CPSC 35 Midterm Exam

Fall 2007

10:30-11:20am, Wednesday 7 November
Closed book exam

NAME:_____

Problem	Max	Obtained
1	15	
2	20	
3	20	
4	20	
5	25	
Total	100	

15 points **Problem 1:** For each of the following three questions, justify your answer with a brief explanation of one or two sentences.

(a) Suppose we know an algorithm runs in $O(n)$ time. Is it also correct to say it runs in $O(n^2)$ time? Why or why not?

(b) True or False: Given a binary search tree with n elements, insertions and removals take at most $O(\log n)$ time? Explain.

(c) Two students Angelina and Brad are arguing about the run-time of their sorting algorithms. Angelina implemented a $O(n \log n)$ sort and claims it will always be faster than Brad's $O(n^2)$ sort. However, whenever they run experiments on $n < 100$ elements, Brad's implementation is occasionally faster. Only when $n > 100$ is Angelina's method faster. How it this possible?

20 points **Problem 2:** A portion of a doubly linked list implementation that stores integer elements is shown below. Write Java code to complete the `removeFirst(int x)` method below that removes the first occurrence of an element x from the list. You may assume that all real (non-sentinel) elements stored in the list have a value greater than 0. What is the run-time of your method? Sample code for a doubly linked node that stores an integer element is shown on page 7. You may tear off the page containing the `DNode` source code so you can examine code side by side.

```
/** Doubly linked list with nodes of type DNode storing strings. */
public class DList{
  protected int size;                    // number of elements
  protected DNode header, trailer;       // sentinels

  /** Constructor that creates an empty list */
  public DList() {
    size = 0;
    header = new DNode(-1, null, null); // create header
    trailer = new DNode(-1, header, null);      // create trailer
    header.setNext(trailer);    // make header and trailer point to each other
  }

  /** Returns the number of elements in the list */
  public int size() { return size; }

  /** Inserts the given element at the tail of the list */
  public void addLast(int x) {
    DNode w = trailer.getPrev();
    DNode v = new DNode(x, trailer, w);
    trailer.setPrev(v);
    w.setNext(v);
    size++;
  }

  /** Removes the first node in the list containing element x
      You may assume x is positive and thus cannot be confused with
      the header or trailer. */
  public void removeFirst(int x) {
     // COMPLETE THIS METHOD









  }
}
```

20 points **Problem 3:** A toll highway such as the Pennsylvania Turnpike contains a limited number of entrance/exit points that partition the road into a number of *legs*. A driver can get on the highway at the beginning of any leg and get off the highway at the end of the same leg or a later leg. Each leg carries a toll which must be paid upon exiting the highway. The toll is the sum of the tolls on each leg the driver rode on. A sample toll for a five leg toll road is shown below. The total toll for driving all of leg 0 is 0.50. To start on leg 1 and exit at the end of leg 2 would cost 0.25 + 0.60 or 0.85.

leg	0	1	2	3	4
cost	0.50	0.25	0.60	0.40	0.50
prefix sum	0.50	0.75	1.35	1.75	2.25

The code on page 8 shows two methods (feeCalc1 and feeCalc2) for computing the total toll for a number of trips. Each method repeatedly prompts the user for an entry leg and exit leg (one tripe) and displays the toll for that trip. The program quits when the user enters -1 for a start leg. For simplicity, the input/output methods getEntryLocation(), getExitLocation() and displayToll() are not shown in the code. Given an array `fees` that lists the cost of each leg, analyze the run-time of both `feeCalc1` and `feeCalc2`.

(a) What is the run time for each method if the user only enters one trip before exiting the program?

(b) What is the total run-time for each method if the user enters k trips before exiting? Express you answers in terms of n, the number of legs in the highway and k the number of different trips the user enters.

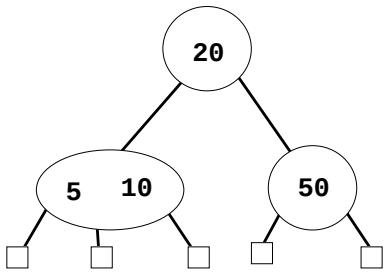
You may tear off any sample code from the end of this exam.

