TEL AVIV UNIVERSITY
Department of Computer Science
0368.1105 – Extended Introduction to Computer Science
Fall Semester, 2000/2001

Homework 2, Nov 5, 2000

This homework is due by 18:00 on Sunday, Nov 12, at the hand-in boxes near room 005 of the Schreiber building. Write your name, id and the **Tirgul group number** on top of the first page. Keep a copy of the homework. Please answer all questions that require you to write a procedure by submitting the procedure itself, and the result of testing your code on the computer (a printout). Try to pick at least one meaningful example of an execution to show that the procedure works. There is a lot of work so we suggest that you start working right away.

Evaluation

1. Consider the following definition:

```
(define what
  (lambda (a b c)
    (if (> a (+ 1 c))
        a
        (+ b c))))
```

Use the substitution model to describe the evaluation of the following expression. Be careful when substituting into special forms ...

```
⟹ (what 4 5 6)
;value:
```

2. Consider the following definitions:

```
(define x 2)
(define y 4)
(define z 5)
(define (f x z)
  (define (g y z)
    (define (h x y)
       (+ x y))
     (+ (h y z) (h x z)))
  (+ ( g x y) (g z z )))
```

What will be the result of each of the following expressions? Explain using the substitution model.

```
⟹ (f x y)
;value:
⟹ (f z x)
;value:
⟹ (f y z)
;value:
```

3. What do (`count1 10`) and (`count2 10`) print? Explain. What type of processes are generated?

```
(define (our-print x)
   (newline) (display x) x)

(define (count1 x)
   (cond ((= x 0) 0)
         (else (our-print x)
               (count1 (-1+ x)))))

(define (count2 x)
   (cond ((= x 0) 0)
         (else (count2 (-1+ x))
         (our-print x))))
```

(We defined `our-print` because we want to make sure that it returns its argument. Different implementations of SCHEME vary in the value returned by the printing procedures.)

4. We define a procedure `new-if` as follows.

```
(define (new-if x y z)
   (cond (x y)
         (else z)))
```

and we redefine the factorial program to use `new-if` instead of `if` as follows.

```
(define (new-fact n)
   (new-if (= n 1)
      1
      (* n (new-fact (- n 1)))))
```

What would be the values of the following expressions? Explain!

```
1)  (new-if (= 2 3) 0 5),
2)  (new-fact 5).
```

Procedures

5. Write a procedure `signum` that takes a number `x` as its argument and returns -1 if the `x` is negative, `1` if `x` is positive, and `0` if `x` is `0`.

6. Write a procedure `true-false` that takes one argument and returns 0 if the argument is true and 1 if it is false, e.g.,

```
==> (true-false (> 3 2))
;value: 0
```

7. Write a procedure `max-power` that is given two numbers, `n` and `b`, and finds the largest power of `b` which is less than `n`. For example

```
(max-power 2 30)            ⟹ 16
```

Determine the order of growth of the time and space complexities of `max-power`.

8. Write a procedure `sum-digits` that receives a positive integer `n` and returns the sum of the base-10 digits of `n`. For example, (`sum-digits` 1024) should return 7. (SCHEME should provide the primitive procedures `remainder` and `quotient` that may be helpful here. If they are not available under your interpreter, please implement them.). Does your procedure produce an iterative or a recursive process? Explain.

9. The *digital root* of a number is obtained by summing its digits, summing the digits of the result, etc., until the result is less than 10. For example, the digital root of 561 is 3. Write a procedure `digital-root` that receives a natural number `n` and computes its digital root. Does your procedure produce an iterative or a recursive process? Explain.

10. Write a procedure `inverse` that receives an integer `x` and returns the number obtained from `x` by reversing the order of its base-10 digits. For example:

```
> (inverse 3456)
  6543
> (inverse -3448)
  -8443
> (inverse 50000)
  5
```

11. Use the following definition of the binomial coefficients for non negative integer n and k

$$\binom{n}{k} = \begin{cases} 0 & \text{if } k > n, \\ 1 & \text{if } k = 0 \text{ or } k = n, \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{otherwise.} \end{cases}$$

to implement a procedure `(binom-coef n k)` which calculates $\binom{n}{k}$. What type of a process does your procedure generate?

12. The function $f(n)$ is defined as follows: $f(0) = 1$, $f(1) = -1$ and $f(n) = 2f(n-2) - f(n-1)$, for $n \geq 2$.

(a) Write a procedure (`f1 n`) that uses tree recursion to compute $f(n)$. What is the time complexity of your procedure?

(b) Write a procedure (`f2 n`) that computes $f(n)$ in *linear* time.

