Assignment 4

`HW4-Answers.txt` Due: 11:59 PM, Wednesday, October 11, 2006

This assignment gives you practice analyzing the running time of algorithms.

Setup

Before you meet with your partner, 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.

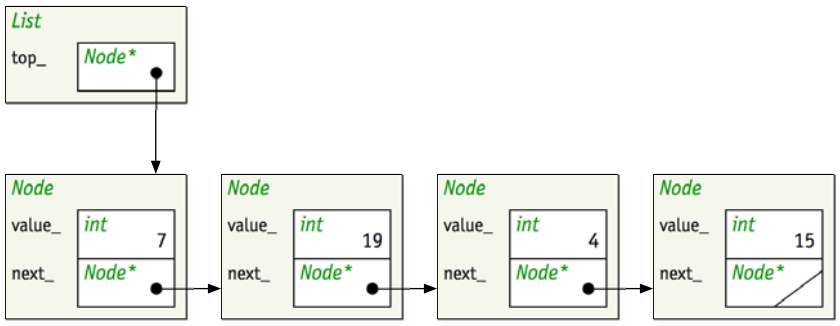
Written Component

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, indicate which lines dominate the execution time and include their running time. It is *not* sufficient to simply give an answer with no working or explanation, even if you think the answer is "obvious".

Remember that when determining the asymptotic complexity ("big-O") of a function, you do not have to calculate *exactly* how many times a statement is executed—such exactitude gets lost anyway when you convert to O-notation. You may simplify mathematical expressions in ways that change their value slightly, but not significantly for the analysis. You may also rewrite the code in ways that makes it easier to analyze but does not alter the fundamentals of the algorithm.

If you express your answer in terms of a variable n, be sure that you make it completely obvious (either from the problem statement or from your own explanation) what n is measuring.

Example diagram of data stucture used in Questions 2 and 3.



W1. Provide a big-O analysis of the running times for the code fragments below, assuming that sum has been declared as an *int* and initialized to zero. Give your analysis in terms of the loop-limit variable, n.

(a)
```
for (int i = 0; i < n; ++i)
    ++sum;
```

(b)
```
for (int i = 0; i < n; i += 2)
    ++sum;
```

(c)
```
for (int i = 0; i < n; i *= 2)
    ++sum;
```

(d)
```
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n; ++j)
        ++sum;
```

(e)
```
for (int i = 0; i < n; ++i)
    ++sum;
for (int j = 0; j < n; ++j)
    ++sum;
```

(f)
```
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n * n; ++j)
        ++sum;
```

(g)
```
for (int i = 0; i < n; ++i)
    for (int j = 0; j < i; ++j)
        ++sum;
```

(h)
```
for (int i = 0; i < n; ++i)
    for (int j = 0; j < n * n; ++j)
        for (int k = 0; k < j; ++k)
            ++sum;
```

(i)
```
for (int i = 1; i < n; ++i)
    for (int j = 0; j < i * i; ++j)
        if (j % i == 0)
            for (int k = 0; k < j; ++k)
                ++sum;
```

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

```
class List {
public:
    ⋮
private:
    struct Node {
        int value_;
        Node* next_;
    };

    Node* head_;
};
```

Note that a struct is like a class, except that all data members are public by default—structs are usually used for "plain old data" (i.e., structs usually have data members, but no member functions). The above code is an example of nested classes. The declaration of the *Node* type means the same as it would if it were declared outside the class, except that it is a private type used by *List*.

Also, 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 *int*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.[1] You should also assume that the *string* type has been implemented in an efficient and effective fashion (i.e., there are no $\Theta(n^2)$ implementations of things that can be done in $\Theta(n)$ time).

1. Interestingly, this assumption is not true for all implementations of the C++ library, but we'll ignore that fact for this assignment.

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
int List::nth(int n)
{
    if (n >= length() || n < 0)
        abortWithError("List::nth:  Position out of range!");

    Node* current = head_;
    for (int i = 0; i < n; ++i) {
        current = current->next_;
    }

    return current->value_;
}

// prints all the values in the list
ostream& List::print(ostream& out)
{
    for (int i = 0; i < length(); ++i) {
        out << nth(i) << endl;
    }
    return out;
}
```

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!