Assignment 5

Documents Due: 12:00 noon, Wednesday, February 27, 2002

Code Due: (*no code due this week*)

This assignment will give you practice analyzing the running time of algorithms.

Preliminaries

As with all assignments in this course, part of your grade will be based on the way your written work is presented. If your work is not neat and easy to read, you may lose points. See the *Homework Policies* handout for details.

Before You Begin

Make sure you are up to date on all the reading for the course. In particular, you should have read Chapter 6 of Weiss on complexity analysis.

Written Component

When answering the following questions, explain your answer clearly, and show your work for significant intermediate steps of computations. For example, when analyzing the running time of a code fragment, include the running time for each individual line.

It is sufficient to provide an asymptotic ("big-O") analysis unless the question indicates otherwise, or when you believe that constants and low-order terms have a significant impact on the answer. The analysis you make should follow the broad form and layout seen in class, but you may skip writing exact formulas (sigma-sums, closed-form expressions) in favor of big-O notation when you feel the situation is simple enough to be confident in doing so.

The code fragments are only fragments, and they may ignore certain niceties of C++ style. Don't pick at the details. Treat them as particularly explicit pseudocode.

W1. Provide a big-O analysis of the running times for the code fragments given in Weiss

   i. Problem 6.15 (a)

   ii. Problem 6.16 (a)

   For both problems, you may assume that `sum` was declared as

   ```
   int sum = 0;
   ```

   before each code fragment. Give your analysis in terms of the loop-limit variable, `n`.

W2. Suppose that we define a linked-list class with the following data fields:

```
class Node {
    ⋮
private:
    long value;
    Node* next;
};

class List {
    ⋮
private:
    Node* head;
};
```

(For this problem, assume that any single call to **new**, **new** [], **delete**, or **delete** [] can be performed in $O(1)$ time.)

(a) How long (in big-O terms) do the following operations take, as a function of the length of the list, n? You may not assume any changes in the *Node* or *List* data structures.

i. Making a copy of the list
ii. Adding a value to the start of the list
iii. Adding a value to the end of the list
iv. Removing the first value from the list
v. Removing the last value from the list
vi. Determining whether the list contains some value V

(b) Suppose that we modify the classes to store C++ *string*s of maximum length m characters, rather than *long*s. For each of the parts i to vi above, indicate whether this change would affect the running time, and, if so, give the new running times (again, in big-O terms).

Notice that each list should contain its own private copy of the strings stored in it. Lists should not share strings with other lists or with other parts of the code.

You should assume that when a string is copied or assigned, a complete copy is made of that string. (Interestingly, this assumption is not true for all implementations of the C++ library, but we'll ignore that fact for this assignment.) You should also assume that the *string* type has been implemented in an efficient and effective fashion (i.e., there are no $O(n^2)$ implementations of things that can be done in $O(n)$ time).

W3. Consider the three functions shown below, which use the *List* and *Node* data structures from question 2:

```
// returns the length of the list
int List::length()
{
    Node *current = head;
    int output = 0;

    while (current != NULL) {
        output++;
        current = current->next;
    }

    return output;
}

// returns the nth value in the list
long List::nth(int n)
{
    if (n >= length() || n < 0)
        error("List::nth out of range position");

    Node *current = head;
    for (i = 0; i < n; i++) {
        current = current->next;
    }

    return current->value;
}

// prints all the values in the list
void List::print()
{
    for (i = 0; i < length(); i++) {
        cout << nth(i) << endl;
    }
}
```

Analyze the running times of these three functions. For each function, show how it could be recoded (if possible) so as to improve its asymptotic running time, and provide an analysis of the new running time. If no improvement is possible, explain why. You are only allowed to modify the executable code; you may not modify or add any fields to the *Node* and *List* data structures.

W4. If n and m depend on the input, what is the complexity of the following code fragment?

```
int timeWaster = 0;
for (int i = 1; i < n; ++i)
    for (int j = 0; j < n; ++j)
        if (j % i == 0)
            for (int k = 0; k < n; ++k)
                ++timeWaster;
        else
            for (int k = 0; k < m; ++k)
                ++timeWaster;
```

Warning! This problem is fairly tricky!