G22.1170: Fundamental Algorithms
Problem Set 0
(Due Monday, Oct 2 2000)

The first three problems require knowledge of discrete mathematics, and the last two involve the use of PASCAL and UNIX. The problems are meant to help you review the material, and should not require too much time. For each problem involving programming, please turn in a printed copy of the file containing the program and its output on some test runs. Write on it, attach to it, or include as comments a *short* prose paragraph describing what you have done.

Problem 0.1 Prove the following identities for all positive n.

$$\begin{array}{lll} \text{(a)} & 1+2+3+\cdots+n & =\frac{n(n+1)}{2} \\ \text{(b)} & 1^3+2^3+3^3+\cdots+n^3 & =(1+2+3+\cdots+n)^2 \end{array}$$

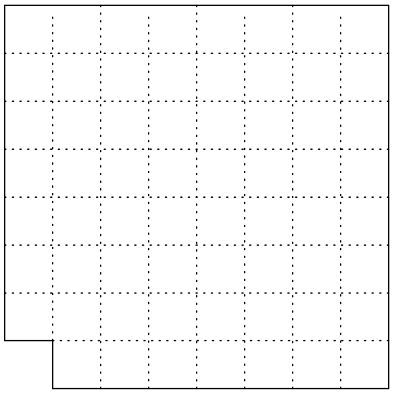
Problem 0.2 For all $m > 0$, we define $H(m)$ as follows:

$$\begin{aligned} H(m) &= 1+\tfrac{1}{2}+\tfrac{1}{3}+\cdots+\tfrac{1}{m} \\ &= \textstyle\sum_{i=1}^{m}\frac{1}{i} \end{aligned}$$

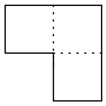
Show that

$$1+\frac{n}{2} \leq H(2^n) \leq 1+n$$

Problem 0.3 (Optional) You are given a $2^N \times 2^N$ checkerboard with one of the corner squares missing. Here N is some positive integer. Show that this mutilated checkerboard can be completely tiled by a set of L-trominoes. (L-trominoes are L-shaped tiles made of three squares. The figure 1 shows a mutilated checkerboard and an L-tromino.)



Mutilated Checker Board
($N = 3$)



L-tromino

Figure 1: L-tromino

Problem 0.4 (Optional) The Juggernaut-Harakiri, Inc. has perfected the art of suicide. In particular, they have come up with an algorithm, which, given N people (numbered 1 through N) and one gun, ensures that all but one (numbered $J(N)$) of them end up dead. They proceed as follows: Arrange the N members in a circle such that $(j+1) \bmod N$ and $(j-1) \bmod N$ are, respectively, the left and right neighbours of j. Start with the gun in 1's hand. At any instant, a member with the gun shoots the living member immedaitely to his left, and passes the gun to the next living member to his left. This continues until one person $J(N)$ survives.

Can you sketch a method to compute $J(N)$ efficiently?

Problem 0.5 Leonardo da Pisa filio Bonaccio (*alias* Fibonacci) has defined the following sequence:

$$\begin{aligned} F(1) &= 1, \\ F(2) &= 1, \\ F(i) &= F(i-1) + F(i-2), \qquad \textit{for all } i > 2. \end{aligned}$$

For all other values of i, $F(i) = 0$.

Here are two different PASCAL programs to compute the Fibonacci Sequence:

Program.1

```
program fib1(input,output);
var     N: integer;

function fibonacci(N: integer): integer;
begin
        if N < 1 then fibonacci := 0
        else if N <= 2 then fibonacci := 1
        else fibonacci := fibonacci(N-1) + fibonacci(N-2)
end;(*fibonacci*)

begin
        while not eof do begin
                readln(N);
                writeln(fibonacci(N))
        end
end.
```