20 points **Problem 4:** Consider the 2-4 Search Tree.

(a) What are two properties that 2-4 Trees maintain after every update?

(b) What is the cost of find, insert, and remove, in a 2-4 Tree containing n elements?.

(c) For the 2-4 Tree shown below, draw the resulting trees after the following sequence of operations: insert(2), insert(15), remove(20). You should show three trees. The first is the result after insert(2), the second is the result after insert(2), insert(15), and the third is the final tree after all three operations.



25 points **Problem 5:** You have been hired as a consultant for a new digital music player. The player will manage a Music Library consisting of a number of Tunes. Each Tune has both an artist name and a song name. The player will support a simple interface that allows users to change the song they are listening to by navigating to the next song by the same artist, the previous song by the same artist, or the first song of the previous or next artist. For the purpose of this problem, artists and songs are arranged alphabetically. You have been asked to design an data structure to support these methods of navigating. An interface for both a single Tune and a MusicLibrary are given on page 9. Describe what data structure(s) you would use to implement this interface and give the run-time of each method for your implementation. Describe you solution in words. You do not need to write Java code or provide full implementation details. You do not need to discuss how Tunes are added/removed from the library, and you may assume that when the player is turned on, it starts playing the first song from the first artist. You may decide what the behavior of the player is if you request the previous artist of the first artist, or the next song of the last song.

Code for Problem 2:

```
/** Node of a doubly linked list of ints */
public class DNode {
  protected int element;        // int element stored by a node
  protected DNode next, prev;   // Pointers to next and previous nodes

  /** Constructor that creates a node with given fields */
  public DNode(int e, DNode p, DNode n) {
    element = e;
    prev = p;
    next = n;
  }

  /** Returns the element of this node */
  public int getElement() { return element; }

  /** Returns the previous node of this node */
  public DNode getPrev() { return prev; }

  /** Returns the next node of this node */
  public DNode getNext() { return next; }

  /** Sets the element of this node */
  public void setElement(int newElem) { element = newElem; }

  /** Sets the previous node of this node */
  public void setPrev(DNode newPrev) { prev = newPrev; }

  /** Sets the next node of this node */
  public void setNext(DNode newNext) { next = newNext; }
}
```

Code for Problem 3:

```
public class toll{
  public static void feeCalc1(double[] fees){
    int start, stop;
    double price; //total cost of trip
    while(true){
      start = getEntryLocation();
      if(start == -1){ return; } //stop loop
      stop = getExitLocation(start);
      //Method 1
      price  = 0.0;
      for(int i=start; i<=stop; i++){
        price += fees[i];
      }
      displayToll(start, stop, price); //show cost of trip
    }
  }

 public static void feeCalc2(double fees[]){
   int start, stop;
   double price;

   //Method 2a
   double[] sums = new double[fees.length];
   sums[0] = fees[0];
   for(int i=1; i<fees.length; i++){
     sums[i]=sums[i-1]+fees[i];
   }

   while(true){
     start = getEntryLocation();
     if(start == -1){ return; } //stop loop
     stop = getExitLocation(start);

     //Method2b
     if (start==0){ price = sums[stop]; }
     else{ price = sums[stop] - sums[start-1]; }

     displayToll(start, stop, price); //show cost of trip
   }
 }

 public static void main(String argv[]){
   double[] fees = {0.50, 0.25, 0.60, 0.40, 0.50};
   feeCalc1(fees);
   feeCalc2(fees);
 }
}
```

Code for Problem 5:

```
interface Tune {
  /** Return name of artist for this tune */
  String artist();

  /** Return song title for this tune */
  String song();
}

public interface MusicLibrary {
  /** Return next song of current artist */
  Tune nextSong();

  /** Return previous song of current artist */
  Tune prevSong();

  /** Return first song of next artist */
  Tune nextArtist();

  /** Return first song of previous artist */
  Tune prevArtist();
}
```