Problem Set 3

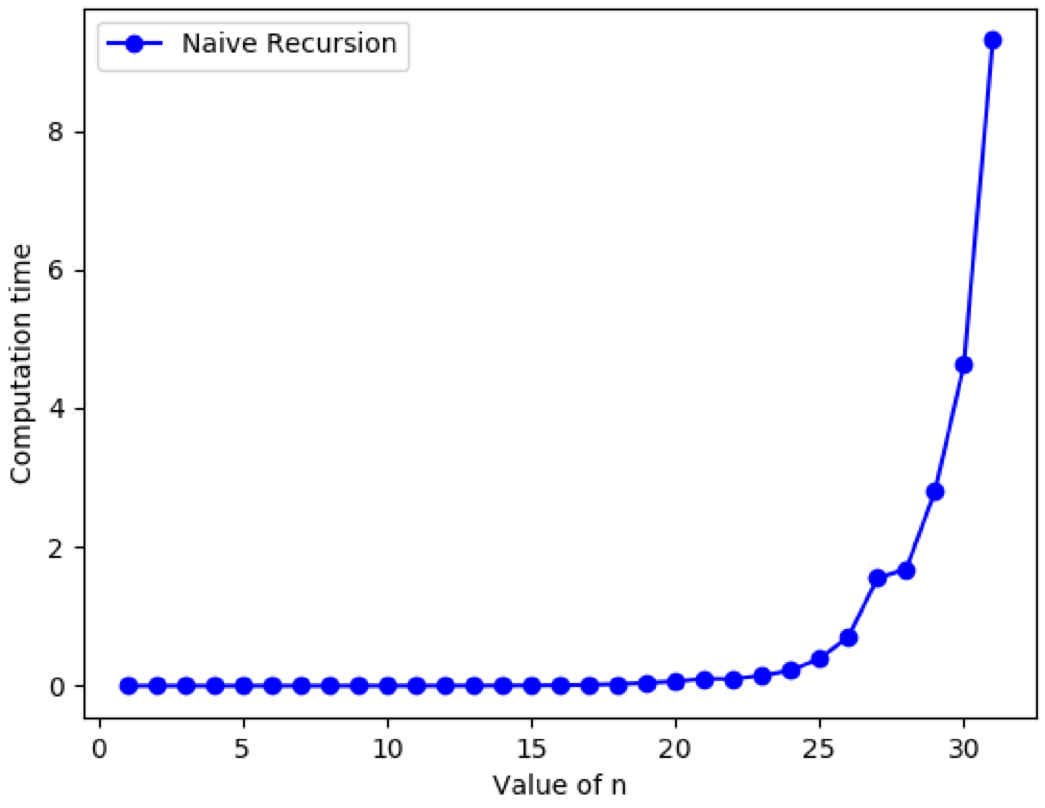
Problem 3.1

Solution:
(a) See the file fibonacci.py for implementation.

