Computer Science 3711
Winter 2003

Midterm Exam

March 6, 2003

Instructor:
T. Wareham

NAME: _____ STUDENT ID #: _____

- This exam is out of 60 marks.
- This exam has 7 pages (including this cover page). There are 5 questions, most of which have multiple parts.
- Please answer all questions in the space provided on this exam; if you find it necessary to continue an answer on the back of a sheet of paper, that is fine, but please make a note on the front side, *e.g.*, "answer cont'd on back".

Question		Mark
1.	9	
2.	12	
3.	9	
4.	18	
5.	12	
	60	

1. (9 marks) Circle the letters associated with the appropriate answers in the following multiple choice questions. Note that some of these questions may have more than one answer whose letter needs to be circled.

(a) (3 marks) Which of the following is true of the function $T(n) = 2^{n+1}$?

a) $T(n)$ is $\Omega(2^{n+3})$ b) $T(n)$ is $O(n^{1000})$ c) $T(n)$ is $\Theta(2^{n-10})$

(b) (3 marks) Which of the following is true of the function $T(n) = 9^{\log_3 n}$?

a) $T(n)$ is $\Omega(1.2^n)$ b) $T(n)$ is $O(3^{\log_9 n})$ c) $T(n)$ is $\Theta(n^2)$

(c) (3 marks) Which of the following is true of the function $T(n) = n^2$?

a) $T(n)$ is $\emptyset((n-1)^2)$ b) $T(n)$ is $\Omega(57 \log_2 n)$ c) $T(n)$ is $\Theta((n+2)^2)$

2. (12 marks)

a) (4 marks) Give the recurrence for the time complexity of the following recursive algorithm:

```
procedure FUNKY-REC(m, n)
    if (m == 1)
        sum = 0
        for (i = 1; i <= n; i++)
            for (j = 1; j <= m; j++)
                sum = sum + FUNKY-ITR(m) / i * j
        return(sum)
    else if (m > 1)
        sum = 0
        for i = 1 to n do
            l = FUNKY-REC(m/3, n)
            sum = FUNKY-ITR(n) * l
        return(sum)
    else
        return(m)
```

Assume that procedure `FUNKY-ITR(x)` runs in $O(x \log_2 x)$ time.

b) (8 marks) Derive an upper bound for $T(n)$ using any method discussed in class or in the textbook, where

$$T(n) = \begin{cases} 3 & n \leq 1 \\ 9T(n/3) + cn^2 & n > 1 \end{cases}$$

3. (9 marks) Consider the following algorithm:

```
sum = 0;
for i = 1 to n * n do
    j = T1(n)
    if (T2(i, j, n)
        for k = 1 to n do
            l = T3(n, k) - 2
            sum = (sum * j / l)
if ((sum % 2) == 0)
    sum = sum / 2
else
    sum = sum * 2
```

Give the asymptotic worst-case time complexity of this algorithm when the T1, T2, and T3 operations require:

a) $O(n^2)$, $O(1)$, and $O(1)$ time, respectively, and we don't know how many times T2 evaluates to **true**.

b) $O(1)$, $O(n)$, and $O(1)$ time, respectively, and we know that T2 evaluates to **true** $O(n)$ times.

c) $O(1)$, $O(1)$, and $O(1)$ time, respectively. and we know that T2 evaluates to **true** $O(n \log_2 n)$ times.

4. (18 marks)

a) (4 marks) How are problems that have dynamic programming algorithms different from problems that have greedy algorithms? Phrase your answer in terms of the problem solution-space properties given in class.

b) (14 marks) Consider the following algorithm:

```
DFS-V(i, sol, U, B, val)
    if (SIZE(sol) > B)
        print("pruned: solution too big for knapsack")
    else if (i == n)
        if (VALUE(sol) == val)
            print(sol)
    else
        DFS-V(i + 1, sol, U, B, val)
        DFS-V(i + 1, UNION-COPY(sol, U[i]), U, B, val)
```

The problem solved here is a variant of 0/1 KNAPSACK that prints all viable knapsack-loads of a given value val. In this algorithm, U is the set of items, n is the number of items in U, B is the knapsack size bound, sol is a subset of U, SIZE(sol) returns the sum of the sizes of the items in sol, VALUE(sol) returns the sum of the values of the items in sol, and UNION-COPY($sol, U[i]$) returns a copy of sol to which the ith item in U has been added. To answer this question, sketch the implicit tree of solution-nodes generated by the algorithm above (marking leaf-nodes which are printed by a circle and nodes that are pruned with a square) with the call DFS-V($0, sol, U, B, val$) when $U = \{X, Y, Z\}$ such that $size(X) = 1$, $size(Y) = 2$, $size(Z) = 1$, $value(X) = 2$, $value(Y) = 1$, and $value(Z) = 1$, sol is the empty set, and B and val have the following values:

i) (7 marks) $B = 1$ and $val = 2$:

ii) (7 marks) $B = 3$ and $val = 2$:

5. (12 marks) The length of the longest weighted common subsequence (LWCS) of two strings s and s' is defined by the following recurrence:

$$D(i,j) = \begin{cases} 0 & i = 0 \text{ or } j = 0 \\ D(i-1, j-1) + 2 & \text{if } s(i) = s'(j) \\ D(i-1, j-1) + 1 & \text{if } s(i) \neq s'(j) \text{ and } SAME(s(i), s'(j)) \\ \max(D(i, j-1), D(i-1, j)) & \text{otherwise} \end{cases}$$

For any two symbols x and y, function $SAME(x, y)$ returns **true** if either (i) x and y are both vowels or (ii) x and y are both consonants, and **false** otherwise (recall that a vowel is one of the letters $\{'A', 'E', 'I', 'O', 'U'\}$ and a consonant is any letter of the alphabet that is not a vowel). Given the above, determine the longest weighted common subsequence of the strings **AQUIF** and **OUT** – that is, fill in the dynamic programming matrix given below, show all matrix-cell backpointers, show one of the backpointer paths that gives an optimal LWCS, and show the LWCS corresponding to this path.

			A	Q	U	I	F
	j	0	1	2	3	4	5
i							
	0						
O	1						
U	2						
T	3						