Homework 5 by Timofei Podlorytov

5.1

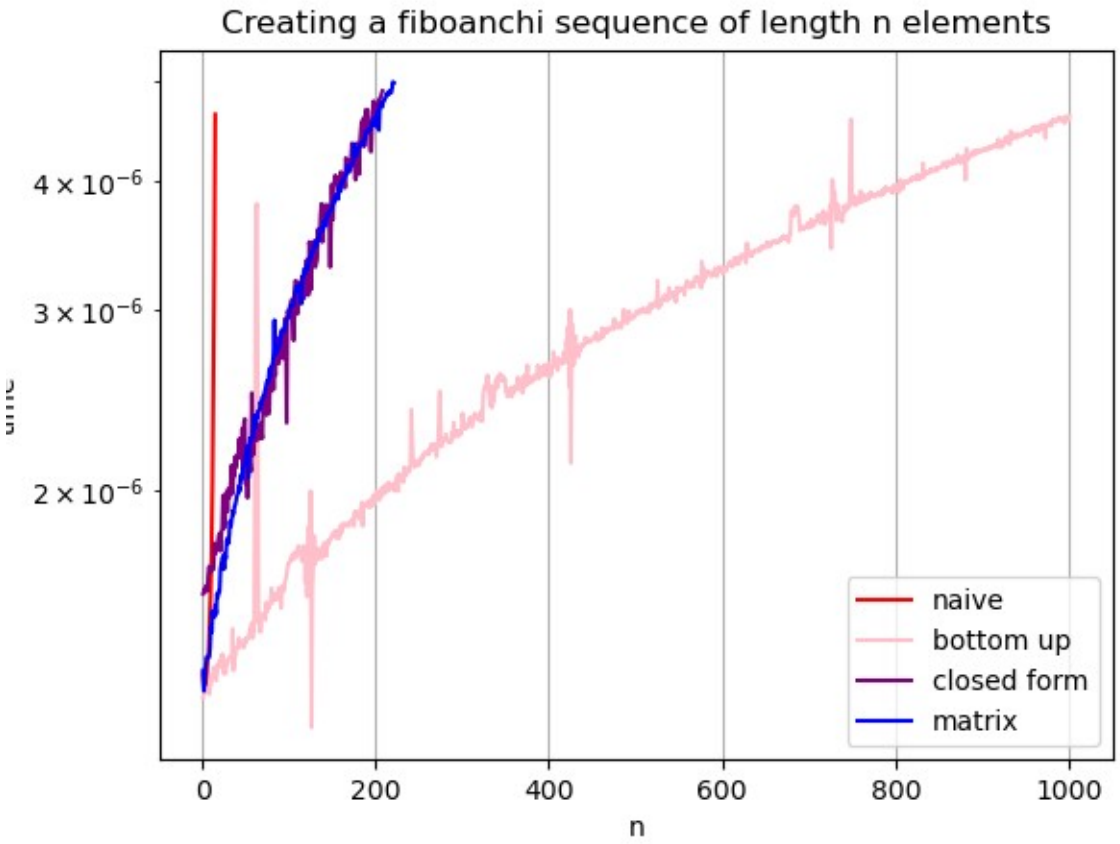
a)All the functions are implemented in p1.cpp file while the graph was created in a python file graph.py as well as the creation of the table.

b)I Sampled and measured the running time for all methods and created a graph as well as csv table. The table is places below. The number in the column header represents how many elements did each method manage to get, while -1 feel the gaps in the table and mean the absence of data.

c)The output is not always the same. As in closed form approach we deal with floating point numbers and thus we accumulate error. And as can be seen even with the first few elements the values don't always match all the other 3 methods which use integer calculation and as a result are exact.

Basically using floating point numbers results in error rate. Meanwhile integer calculation is precise.

d) We plotted results on a logarithmic scale and expected the following rate:
T (n) = $\Omega(\Phi^n)$ -Naive
T (n) = $\Theta(n)$ – bottom up, where we store 2 previous elements.
T (n) = $\Theta(\lg n)$ -for matrix and closed form



The expectations don’t fully match the results. As we see naive approach is by far the worst. While matrix and closed form performed really close, and bottom up approach managed to do the most elements.

Even though matrix and closed form are logarithmic, we dismiss the constants, which evidently appear to be quite large. So, in our test we see that a seemingly less efficient but simpler algorithm performs better. The reasoning is likely that floating point computations take longer than integer sum. As for the matrix inner function call and all the matrix multiplication might have made it also less efficient. The multiplication operation is costlier than addition especially for floats, causing this output.

Consequently, even though in asymptotic behavior closed form and matrix might be better, for n<1000 the complexity of their implementation makes them worse than a linear bottom up approach whose implementation is extremely simple and requires no operations besides addition of integers.