Program.2

```
program fib2(input,output);
var     N: integer;

function fibonacci(N: integer): integer;
var     A, B, I: integer;
begin
        if N < 1 then fibonacci := 0
        else if N <= 2 then fibonacci := 1
        else begin
                A := 1; B := 1;
                for I := 1 to (N div 2) - 1 do begin
                        A := A + B; B := B + A;
                end;
                if odd(N) then B := B + A;
                fibonacci := B
        end
end;(*fibonacci*)

begin
```

```
        while not eof do begin
                readln(N);
                writeln(fibonacci(N))
        end
end.
```

Your job is to type these two programs into two files (say `fib1.p` and `fib2.p`), compile and run them on a set of input values. You should go through the documents describing C-Shell and the display editor Vi. These will help you to understand how to login, organize and edit your files. After you have entered the programs in the files, compile them by typing

```
% pc <filename>
```

The `<filename>`, in this case, is either `fib1.p` or `fib2.p`. This will produce a file `a.out` that can be run by simply typing

```
% a.out
```

Your next task is to *profile* the programs in order to get some idea about their *time complexity*. To do this, you have to first compile the programs with the `-p` flag, for example

```
% pc -p fib1.p
```

When you run the program (by typing `a.out`, as before), the program will leave a file called `mon.out` of data to be interpreted by the profiling program `prof`. Now if you type

```
% prof
```

you will get a table showing the amount of time spent in different portions of the program. The heading of the table will look something like this:

```
%time   cumsecs #call   ms/call name
```

For each routine (given by `name`), the table shows (*i*) `%time`: the percentage of time spent executing the routine, (*ii*) `cumsecs`: a running sum of the amount of time in seconds accounted for by this routine and those above it, (*iii*) `#calls`: the number of time the routine is called, and (*iv*) `ms/call`: the number of milliseconds per call.

Using `prof`, find out how much time `fib1` and `fib2` spend for each input value in the range of 1..25. Plot the amount of time spent against the input values.

Warning: The time taken by (`fib2`) may be so small that no significant digit will appear in the table created by `prof`. One simple way to remedy this problem is to introduce a `for` loop in the main part of the program that will call the function `fibonacci` several times (say 100 times). The time spent in one single execution of the program can be determined by appropriately scaling the readings.

Problem 0.6 a. Assume you are given the type declaration

```
type digitset = set of 0..9;
```

Write a PASCAL procedure `symdiff` that takes two input arguments `X` and `Y` of type `digitset` and writes to a `var` paramter named `Z`, also of type `digitset`. The parameter `Z` is to receive the symmetric difference of the two sets `X` and `Y`. An element is in the result set (`Z`), if it is in either input set (`X` or `Y`) but not in both sets. You may of course use the built-in operations on sets provided by PASCAL, such as intersection (`*`) and union (`+`).

Use your program to find the symmetric differences of the following pairs of sets

1. $\{0, 2, 4, 6, 8\}, \{1, 3, 5, 7, 9\}$
2. $\{0, 2, 4, 6, 8\}, \{0, 2, 4, 6, 8\}$
3. $\{0, 1, 2, 3, 4, 5\}, \{4, 5, 6, 7, 8, 9\}$

b. Again assume you are given the type declaration

```
type digitset = set of 0..9;
```

Write a PASCAL procedure `powerset` that takes an argument `X` of type `digitset` and writes to a `var` parameter named `Z`. One would like `Z` to be of type `set of digitset`, but PASCAL does not allow that, so let `Z` be presented as an array of type `array[0..1023] of digitset`. (Because the input set has at most ten elements, the result will be a set of at most $2^{10} = 1024$ elements.) If the argument `X` has n elements ($0 \leq n \leq 10$), then all subsets of that set should be stored into the first 2^n elements of the array `Z`; you need not write into any elements after the first 2^n of them.

Use your program to find the power sets of the following of sets

1. $\emptyset$, the empty set.
2. $\{0, 2, 4, 6, 8\}$
3. $\{0, 1, 2, 3, 4, 5\}$