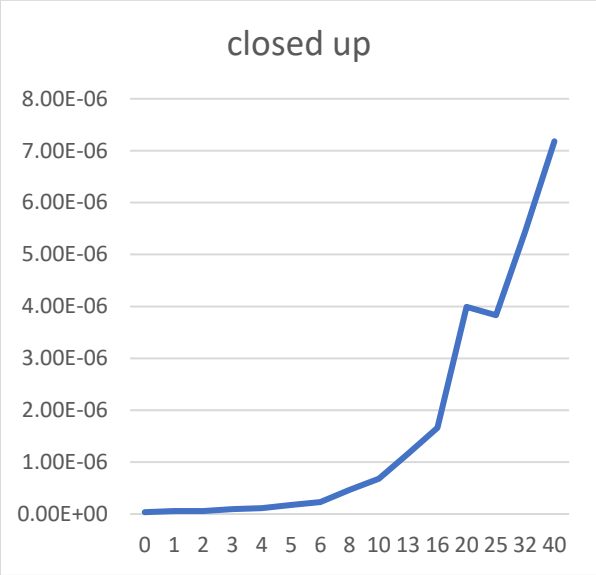
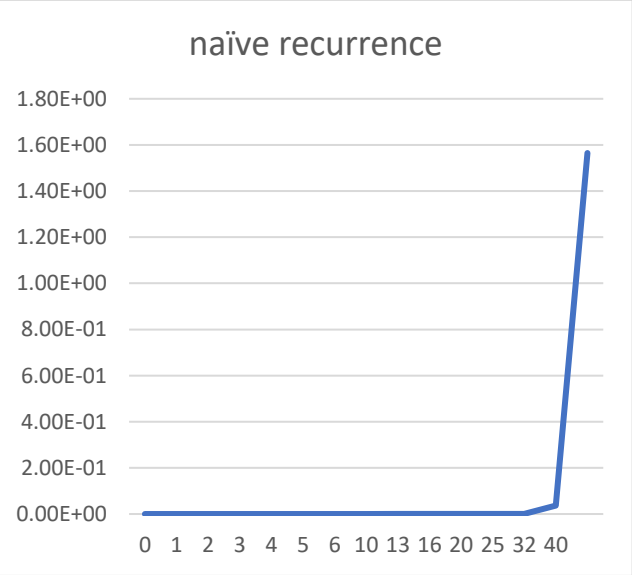
ADS – HW3

