Algorithms Spring 2006 Graduate Comprehensive Exam

1. Answer the following short questions: [20 pts]

a. The *Dynamic Programming* algorithms have bottom up control while the *Divide and Conquer* algorithms have top down control. True/False

b. The set of NP-complete problems is a subset of the NP-class of problems. True/False

c. It has been proved that NP-complete problems cannot have polynomial algorithms. True/False

d. In order to prove a problem X to be NP-hard one should develop a polynomial transformation from X to a known NP-hard problem. True/False

e. The *Single source shortest path finding problem* is P-class problem. True/False

f. Name a well-known algorithm for the *Minimum spanning tree* finding problem.

g. 4-SAT (where each clause in a Boolean Satisfiability problem has four literals) is an NP-hard problem. True/False

h. In general $O(N^2)$ algorithm is worse than $O(N\log N)$ algorithm. True/False

i. When would you buy an $O(N^{100})$ algorithm over an $O(2^N)$ algorithm for the same problem?

j. Which problem does the well-known *Floyd's algorithm* solve?

2. The next question is related to the Maximum Subsequence problem. MaxSubseq problem over a sequence of positive and negative numbers is to find a subsequence that produces the largest sum. For instance, over a sequence (3 -1 9 -5 2), the answer is 11 for the subsequence (3 –1 9). The following iterative algorithm calculates the MaxSubseq.

```
Algorithm MaxSubseq1(an array of numbers a, of length n)
MaxSum=0;
For ( i=0; i<n; i=i+1)
        For ( j=i; j<n; j=j+1)  {
                thisSum=0;
                For ( k=i; k<=j; k=k+1)
                        thisSum=thisSum+a[k];
                If  (thisSum>MaxSum)
                        MaxSum=thisSum;
                };
return MaxSum;
End Algorithm.
```

The innermost loop over *k* is redundant. Improve the algorithm by appropriately removing it and describe how is the time-complexity improved in your algorithm. [20]

3. The following is a recurrence formula. Write a Dynamic Programming algorithm for computing all a[i,j]'s, where i and j are integers between 0 and a constant $N>0$.

$a[i, 0] = -i$, $a[0, j] = -j$,

$$
a[i, j] = \max\{\ a[i-1,k]-2,\ 0 \le k < j;\quad a[p, j-1]-2,\ 0 \le p < i;\quad a[p-1, k-1]-1\},\ 0 \le p < i,\ 0 \le k < j\},\ \text{for both } i \text{ and } j > 0.
$$

Analyze the time-complexity of the algorithm. [20]

4. Set up the recurrence equation for asymptotic time complexity of the following algorithm and solve it for the usual theta function. [Assume n=end-start+1= 2k, for some integer k>0.]

```
Algorithm Little (int array A[], int start, int end)
begin
if end = = start do
        return              // null
else    Little (A, start+2, end);

End Algorithm.
```

[20]

5. The following is a directed weighted graph. Draw it first. [Usual presumption of adjacency list representation of the graphs holds for a graph theoretic question.]
V={a, b, c, d, e}, E={(a, b, 2), (a, d, 8), (b, c, 3), (c, d, 2), (c, e, 5), (d, e, 1), (e, b, 2)}.

For the following algorithm find out what the output from line 4 would be.

(1) enqueue all arcs in Q;
(2) while Q not empty do
(3) $\quad (v, w, d) = pop(Q)$;
(4) $\quad$ print (v, w, d);
(5) $\quad d = d - 3$;
(6) $\quad$ if $d \geq 0$ then $push$ (v, w, d) on Q;
end while loop;