UNBC Laboratory Assignment
Computer Science 200—Fall 2003

Asymptotics and Timing

Purpose:

To verify Θ-behaviour of programs by timing their performance and plotting the results.

Estimated Time:

Six hours. This laboratory exercise has not been assigned before, and so figure given is merely an estimate. It is your responsibility to find enough time to complete the assignment.

Keep track of the time spent on this assignment and complete the attached workload sheet for a 5% bonus.

Due Date:

This assignment is due *Friday, 19 September, 2003* at the beginning of class.

Assignment:

Create a `StopWatch` class. You are going to need the `StopWatch` class in later assignments, so make sure that you create a separate `StopWatch.h` file, `StopWatch.o` file, *etc.*

The `StopWatch` class should behave like a mechanical three-button stop-watch. That is, the interface should look something like:

```
class StopWatch
{public:
    StopWatch() ;
    // The three buttons.        // and queries
    StopWatch& start() ;         double elapsed() const ;
    StopWatch& stop() ;          bool is_running() const ;
    StopWatch& reset() ;
private: ...} ;
```

You must write your `StopWatch` class so that a `StopWatch` object is capable of being stopped and started and reset in arbitrary sequences. It must always return accurate information from the query functions.

There is more than one notion of elapsed time for a multi-process operating system (such as UNIX) because because single processes don't get all of the processor. What you want to measure

is the elapsed CPU time, not the elapsed wall-clock time. Read the **man**-pages for `clock` and `times` for information on how to get elapsed time information.

⇒ Write a short driver program that fully tests the functionality of the stopwatch.

The parentheses problem

Write a function with signature `std::vector<std::string> allParens(int n)` that returns all of the legal sequences of n left parentheses and n right parentheses.

⇒ Write a driver to test that program works up to $n = 4$ by printing all of the solutions.

⇒ Write a driver program that produces timing data for your function. Time your algorithm up to about $n = 12$. Print the number of solutions for each n, but do not print the actual solutions. Show the compilation and running of your driver program.

⇒ Plot the data from your driver program. What do you think the asymptotic complexity of your function is?

Fibonacci algorithms

Write two recursive programs that compute Fibonacci numbers ($F_0 = F_1 = 1$, $F_{n+2} = F_{n+1} + F_n$). One program should directly use the recursive definition. The other should also use recursion, but more efficiently.

Note that you probably want your program to produce `double` rather than `long` results since F_{46} is too large to store in a 32 bit integer.

⇒ Write a driver to test that each function works.

⇒ Write a driver program that produces timing data for each of the functions. Show the compilation and running of your driver program.

⇒ Plot the data from your driver program.

⇒ Attach a sheet to the material you hand in entitled *Analysis of the Fibonacci timings*. Discuss your actual timings and suggest what the Θ-behaviour might be. For both functions attempt to estimate the time required to compute F_{50}.