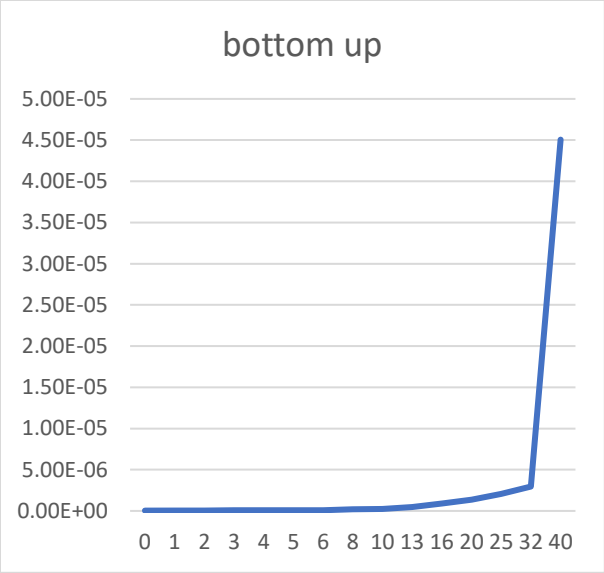
Shorouk Gabr Awwad

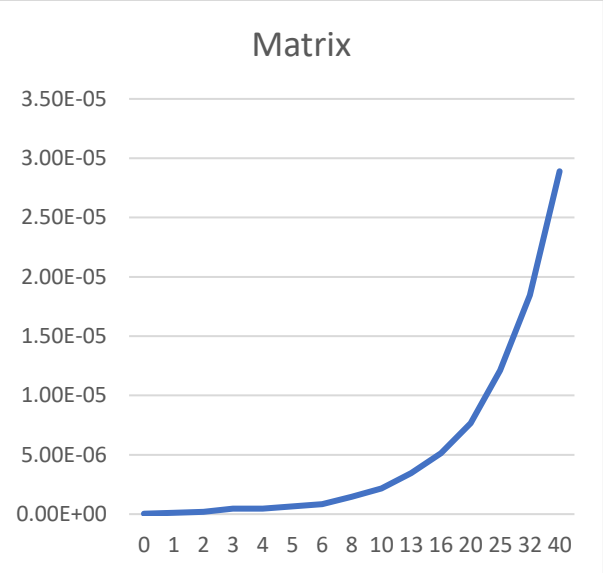
30002030

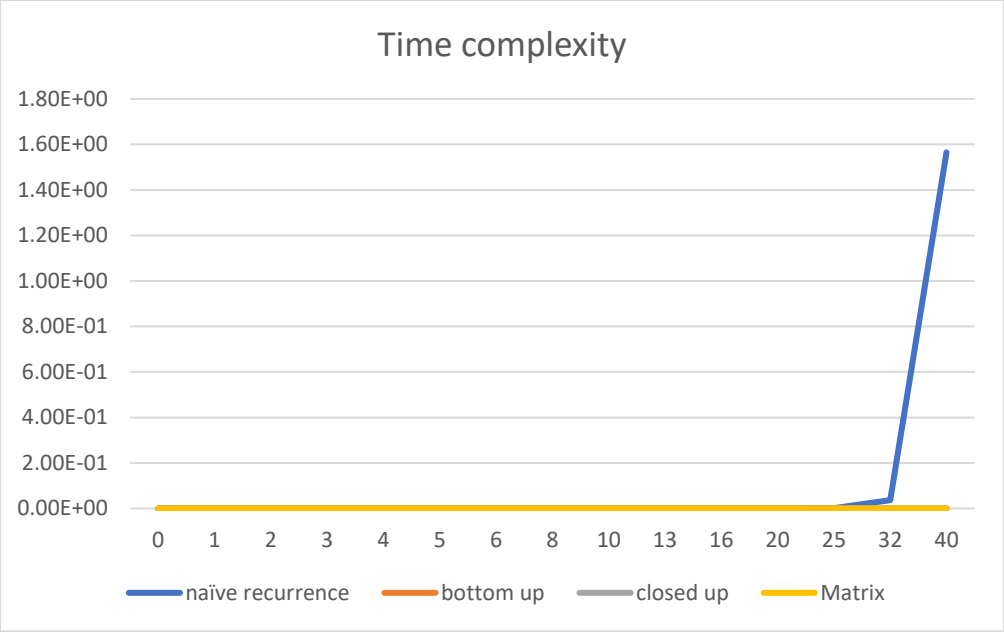
b)


closed up
8.00E-06
7.00E-06
6.00E-06
5.00E-06
4.00E-06
3.00E-06
2.00E-06
1.00E-06
0.00E+00
0 1 2 3 4 5 6 8 10 13 16 20 25 32 40


naïve recurrence
1.80E+00
1.60E+00
1.40E+00
1.20E+00
1.00E+00
8.00E-01
6.00E-01
4.00E-01
2.00E-01
0.00E+00
0 1 2 3 4 5 6 10 13 16 20 25 32 40


bottom up
5.00E-05
4.50E-05
4.00E-05
3.50E-05
3.00E-05
2.50E-05
2.00E-05
1.50E-05
1.00E-05
5.00E-06
0.00E+00
0 1 2 3 4 5 6 8 10 13 16 20 25 32 40

