HW2b:

b) for any n>2 we have:
fib(n) = fib(n-1) + fib(n-2)
fib(n-1) = fib(n-2) + fib(n-3)
fib(n-2) = fib(n-3) + fib(n-4)
and so on...

the algorithm trace above shows how the same fibonacci numbers are computed more than one time

e)

n	recFib	T(recFib)	IterFib	T(iterFib)
30	832040	0	832040	0
31	1346269	10000	1346269	0
32	2178309	30000	2178309	0
33	3524578	40000	3524578	0
34	5702887	90000	5702887	0
35	9227465	140000	9227465	0
36	14930352	230000	14930352	0
37	24157817	380000	24157817	0
38	39088169	620000	39088169	0
39	63245986	1000000	63245986	0
40	102334155	1630000	102334155	0
41	165580141	2640000	165580141	0

by looking at n=31, 32, 33 you can see the the T(n) = T(n-1)+ T(n-2)
or T(33) = T(32) + T(31);
While the T represents computer time, it represents the time complexity in a very practical manner.

The following table shows larger values of n's
compared to the golden ratio estimate

n	recFib	(phi)^n/sqrt(5)
30	832040	832040.019285451
31	1346269	1346269.03224411
32	2178309	2178309.05385538
33	3524578	3524578.08986275
34	5702887	5702887.14980721
35	9227465	9227465.24952228
36	14930352	14930352.4152709
37	24157817	24157817.6905869
38	39088169	39088170.1475929
39	63245986	63245987.9057086
40	102334155	102334158.162565
41	165580141	165580146.245067

The estimate is failry accurate. The time complexity of the function is really the same as the recursion of the fibonacci function, so the golden ratio estimate also applies to the time complexity;