Problem 5.1

a)

b)



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)![图片包含 文字

描述已自动生成]()

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)![图片包含 文字

描述已自动生成]()

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)![图片包含 文字

描述已自动生成]()