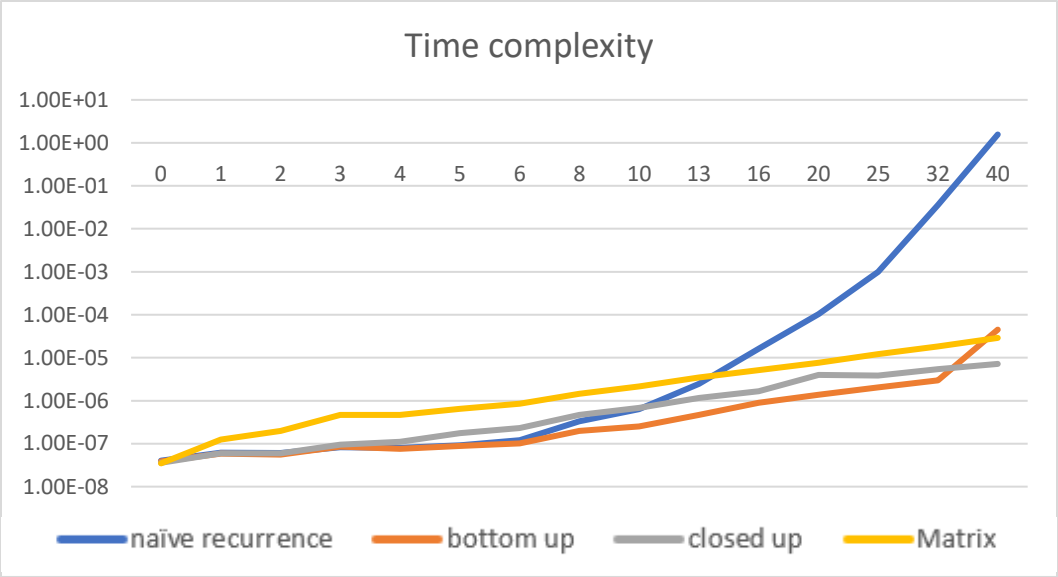

Matrix
3.50E-05
3.00E-05
2.50E-05
2.00E-05
1.50E-05
1.00E-05
5.00E-06
0.00E+00
0 1 2 3 4 5 6 8 10 13 16 20 25 32 40



c)

- No, they are similar; but at the beginning of the implementation of each one of the four implemented algorithms , there was a different start. For example, both of the recurrence and bottom up algorithm were starting with 0,1,2 as seeds (initial values). On the other hand, the closed-up algorithm started with 1,1 and finally the matrix algorithm started with 0,1,1. Which means the four algorithms differ at the beginning before depending on the seed elements, which causes one algorithm to lead the other with one element, but other than that they grow with the same values.
- For the final implementation, since all the algorithms started with the same elements, the subsequent results were always the same.

d)



The time growth rate of the naïve recurrence e algorithm is the greatest among the four algorithms , which means the recurrence is the slowest in terms of time complexity.

For small n, you there was un noticeable difference in time complexity for all the four, but as shown the difference grows for larger n logarithmically.

The fastest algorithm in terms of time complexity is bottom up algorithm.

So we can arrange them according to the time taken:

Naïve recurrence > Matrix algorithm > closed up > bottom up

SECOND QUESTION

a)

we can implement the algorithm of the product of two numbers in the following psudu code:

```
//Multiplication (A[n_digits], B[m_digits])
Multiplication;
        For  i to n
            For j to m
                Multiplication += A[i]*B[j];
Return Multiplication;
```

Calculating the time complexity:

Multiplication (A[n_digits], B[m_digits])	$\Theta(1)$
Multiplication;	$\Theta(1)$
For i to n	$\Theta(n)$
For j to m	$\sum^{n-1}{}_{1}T(m) : \Theta(n^2)$
Multiplication += A[i]*B[j];	$\sum^{n-1}{}_{1}T(m-1) : \Theta(n^2)$
Return Multiplication;	$\Theta(1)$

From the table, we can say that the time complexity for the multiplication process in the brute force implementation = n^2 .

b)

Divide_conquer (a_digits, b_digits)

If (a_digits >1)

A_r = a_digits/2

If (b_digits>1)