13. Write a procedure (`ways-to-factor n m`) that computes the number of different ways in which n can be represented as a product of integers, each of which is at least m. (Note that a factor may appear more than once in the representation and the order of the factors does not matter. Assume that n and m are integers.) For example,

```
(ways-to-factor 4 3)  ⟹ 0
(ways-to-factor 8 2)  ⟹ 2  ; 2*2*2 and 2*4
```

Time and Space Orders of Growth

The following is a short reminder of the ideas we talked about during lecture No. 3.

Goal: For a problem of size n, we want to characterize $R(n)$ which is a function that quantifies how much of some resource is needed to solve the problem. Typically, we care about time $T(n)$ and space $S(n)$ requirements. Take for example $T(n)$. Approaches to describing resource usage include:

- An *absolute* measure. Exact count of resource use, e.g.,

$$T(n) = 5n^2 + 3n\log_3 n + 5$$

- An *asymptotic* measure.

  (a) Approximate the behavior for large n. For large n, we see that

  $$5n^2 \gg 3n\log_3 n \gg 5$$

  so we can approximate the time requirements for large n as

  $$T(n) \simeq 5n^2.$$

  (b) Ignore the constants of proportionality:

  $$T(n) \quad \simeq \quad k\,n^2$$

  We can use "theta" notation to understand the *order* of the growth. If there exist constants k_1 and k_2, independent of n, such that $k_1 f(n) \leq R(n) \leq k_2 f(n)$ for large n, then $R(n) = \Theta(f(n))$. So for our T above we have:

  $$T(n) \quad = \quad \Theta(n^2)$$

- A *bounded* measure. We may only know a maximum bound on resource use. If $R(n) \leq kf(n)$ for some k and for large n, then we can say $R(n) = O(f(n))$. We see that $O(\cdot)$ is a weaker statement than $\Theta(\cdot)$:

  $$T(n) = 5n^2 \quad \text{so that} \quad T(n) = O(n^3) \quad \text{is true.}$$

  Thus $O(\cdot)$ gives some (perhaps overstated) upper bound, while $\Theta(\cdot)$ gives us the asymptotic behavior of the process or algorithm.

Choosing an Algorithm

When choosing an algorithm to solve a particular problem, the actual problem size n and constant k may indeed matter. Suppose we have two algorithms with time behavior T_1 and T_2, respectively:

$$\begin{aligned} T_1(n) &= 2 \times 10^6 n \\ T_2(n) &= 2n^2 \end{aligned}$$

For what n would you choose algorithm 2 over algorithm 1?

Some Typical Orders of Growth

Typical orders of growth, and the kinds of problems that often give rise to these growths:

- $\Theta(1)$
  Constant growth. In time, this corresponds to simple computations (e.g. `(square x)`). Iterative algorithms use constant space.
- $\Theta(\log n)$
  Logarithmic growth. Logarithmic time algorithms often arise when a big problem can be transformed into a smaller problem via reduction by some constant fraction.
- $\Theta(n)$
  Linear growth. Linear time algorithms arise, for example, when some given amount of processing is needed for each of n input elements.
- $\Theta(n \log n)$
  "n log n" growth. Commonly occurs when a big problem is broken into smaller subproblems, solving them independently, then combining the solutions.
- $\Theta(n^2)$
  Quadratic growth. Quadratic time algorithms often occur when all pairs of data items must be processed. Such algorithms are only practical for relatively small problems.
- $\Theta(n^3)$
  Cubic growth. Commonly occurs when all triplets of data items must be processed. Only practical for relatively small problems.
- $\Theta(2^n)$
  Exponential growth. Commonly arises in "brute force" solutions to problems (e.g. when all other elements must be considered as each individual element is processed). Impractical for all but the smallest problems.

Recursion, Iteration, and Orders of Growth

14. For the following functions R find the simplest function f for which $R(n) = \Theta(f(n))$.

a. $R(n) = 8$

b. $R(n) = 5n^3 + 1000$

c. $R(n) = 400 \log n + 2n^4 - 10n^3 + 7n - 5$

d. $R(n) = 2^{3n-5} + 2^{n/3} + 1000$

15. Consider the procedure:

```
(define mystery
  (lambda (n)
    (if (= n 0)
        0
        (+ n (mystery (- n 1))))))
```

(a) Which function does mystery compute?

(b) Determine the orders of growth of the time and space complexities of `mystery`.

(c) Does `mystery` generate an **iterative** or a **recursive** process?

(d) Write a procedure `mystery2`, which computes the same function as `mystery`, but generates the other kind of process.

(e) Determine the orders of growth of the time and space complexities of `mystery2`.

16. Determine the orders of growth of the time and space complexities of `mul1`:

```
(define mul1
  (lambda (n m)
    (if (= n 0)
        0
        (+ m (mul1 (- n 1) m)))))
```

17. Determine the orders of growth of the time and space complexities of `mul2`:

```
(define (mul2 n m)
  (define (iter count ans)
    (if (= count 0)
        ans
        (iter (-1+ count) (+ m ans))))
  (iter n 0))
```