].start();
                }

                System.err.println("5 Threads created and running ... which to die first? ");

                … Set up to read from keyboard …
                int start;              // which thread to start the termination …
                … input start  from the keyboard ...

                syncObj.SetID(start);

                // make sure all child threads are terminated before
                // we print out our message and terminate!
                while (syncObj.GetCount() != 5);

                // this is to make sure the main termination message
                // will only appear after all the children threads has die
                //
                System.err.println("Main User Interaction Thread exiting ... !");
        }
};
```

a. **(5%)** When the user enters a 2 as input, what would the output to look like? Please provide descriptions justifying why the output. If you run the above program n-times, where n is some large number, would you get the same output every time? Would this program terminate?

b. **(5%)** After looking at the code for 8.6 seconds, Student Mary once again argued that this is a really bad design because the threads are still performing busy wait. Is Student Mary correct? Please justify your answers.

c. **(15%)** Learned her lesson from hw#1, Student Mary claimed that none of the methods in the TsyncObject needs to be ***synchronized***. Please evaluate Mary's argument by going through each of the 4 methods in TsyncObject class and describe if the method needs to be synchronized.

Problem 2 (30%). Refer the following array based linked list implementation:

```
int SystemMemory[20];

int nextField(int addr) { return SystemMemory[addr+1]; }
int keyField(int addr) { return SystemMemory[addr]; }


 int WhyDoIRecurse(int newList, int link) {
        int cntPtr, prePtr;
        if ( link != -1 ) {
                int next = nextField(link);
                SystemMemory[link+1] = -1;

                if (keyField(link) <= keyField(newList)) {
                        SystemMemory[link+1] = newList;
                        newList = link;
                } else {
                        cntPtr = newList;
                        prePtr = cntPtr;
                        while ( (cntPtr != -1) &&
                                (keyField(link) > keyField(cntPtr)) ) {
                                prePtr = cntPtr;
                                cntPtr = nextField(cntPtr);
                        }
                        SystemMemory[link+1] = cntPtr;
                        SystemMemory[prePtr+1] = link;
                }
                return WhyDoIRecurse( newList,  next );
        }
        return newList;
}


 int ThisIsWhereItStart(int list) {
        int newList = list;
        list = nextField(list);
        SystemMemory[newList+1] = -1;
        return WhyDoIRecurse(newList, list);
}
```

Given that initially fMemorySystem looks like:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	-1	0	-1	4	6	6	-1	16	16	-1	12	12	14	9	8	18	18	-1	0

Also, the initial values for *myList*=12:

a. **(5%)** Draw the initial list content of *myList*. You must include both the **keyField**, and the **nextField** of each link in the link list.

b. **(8%)** What is the content of the list returned by **ThisISWhereItStart(12)**?

c. **(2%)** What does **ThisIsWhereItStart()** do?

d. **(6%)** Please derive an equation describing ***best*** case run time of **ThisISWhereItStart()**. This question asks, in the very best case, assuming all conditions are favorable, what is the run time of **ThisIsWhereItStart()** for a n element link list? *You can use the space besides the pseudo code to help show your analysis.*

e. **(6%)** Based on your equation in **Part (d)**, what is the best case runtime of **ThisIsWhereItStart()**?

f. (3%) What is the asymptotically tight lower bound (Ω) for the best case run time of **ThisIsWhereItStart()**?

Problem 3 (15%). My co-worker says sorting a link-list based data structure is inefficient and complicated to implement. As such, he will copy the link list into an array, sort the array and then copy the results back to the link list. Professor Kelvin Sung says, 2 copying procedures each with *O(n)* running time, and the *O(nlgn)* sorting time, my co-worker has an $O(n^3 \lg n)$ algorithm.

Assuming you have access to: ***CopyListToArray(inputList, outputArray)***, and, ***CopyArrayToList(inputArray, outputList)***, and ***SortArray(array)*** functions. Show the pseudo code of my co-work's sorting strategy and comment on Professor Sung's analysis.

Problem 4 (30%). Professor Kelvin Sung proposed the following *best* sorting algorithm known to humankind:

```
KelvinSort(A, startIndex, endIndex)
    // A is the array containing all the input numbers
    // startIndex is the starting position we will sort
    // endIndex is the end position

    if A[startIndex] > A[endIndex] then
            exchange A[startIndex] ←→ A[endIndex]
    if startIndex + 1 >= endIndex then
            return

    //*********************************************
    //      LOCATION A:
    // PrintArray(A, startIndex, endIndex);
    //    For Debugging purposes:
    //     prints out every element from startIndex to endIndex
    //*********************************************

    // now recursively call ourselves with smaller subproblem.
    offset= floor((endIndex-startIndex+1)/3)          // round down
    KelvinSort(A, startIndex, endIndex-offset)        // First two-thirds
    KelvinSort(A, startIndex+offset, endIndex)        // Last two-thirds
    KelvinSort(A, startIndex, endIndex-offset)        // First two-thirds again
```

a. **(10%)** Use *A[3 4 2 1]* to show that KelvinSort(A,1,n) correctly sorts the input array A[1..n]. You must include the tracing of each recursive calls to KelvinSort, clearly indicating the values of ***startIndex***, ***endIndex***, ***offset*** and ***the contents of the array A*** before and after each calls to KelvinSort.

b. **(10%)** Give a recurrence for the worst-case running time of KelvinSort and a tight asymptotic (Θ – *notation*) bound on the worst-case running time. Compare this with other sorting algorithms we have learned. Should Professor Kelvin Sung teach algorithm classes?

c. **(10%)** Trying to debug this algorithm, Kelvin un-commented the ***PrintArray()*** function located at **LOCATION A:**. Now, what is the tight asymptotic (Θ – *notation*) bound on the worst-case running time of KelvinSort?