B_r=b_digits /2

b_l= b_digits - A_r+1

a_l= a_digits – B_r +1

Z= Divide_conquer (a_l,b_l)

Y= Divide_conquer (a_r+a_l,b_r+b_l)

X=Divide_conquer (a_r,b_r)

Return $X*2^n + (Y- X-Z)*2^{n/2} + X$

c)

Let's assume a_digits = b_digits =n , where n is the number of digits of the entered number. Since n is devided by 2 each iteration and w have three recurrences of n, then the time complexity will be 3T(n/2), when n is equal to 1 we have 2*1=2 (a_digits +b_digits)

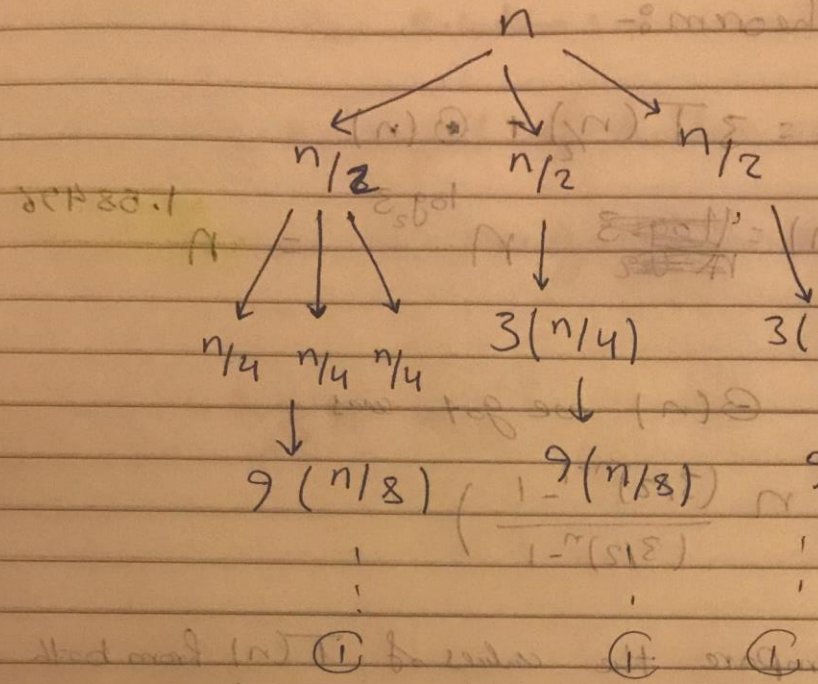
So, the time complexity becomes :

T(n) = 3T(n/2)+f(n)

d)

bject. Date.

d)



the hight $= \log_3 n$

$$T(n) = 3n/2 + 9n/4 + 27n/8 + \cdots$$

$$T(n) = n\left(3/2 + 9/4 + 27/8 + \cdots\right)$$

$$= n \sum_{1}^{n} (3/2)^n$$

$$\sum_{1}^{n} (3/2)^n = \frac{(3/2)^{n+1} - 1}{(3/2)^n - 1}$$

$$T(n) = \Theta\left(n * \frac{(3/2)^{n+1} - 1}{(3/2)^n - 1}\right)$$

$\lim_{n \to \infty}$ the sum diverges

E)

E)

Master theorem :-

$$T(n) = 3T(n/2) + \Theta(n)$$

$$\Theta(n) = n^{\log_2 3} = n^{1.58496}$$

while $\Theta(n)$ we got was

$$T(n) \in \Theta\left(n \frac{(3/2)^{n+1} - 1}{(3/2)^n - 1}\right)$$

if we compare the values of $T(n)$ from both methods for different inputs, we get

n	T(n) master method	T(n) recurrence tree
3	5.7	5.13
5	12.8185	7.879
10	38.456	15.088237

from the table we can tell that

since $T(n) < \Theta(n^{1.58496})$

$T(n) \in O(n^{1.58496})$ ☐

Big O