Problem 5.1

a)

b)

max time 10000ms	naive recursive		bottom up		closed form		matrix representation	
n	index	time(ms)	index	time(ms)	index	time(ms)	index	time(ms)
1	1	0	1	0	1	0	1	0
5	5	0	5	0	5	0	5	0
10	10	0	10	0	10	0	10	0
20	20	1000	20	0	20	0	20	0
50	33	16002	50	0	50	1000	50	1000
100	32	10002	100	1000	100	1000	100	1000
300	32	13002	300	1001	300	1000	153	10002
500	33	18513	500	2001	500	1000	172	12002
1000	32	21002	1000	6001	1000	1000	189	10000
3000	33	28002	976	13003	3000	1000	172	12001
5000	33	14001	1116	10007	5000	1000	136	10002

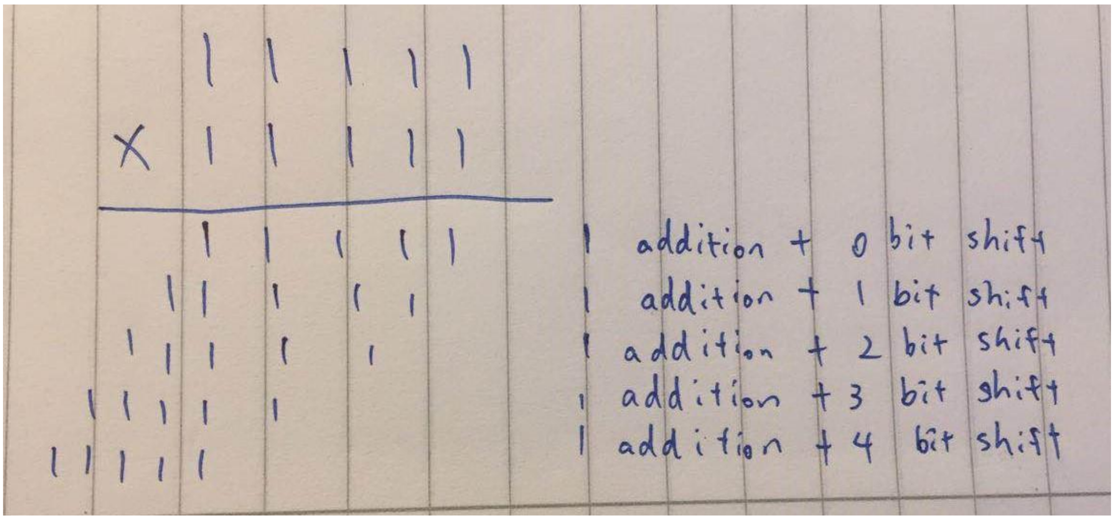
c)

For Naive recursive, Bottom Up, and Matrix representation, the same n always returns the same Fibonacci Number. Because they have the same mechanism behind, which is add up the previous two numbers to get the next one.

For Closed Form method, since it is based on a formula and the return type of the function is float, for some n the result can be different from the actual Fibonacci Numbers. Especially the bigger n gets, float numbers have limited number of decimals for square root of 5, which can be a reason for results to be different.

Problem 5.2

a)



Since these two large integers have the same number of digits, assume the length is n, the sum of the addition operation will be x times n, where x is between 0 and 9. Assume the worst case, it is 9n. For every line there is a bit shifting process, the sum of that is (0+n-1)*n/2, which is almost like 0.5n^2.

Therefore, the time complexity for brute-force approach is 9n+0.5n^2, in which case

0.5n^2 dominates. The total running time for this approach is θ(n^2).

b)

$$x = 10^{n/2} a + b$$

$$y = 10^{n/2} c + d$$

$$xy = 10^{n} ac + 10^{\frac{n}{2}} (ad + bc) + bd$$

where $ad + bc = (a+b)(c+d) - (ac+bd)$

According to the above Karatsuba Multiplication formula, divide and conquer can be applied here.

```
Int Multi (int x, int y, int length) {
    if (length==1)
        return x*y
    a = first length/2 digits of x
    b = second half of x
    c = first half of y
    d = second half of y
    ac = Multi (a, c, length/2)
    bd = Multi (b, d, length/2)
    ad_bc = Multi (a+b, c+d, length/2) - (ac+bd)
    return 10^length * ac + 10^(length/2) * ad_bc + bd
}
```

c)

Obviously, the recurrence is T(n) = 3T(n/2)+n, since addition and bit shifting can be done linearly and length of the numbers are divided into 2 after every recurrence.

e)

$$T(n) = 3T(n/2) + n$$

$$n^{\log_b a} = n^{\log_2 3} = n^{1.58}$$

$$f(n) = n$$

Case I: $f(n) = O(n^{1.58-\epsilon})$

Thus $T(n) = \Theta(n^{\log_2 3})$