Name: Wendong Chen Student ID:931018 Username: wendongc1

1. (a)

the recurrence relation: $F(n) = \begin{cases} a & ,n = 1 \\ F(n-1) + cn + k, & n > 1 \end{cases}$

$F(n) = F(n-1) + cn + k;$

$= (F(n-2) + c(n-1) + k) + cn + k = F(n-2) + cn + c(n-1) + 2k;$

$= (F(n-3) + c(n-2) + k) + cn + c(n-1) + 2k = F(n-3) + cn + c(n-1) +$
$c(n-2) + 3k;$

.....

$= F(n-i) + cn + c(n-1) + \cdots + c(n-i+1) + ik;$

.....

$= F(n-n+2) + cn + c(n-1) + \cdots + 3c + (n-2)k;$

$= F(1) + cn + c(n-1) + \cdots + 2c + (n-1)k;$

$= a + \frac{n^2 + n - 2}{2}c + (n-1)k$

(b)

There is only 1 multiplication executed on each recursion of this algorithm. So the recurrence relation is: $M(n) = \begin{cases} 0 & ,n = 1 \\ M(n-1) + 1, & n > 1 \end{cases}$

$M(n) = M(n-1) + 1;$

$= (M(n-2) + 1) + 1 = M(n-2) + 2;$

$= (M(n-3) + 1) + 2 = M(n-3) + 3;$

.....

$= (M(n-i) + 1) + i - 1 = M(n-i) + i;$

.....

$= (M(n-n+2) + 1) + n - 3 = M(2) + n - 2;$

$= (M(1) + 1) + n - 2 = M(1) + n - 1;$

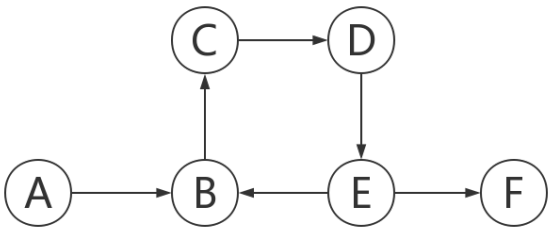
$= n - 1$

(c)

There are 2 arithmetical operations in the recursion: addition and multiplication, which both of them can be the basic operation. We can just count one of them which means the other arithmetical operation is counted automatically. We choose multiplication as the basic operation and we've already got M(n)=n-1, so the time complexity of the algorithm is $\Theta(n)$.

2. Time complexity represents the speed of solving problems in the worst case, but it doesn't mean it goes faster (or slower) certainly. Like insertion sort, the time complexity in the worst case is $\Theta(n^2)$, but in the best case, its complexity is $\Theta(n)$ which is faster than $\Theta(n \log n)$. There is no algorithm which rans fastest in all situations. Some may better perform on almost-sorted lists, others may work faster on randomly ordered items. Maybe Algorithm B rans faster than Algorithm A in most cases but I cannot say either one of them will sort an array of 1000 elements faster, because it depends on situations.

3. In a cyclic graph, there must be part of the path which starts and ends at the same vertex, so every node in a cycle must have an incoming edge and an outgoing edge which points to another node in the cycle (Like cycle 'B','C','D','E' in the example). The decrease-and-conquer approach doesn't allow the nodes in a graph having incoming edges, so we cannot remove any of the nodes ('B','C','D','E') in the cycle (because all of them have incoming edges).



4.

(a)

k	V	E
1	2	0
2	4	4
3	6	12
4	8	24
5	10	40
6	12	60

(b)

$V_k = 2k$

(c)

the recurrence relation: $E(k) = \begin{cases} E(k-1) + 2V_{k-1} = E(k-1) + 4(k-1), k > 1 \\ 0 \qquad\qquad\qquad\qquad\qquad\qquad\quad , k = 1 \end{cases}$

(d)

$$E(k) = E(k-1) + 4(k-1);$$
$$= \big(E(k-2) + 4(k-2)\big) + 4(k-1) = E(k-2) + 4(k-1) + 4(k-2);$$
$$= \big(E(k-3) + 4(k-3)\big) + 4(k-1) + 4(k-2) = E(k-3) + 4(k-1) + 4(k-2) + 4(k-3);$$
$$\cdots\cdots$$
$$= E(k-i) + 4(k-1) + 4(k-2) + \cdots + 4(k-i);$$
$$\cdots\cdots$$
$$= E(k-k+2) + 4(k-1) + 4(k-2) + \cdots + 4(k-k+2);$$
$$= E(1) + 4(k-1) + 4(k-2) + \cdots + 4(k-k+2) + 4 * 1;$$
$$= 2k^2 - 2k$$

5.

```
function CountFeedLotPaths(G<V, E>, u, v)
    //Input: G the graph representing the cattle station consisting of V feed lots with E edges between
                the feed lots
                starting feed lot u and finishing feed lot v
    //Output: the number of shortest paths from u to v

    mark each node in V with 0
    count ← 1, init(queue)                          //create an empty queue
    mark u with count
    inject(queue, u)                                //queue containing just u

    while v is not in the queue do
        count ← count + 1
        while the front vertex a is marked with  count − 1  do
            a ← eject(queue)                        //dequeues a (the front vertex)
            for each edge (a, b) adjacent to a do
                if b is marked with count or 0 do
                    mark b with count
                    inject(queue, b)                //enqueues b

    i ← 0
    while the queue is non-empty do
        c ← eject(queue)                            //dequeues c (the front vertex)
        if c is vertex v do
            i ← i + 1
    return i
```

The graph is traversed by using adjacency list, so for each node we traverse the list *adj[v]*. In this case, the time complexity is *O(V+E)*.

6.

function UPDATE(*A[0, ..., n-1], G, p, q*)

//Input: *A* an array of neural unit activation levels x_i in the neural network at time t

G the neural unit interaction graph in adjacency matrix *(n × n)* format

//Output: A an updated array containing the activation levels for each neural unit x_i in the neural network at time *t + 1*

for $i \leftarrow 0$ to $n-1$ **do**

$x_i^t \leftarrow A[i]$

for $i \leftarrow 0$ *to* $n-1$ **do**

$A[i] \leftarrow p \tanh(x_i^t)$

for $j \leftarrow 0$ to $n-1$ **do**

if $j \neq i$ **do**

$A[i] \leftarrow qG_{ij} \tanh(x_j^t) + A[i]$

return $A[i]$