**oHomework 5**

**-Nayan Man Singh Pradhan**

Problem 5.1

a.

All four method to compute Fibonacci numbers have been implemented in c++ file “5\_1a.cpp”.

b.

The runtime is measured for all the methods using chrono standard library. To make the program stop after a certain amount of time, I have a while (1) loop that either stops when the max time has been reached or when all elements till n is executed in the loop. An example of the implementation is shown below (for naïve recursive method). (I would recommend running the program with maxsize = 40 as the numbers start to wrap even when using long long).

A screenshot of a cell phone

Description automatically generated

A table with the time taken for each method is made in a four different data files “naive.txt”, “bottom.txt”, “closed.txt”, “closed.txt”, and “matrix.txt” for naïve recursive method, bottom up method, closed form method, and matrix representation method respectively. The file should be created after the program “5\_1b.cpp” is compiled and executed. I am using the “Power of Number” method while computing the power of terms in my matrix and closed form calculation in order to reach time complexity of .

A table for the ‘n’ (in this case going from 1 to 40 because the data needs to fit in one page!) and the time taken to compute the Fibonacci series for the different methods. The time is calculated in nanoseconds! We notice that the time taken to compute the Fibonacci series gets considerably larger for the Naïve Recursive method (when comparting to the other data types).

|  |  |
| --- | --- |
| n | Naïve Rec |
| 1 | 139 |
| 2 | 168 |
| 3 | 152 |
| 4 | 196 |
| 5 | 320 |
| 6 | 401 |
| 7 | 480 |
| 8 | 533 |
| 9 | 716 |
| 10 | 987 |
| 11 | 1404 |
| 12 | 2182 |
| 13 | 3461 |
| 14 | 5572 |
| 15 | 8909 |
| 16 | 14340 |
| 17 | 22811 |
| 18 | 36842 |
| 19 | 59576 |
| 20 | 96197 |
| 21 | 156648 |
| 22 | 253129 |
| 23 | 407155 |
| 24 | 657429 |
| 25 | 1067920 |
| 26 | 1732901 |
| 27 | 2506440 |
| 28 | 4290888 |
| 29 | 5022036 |
| 30 | 7090633 |
| 31 | 10655819 |
| 32 | 15383734 |
| 33 | 23212773 |
| 34 | 37733338 |
| 35 | 59351110 |
| 36 | 95374154 |
| 37 | 152553353 |
| 38 | 259395961 |
| 39 | 416486242 |
| 40 | 673924852 |

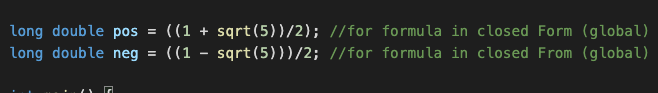
|  |  |
| --- | --- |
| n | Bottom up |
| 1 | 265 |
| 2 | 881 |
| 3 | 697 |
| 4 | 1150 |
| 5 | 1120 |
| 6 | 1192 |
| 7 | 1122 |
| 8 | 1961 |
| 9 | 1976 |
| 10 | 1893 |
| 11 | 1910 |
| 12 | 1985 |
| 13 | 1924 |
| 14 | 2112 |
| 15 | 2022 |
| 16 | 3668 |
| 17 | 3559 |
| 18 | 3468 |
| 19 | 3523 |
| 20 | 3803 |
| 21 | 3533 |
| 22 | 3628 |
| 23 | 3493 |
| 24 | 3712 |
| 25 | 3499 |
| 26 | 3674 |
| 27 | 3556 |
| 28 | 3697 |
| 29 | 3520 |
| 30 | 3616 |
| 31 | 3589 |
| 32 | 7083 |
| 33 | 6790 |
| 34 | 6792 |
| 35 | 6723 |
| 36 | 7111 |
| 37 | 6733 |
| 38 | 6858 |
| 39 | 6744 |
| 40 | 7092 |

|  |  |
| --- | --- |
| n | Matrix Rep |
| 1 | 269 |
| 2 | 855 |
| 3 | 796 |
| 4 | 1317 |
| 5 | 1400 |
| 6 | 1354 |
| 7 | 1233 |
| 8 | 2391 |
| 9 | 2505 |
| 10 | 1938 |
| 11 | 2052 |
| 12 | 1762 |
| 13 | 1385 |
| 14 | 1407 |
| 15 | 1395 |
| 16 | 2640 |
| 17 | 2608 |
| 18 | 2674 |
| 19 | 2632 |
| 20 | 2741 |
| 21 | 2741 |
| 22 | 2698 |
| 23 | 2698 |
| 24 | 2694 |
| 25 | 2689 |
| 26 | 2750 |
| 27 | 2675 |
| 28 | 2710 |
| 29 | 2690 |
| 30 | 2677 |
| 31 | 2625 |
| 32 | 5099 |
| 33 | 5098 |
| 34 | 5085 |
| 35 | 5102 |
| 36 | 5082 |
| 37 | 5092 |
| 38 | 5092 |
| 39 | 5066 |
| 40 | 5091 |

|  |  |
| --- | --- |
| n | Closed |
| 1 | 212 |
| 2 | 721 |
| 3 | 602 |
| 4 | 1093 |
| 5 | 1101 |
| 6 | 1100 |
| 7 | 1127 |
| 8 | 1972 |
| 9 | 1995 |
| 10 | 2046 |
| 11 | 2158 |
| 12 | 2184 |
| 13 | 2165 |
| 14 | 2078 |
| 15 | 2082 |
| 16 | 3767 |
| 17 | 3729 |
| 18 | 3783 |
| 19 | 3694 |
| 20 | 3815 |
| 21 | 3841 |
| 22 | 3731 |
| 23 | 3791 |
| 24 | 3852 |
| 25 | 3851 |
| 26 | 3803 |
| 27 | 3799 |
| 28 | 3811 |
| 29 | 3799 |
| 30 | 3724 |
| 31 | 3746 |
| 32 | 7329 |
| 33 | 7311 |
| 34 | 7344 |
| 35 | 7255 |
| 36 | 7276 |
| 37 | 7299 |
| 38 | 7172 |
| 39 | 7284 |
| 40 | 7220 |

c.

For the same value of n, all four methods might not return the same Fibonacci number! This is because, as the value of n gets larger and closer to infinity, the Closed Form approach contain floating-point errors. We use the equation involving the terms and as a long double values, then subtract neg from pos, compute the power of the term to n, then finally divide the term by the square root of 5! This value may round off certain terms based on the property and size of the data type “double”, which may give possible errors.



A screenshot of a cell phone

Description automatically generated

d.

The graph has been plotted using gnuplot. To get the graph, one must type “gnuplot plot.plt” into the terminal. I have attached a picture of the graph below (and also in my zip file for reference). The graph shows that Naïve Recursive takes exponential time (the most). The graph is made with logarithmic scales so that the values can be seen clearly.

**Time complexity for the following methods:**

Naïve Recursive:

Closed Form: when using the “power of number” recursion

Bottom Up Approach:

Matrix Representation:

A close up of a map

Description automatically generated

Problem 5.2

a.

b.