Table:(-full in csv)

naive 14	bottom up 1000	closed form 207	matrix 220
0.00000133	0.00000126	0.00000159	0.00000134
0.00000128	0.00000128	0.0000016	0.00000128
0.00000131	0.00000128	0.00000161	0.00000132
0.0000013	0.00000128	0.0000016	0.00000135
0.00000133	0.00000128	0.00000162	0.00000138
0.00000138	0.00000129	0.0000016	0.00000138
0.00000139	0.00000129	0.00000166	0.00000138
0.00000149	0.00000127	0.00000169	0.00000138
0.00000158	0.00000129	0.00000168	0.00000141
0.00000173	0.0000013	0.00000167	0.00000148
0.00000202	0.00000132	0.00000168	0.00000146
0.00000255	0.00000133	0.00000169	0.00000153
0.0000033	0.00000134	0.00000168	0.00000151
0.00000464	0.00000135	0.00000178	0.00000151
-1	0.00000132	0.00000174	0.00000155
-1	0.00000131	0.00000175	0.0000015

			2
-1	0.00000133	0.00000176	0.0000015 9
-1	0.00000136	0.00000178	0.0000015 8
-1	0.00000135	0.00000181	0.0000016
-1	0.00000137	0.00000182	0.0000016
-1	0.00000135	0.00000175	0.0000016 8
-1	0.00000136	0.00000182	0.0000017
-1	0.00000139	0.00000184	0.0000017 2
-1	0.00000137	0.00000198	0.0000017 3
-1	0.00000133	0.00000184	0.0000016 8
-1	0.00000134	0.00000188	0.0000017 4
-1	0.00000133	0.00000199	0.0000017 2
-1	0.00000137	0.00000199	0.0000017 9
-1	0.00000136	0.00000191	0.0000018
-1	0.00000137	0.00000201	0.0000017 8
-1	0.00000138	0.00000197	0.0000018 1
-1	0.00000138	0.00000199	0.0000018 7
-1	0.00000138	0.00000205	0.0000018 5
-1	0.00000138	0.00000212	0.0000018 8
-1	0.00000147	0.00000199	0.0000019 1
-1	0.00000134	0.0000021	0.0000019
-1	0.00000137	0.00000207	0.0000019 3
-1	0.00000138	0.00000216	0.0000019 4
-1	0.0000014	0.00000211	0.0000019 6
-1	0.00000141	0.00000216	0.0000019 8
-1	0.00000141	0.00000217	0.0000019 9
-1	0.00000144	0.00000219	0.0000020 2
-1	0.00000141	0.00000225	0.000002

5.2

Consider the problem of multiplying two large integers a and b with n bits each (they are so large in terms of digits that you cannot store them in any basic data type like long long int or similar). You can assume that addition, substraction, and bit shifting can be done in linear time, i.e., in Θ(n)

a)

Ex: we have 4 bit numbers:
1011 x 1110=11*15=165

1011
1110
0000-replace with 0 addition
1011 -shift left 1 + addition 2*Θ(n)*
1011 -shift left 1 + addition 2*Θ(n)*
1011 -shift left 1+ addition 2*Θ(n)*
10100101=165 we performed it in 6*Θ(n)=Θ(n) as constants are dismissed

For a more general case we have to shift (n-1) times and add n times→
(n-1)*Θ(n)+n*Θ(n)=Θ(n)*(2n-1)=$\Theta(n^2)$

The result is $\Theta(n^2)$ time complexity for the brute force method.

b) We could simplify the problem to multiplying 2 2digit numbers. We split both numbers I half and multiply them separately.

	a	b
	c	d
	a*d	b*d
a*c	b*c	
a*c	a*d+b*c	b*d

For a length n of the initial numbers we can express the formula as such. Each "*" means a recursion. << and + are left shift as well as addition that can be performed in linear time.
f=b*d+(a*d+b*c)<<(n/2) + a*c<<n
4 multiplications and 5 linear operations.
The algorithm:
-here I replaced halves with a and b and c and d

```
mult(ab,cd):
    if(length ab ==1 and length cd==1)
        if (ab==0 or cd==0)
            return 0
        else
            return 1
    else
        return mult(b,d)+(mult(a,d)+mult(b,c))<<(n/2) + mult(a,c)<<n
```

c) As mentioned before 4 multiplications and 5 linear operations. Means 4 children of n/2 size since we split in half and 5 linear operations of Θ(n)
$T(n)=4T(n/2)+5\Theta(n)=$**$4T(n/2)+\Theta(n)$**

$T(1)=1$-base case

d)
We use a recursion tree method: height is $\log_2 n$

$T(n)$	$\Theta(n)$
|	
$4T(n/2)$	$4*\Theta(n/2)$
|	
$16T(n/4)$	$16*\Theta(n/4)$
…..	
$nT(1)$	n-base case

$sum=n+\Theta(n)+4\Theta(n/2)+16\Theta(n/4)...=n+\Theta(n)(1+4+16+...)=\Theta(n)(4^k-1)/3$
$k=\log_2 n$
$\Theta(n)(n^2-1)/3=O(n^3)$- the upper bound for our algorithm
e)
$\log_b a=\log_2 4=2$; $2-1=1 \rightarrow n^2>n$ we have the first case and that means
we have an algorithm that has $\Theta(n^2)$ time complexity which means its upper bound is anything larger than n^2, which is true for n^3, thus confirming our previous discovery. So, point d is still true.