Midterm Exam

CS 311 Data Structures
Fall Quarter 2002

11:00am - 11:50am, Monday, October 28, 2002

THIS EXAM IS *CLOSED BOOK, CLOSED NOTES* .

Print your name neatly in the space provided below; print your name at the upper right corner of every page.

Name:

This booklet should have 7 written pages (including this one). If it does not, report this to the instructor.

Work efficiently. Do not get stuck too long on any one problem.

Write short, succinct answers.

Try to answer every question.

Problem	Max. Points	Your Score
1	11	
2	7	
3	14	
4	18	
Total	50	

1. (11 points) **Potpourri**

(a) Write a function (in pseudocode or english) that takes as input a string and returns TRUE if the string is of the form $w\#w^R$, where w is an alphanumeric string and w^R its reverse, and FALSE otherwise. For example, on input "abc#cba" your function should return TRUE, whereas on input "aab#abb" it should return FALSE.

You must not use any functions from `string.h`

(b) Using big-oh notation, give the running time of the following piece of code. Briefly justify your answer.

```
for (i = 1; i <= n; i++)
   for (j = 1; j <= n/2; j++)
      for (k = 1; k <= j; k++)
          cout << "Ha ha\n";
```

(c) Which data structure would you use in each of the following situations?

- You want to store a reverse telephone directory (i.e. the search key is the phone number and the stored data is the name of a person) so that a search takes constant time. Hint : the size of the keys is constant.
- The company policy is that when it needs to lay off personnel, the last person to be hired would be the first to go. What is the best data structure for storing the employee records?
- You want to write a simulation of a busy intersection. What structure will you use to store the cars at the traffic lights?
- What structure would you use to store the dictionary used by the spell checker of an editor?

2. (7 points) **Sorting**

(a) Insertion sort can be expressed as a recursive procedure as follows: In order to sort array A[1..n], we recursively sort array A[1..n-1] and then insert A[n] into the sorted array A[1..n-1].

Give an equation that describes the overall running time of this algorithm on an input array of size n, in terms of the running time on smaller input.

```
T(n) =
```

(b) Recall the counting sort algorithm, where

- n is the number of elements to be sorted
- each element is an integer in the range 1 to k
- A is the input array, of size n
- B is the final sorted output array
- C is the intermediate array of size k

```
1: for i = 1 to k
2:     C[i] = 0
3: for i = 1 to n
4:     C[A[i]] = C[A[i]] + 1
5: for i = 2 to k
6:     C[i] = C[i] + C[i-1]
7: for i = n downto 1
8:     B[C[A[i]]] = A[i]
9:     C[A[i]] = C[A[i]] - 1
```

Suppose that the for-loop on line 7 above, is modified as follows:

```
7: for i = 1 to n
```

Will the algorithm still sort the input? If no, explain why not. If yes, what is the effect of this modification to the algorithm and the output?

(c) When using mergesort to sort an array, do the recursive calls to mergesort depend on the number of elements in the array, the values of the elements or both? Briefly explain.

3. (14 points) **Linked Lists**

(a) A student wrote the InsertSorted() function shown below for the following List class:

```
class ListNode {
private:
        int element;
        ListNode *next;
        ListNode *prev;
public:
        ...
};
```

```
class List {
private:
        ListNode *head;
        int size;
public:
        ...
};
```

The function InsertSorted() inserts an element in its sorted position in a doubly-linked list (where the head node's prev pointer and the tail node's next pointer are both null). It iterates through the entire list and tries to find the position of the new element using the '<' operator.

```
// InsertSorted
// - parameters : newElem - an integer element
// - inserts newElem according to its sorted position
void List::InsertSorted(int newElem) {

    ListNode *temp = new ListNode(newElem);
    if (head == NULL) {
        size = 1;
        head = temp;
    }
    else {
        ListNode *current = head;
        while (current != NULL) {
            if (current->element < newElem)
                current = current->next;
            else
                break;
        }
        temp->next = current;
        temp->prev = current->prev;
        if (temp->prev != NULL)
            temp->prev->next = temp;
        temp->next->prev = temp;
        size++;
    }
}
```

This function has no syntax errors, but it does have logical errors that may cause problems in certain situations. Identify those errors and explain how you would modify the code shown above so that it works correctly.

(b) In a singly-linked list with no dummy node, which (if any) of the following operations can be implemented so that it will always run in worst-case constant time? The list has a `head` and a `current` pointer only.

```
InsertBefore(x) // insert x before current pointer

Find(x)

Remove() // remove node at current pointer
```

4. (18 points) **Trees**

(a) Below is a post-order traversal of a binary search tree. Draw the tree.

```
12 9 24 17 40 49 38
```

(b) If we have a binary search tree that stores integers, which traversal prints the integers in increasing order?

(c) In our definition of binary search trees there are no nodes with duplicate keys. In practice however this is not always the case. If we allow duplicates we have a choice between:

(a) storing items that duplicate the root on either side of the root

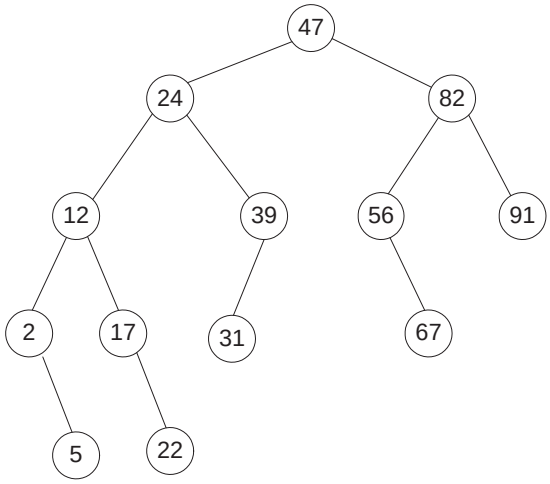
(b) storing items that duplicate the root either always on the left or always on the right subtree of the root.

What is the main disadvantage of option (a)?

What is the main disadvantage of option (b)?

(d) Consider the alphabet {A, B, D} and the set of words {A, ADD, BAD, DAB, DAD}. Draw a trie that stores these words.

(e) Consider the following AVL tree:



i. Show the resulting tree after inserting 23 to the given tree. If part of the tree remains the same, you may indicate so without redrawing it.

ii. Show the resulting tree after deleting 91 from the original tree. If part of the tree remains the same, you may indicate so without redrawing it.

SCRATCH PAPER