Lab 5: Algorithm Efficiency w/ Big-O

By Dr. Yingwu Zhu, Seattle University

1. Goals

This lab aims to help you understand algorithm efficiency/complexity using Big-O notations. The algorithmic complexity of an algorithm is a measure of how many instructions (or basic operations) an algorithm requires as a function of the input size n. The simplifying analysis allows us to focus on the most executed instruction. The Big-O notation is used to describe the upper bounds of an algorithm's complexity. The big-O value of an algorithm can be determined by first generating an equation that determines the number of steps an algorithm will require to complete and then by reducing that equation to its most significant term.

2. Exercises

In this exercise you will evaluate four programs to try and determine the Big-O complexity of each. You will not have access to the C++ source code for the applications, but only the compiled executables. The four applications do the same thing, but each has a different Big-O complexity. The complexities are O(n), O(n log n), $O(n^2)$, and $O(n^3)$. It will be your job to match the programs with the correct complexity.

Obtaining the executables

You will download the executables by executing the command in your lab5 directory:

/home/fac/testzhuy/CPSC250/lab/download

The four programs are: *max1*, *max2*, *max3*, and *max4*.

File Input

All four of the programs do the same thing; they read a file consisting of the number of values, followed by a sequence of positive and negative integers and find the consecutive subsequence with the highest sum.

For example in the series: -5 *3 6 -2 7* -5 3 -2 the italicized red-color values form the subsequence with the highest value.

To determine the complexity of the four test programs (they all do the same thing, but they are not all equally efficient) you will need to test them on sequences of different

sizes. The easiest way to do this is to automatically generate files of random positive and negative integers for the programs to work on.

A program to create such files is below. We have not covered everything that is used in the program, but it will work if typed in exactly as shown.

```
/* genRandom.cpp
   Generate random values.
*/
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>


using namespace std;
int main()
{
	const int MAX_ABSOLUTE_VALUE = 1000;
	// Seed the random number generator, so that it will
	// output different values each time it's run.
	srand(time(0));
	cout << "Please enter a save filename (no spaces): ";
	string outputFilename;
	cin >> outputFilename;
	ofstream outstream(outputFilename.c_str(), ios::out);
	if (outstream.fail())
	{
		cout << "Unable to open " << outputFilename << endl;
	}
	else
	{
		int numInts;
		cout << "How many integers would you like? ";
		cin >> numInts;
		outstream << numInts << endl; // store in output file
		for (int i = 0; i < numInts; ++i)
		{
			// Get a random number, and scale
			// it to [0, MAX_ABSOLUTE_VALUE * 2)
			int randomInteger = rand() % (MAX_ABSOLUTE_VALUE * 2);
			// Shift number to [-MAX_ABSOLUTE_VALUE, MAX_ABSOLUTE_VALUE)
			randomInteger -= MAX_ABSOLUTE_VALUE;
			outstream << randomInteger << endl;
		}
	}
	outstream.close();
	return 0;
}
```

Testing and Timing the Program

The four test programs take the name of a file to read as an argument when run, meaning the data file name simply follows the program name when run.

To measure the time a program takes to evaluate a file, you will use the Unix/Linux time command. For example,

time ./max4 t1000.dat

This times and runs ***max4*** on the file ***t1000.dat***. You will get output similar to that below:

```
The maximum segment sum is 12993
real  0m0.047s
user 0m0.010s
sys   0m0.020s
```

The user time (0.010 seconds in this example) is the time the computer spent running the program, and is the one to which you should pay attention.

Turn In

Come up with a method of determining the Big-O complexity of each program. Remember that there is one each of O(n), O(n^2), O(nlogn), and O(n^3). Turn in your categorizations of the four programs, and a one or two paragraph description (or graphs) explaining how you came to your conclusions.

Due: 10/21/2011, Friday, 9:20AM
Submission: Hardcopy