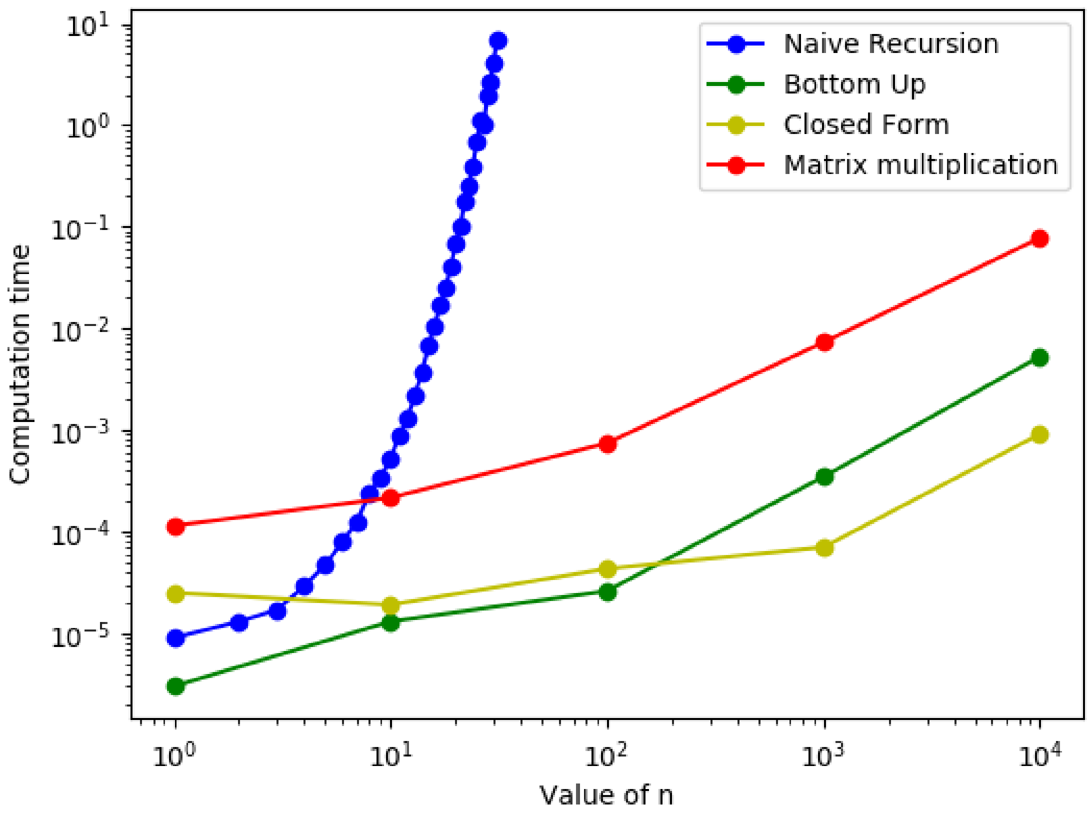
(b) See sample.txt for results. Did not sample for values higher than 10000 to avoid decimal overflow.

(c) In practice, some of the results are an approximation. For the closed form approach, since it is a numerical approximation, for large values of n, it will not always return a good approximation because of limited precision. Therefore, the result will possibly be different compared to the other approaches for large values of n. However, using arbitrary libraries such as decimal (for python), one can obtain pretty good approximations by specifying precision.

(d) For the naive recursion approach:



For all approaches together:



For the naive approach, which is quite slow, the time complexity is $O(2^n)$. The other approaches are quite fast and we get quick results. Bottom-up, matrix and closed form get slower at very large values because the cost of the exponential and multiplication operations in the algorithm increases.

Problem 3.2

Solution:
(a) For bit shifting and for addition, it takes $\Theta(n)$. When we perform the brute-force multiplication, we multiply each bit of the first number with all bits bit of the second, and then add the shifted sums. This is n^2 operations, so $\Theta(n^2)$.

(b)The divide and conquer algorithm is as follows:
function multiply(a,b): for binary strings a and b.
1. Split the binary strings a and b into two: al, ar, bl, br.
2. Solve these problems recursively $p1 = multiply(al, bl)$, $p2 = multiply(bl, cl)$,$p3 = multiply(al + ar, bl + br)$.
3. return $p1 * 2^n + (p3 - p1 - p2) * 2^{n/2} + p2$, the sum of the shifted results.

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

(d)The tree has $1 + log_2 n$ levels. The sum of the time at all levels will be: $n + (3/2)n + (3/2)^2 n + (3/2)^3 n + ... + (3/2)^{log_2 n} n$. Using the formula for geometric series sum, this expression $= 3n^{log_2 3} - 2n = \Theta(n^{1.585})$.

(e)$T(n) = 3T(n/2) + O(n)$. Using master theorem, $a = 3, b = 2, f(n) = n$. Case 1: $f(n) = O(n^{log_2 3 - 0.585})$. Therefore, $T(n) = \Theta(log_2 3)$.