Problem 1

b)

The following tables show the number n in the first column, the Fibonacci number in the second column and the execution time in nanoseconds (used chrono to measure)

naive		
Column1	Column2	Column3
0	0	200
1	1	100
2	1	100
3	2	100
4	3	200
5	5	200
7	13	200
9	34	600
11	89	900
14	377	2800
17	1597	13400
21	10946	69700
26	121393	759500
32	2178309	10794800
39	63245986	310379200
47	1.84467E+19	15608266500

bottom up		
Column1	Column2	Column3
0	0	100
1	1	100
2	1	100
3	2	100
4	3	100
5	5	100
7	13	100
9	34	100
11	89	100
14	377	100
17	1597	100
21	10946	100
26	121393	100
32	2178309	200
39	63245986	200
47	1.84467E+19	200

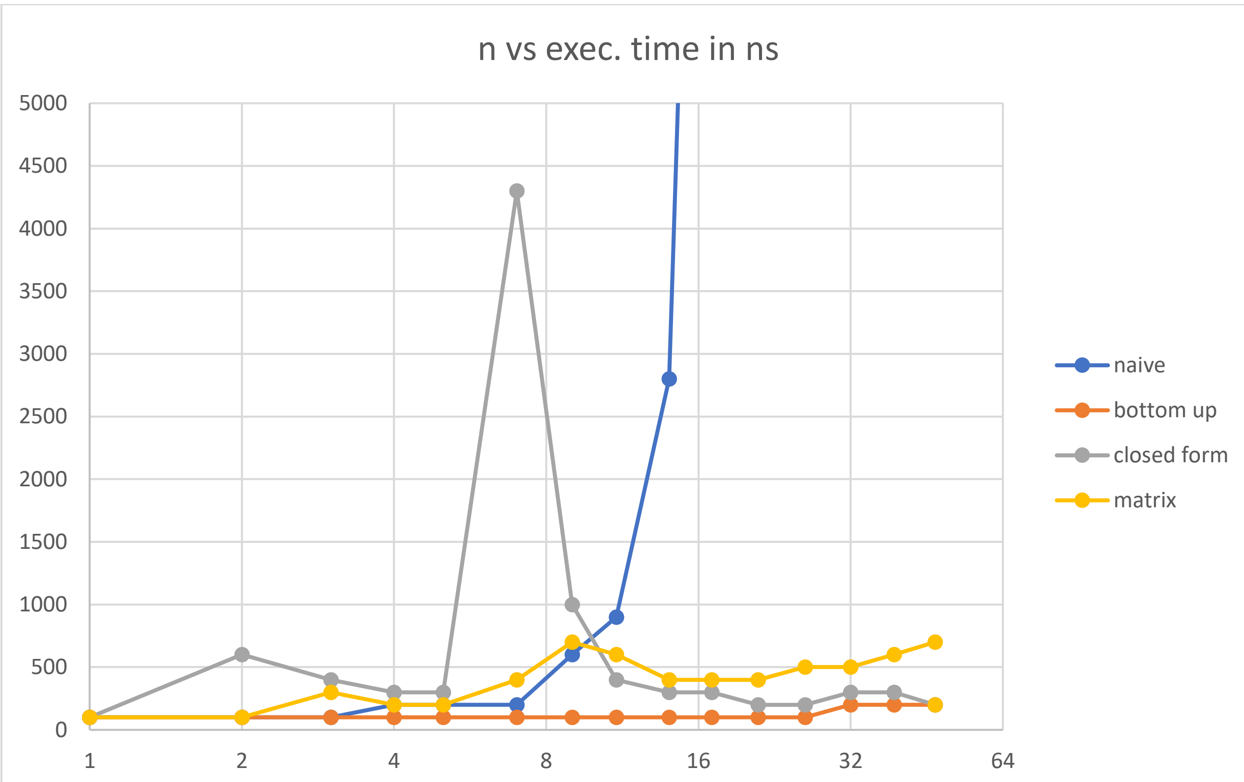
closed form		
Column1	Column2	Column3
0	0	100
1	1	100
2	1	600
3	2	400
4	3	300
5	5	300
7	13	4300
9	34	1000
11	89	400
14	377	300
17	1597	300
21	10946	200
26	121393	200
32	2178309	300
39	63245986	300
47	1.84467E+19	200

matrix		
Column1	Column2	Column3
0	0	100
1	1	100
2	1	100
3	2	300
4	3	200
5	5	200
7	13	400
9	34	700
11	89	600
14	377	400
17	1597	400
21	10946	400
26	121393	500
32	2178309	500
39	63245986	600
47	2971215073	700

c)

In my testing the numbers remained the same except for the matrix multiplication which was not able to calculate the correct number for its last data point. In this small data range, I would assume that most of the numbers will stay the same no matter what method we are using to calculate them. If with way larger n I would assume that the floating-point error could influence the closed form method. Since the other version do not rely on floating point number, we should not get into problems with those.

d)



Unfortunately, I did not really manage to find a data type that can handle large enough number for numbers over 50. The best data type I found was unsigned long long. Therefore, the execution time is similar.