COMP 3270 Programming Assignment

Names: Liam Maher and Aiden Townsend
Contributions: (All collaborative but main contributions are listed below)
Liam Maher: Implementation of algorithms 1-3 in Jupyter Notebook file.
Aiden Townsend: Implementation of algorithms 4, 5,Theoretical Time Complexity Analysis
For Experiment-I, II, and Matrix-Chain problem: Equal contributions

1. Algorithm_1: there are 3 nested loops in this algorithm, first from i = 1 to p, them from j = 1 to r, then from k = 1 to q this means the 2nd and 3rd loops will execute at least p times, then after the second loop the third loop will iterate at least p*r times, then after the third loop the contents will execute p*r*q times. This would add up to T(n) = 4pr+2prq+2p+1. This would make the complexity of the algorithm O(pqr).

Algorithm 1 step	Cost	Exec	Total
for i = 1 to p	6	p+1	6(p+1)+1
for j = 1 to r	6	p(r+1)	6p(r+1)+1
sum = 0	1	p(r)	pr
for k = 1 to q	6	p(r)(q+1)	6(p(r)(q+1))
sum = sum + A_{ij}*B_{kj}	13	p(r)(q)	13prq
C_{ij} = sum	6	p(r)	6pr
			= 19prq+19pr+12p+8
T(n) = 19prq+19pr+12p+8			

COMPLEXITY ORDER: O(prq)

2. Algorithm_2:

Algorithm 2 step (T=5) *technically they are floor but omitted for simplicity	Cost	Exec	Total
for I = 1 to p in steps of T	7	p/T +1	7(p/T +1)+1
for J = 1 to r in steps of T	7	pr/T^2+p/T	7(pr/T^2+p/T)+1
for K = 1 to q in steps of T	7	pqr/T^3+pr/T^2	7(pqr/T^3+pr/T^2)+1
for i = I to min(I+T, p)	7	(qrp^3)/2T^3+pqr/T^3	7((qrp^3)/2T^3+pqr/T^3)+6
for j = J to min(J+T, r)	7	((qr^3p^3)/2T^3)+(qr^2p^3)/2	7(((qr^3p^3)/2T^3)+(qr^

		T^3)/2+1	2p^3)/2T^3)/2+1)+6
sum = 0	1	((qr^3p^3)/2T^3)+(qr^2p^3)/2 T^3)/2	((qr^3p^3)/2T^3)+(qr^2p ^3)/2T^3)/2
for k = K to min(K+T, q)	7	(((qr^3p^3)/2T^3)+(qr^2p^3)/2 T^3)/2)((q^2+q)/2+1)	7(((qr^3p^3)/2T^3)+(qr^ 2p^3)/2T^3)/2)((q^2+q)/ 2+1)
sum = sum + A_{ij}*B_k	14	(((qr^3p^3)/2T^3)+(qr^2p^3)/2 T^3)/2)((q^2+q)/2)	14(((qr^3p^3)/2T^3)+(qr ^2p^3)/2T^3)/2)((q^2+q)/ 2)
C_{ij} = sum	4	((qr^3p^3)/2T^3)+(qr^2p^3)/2 T^3)/2	4((qr^3p^3)/2T^3)+(qr^2 p^3)/2T^3)/2

COMPLEXITY ORDER:

You are a psychopath for assigning this.

I don't think I can sum all of that together and remain sane. Wolfram Alpha won't take it because it exceeds the character limit. If it's too much for Wolfram it's too much for me, but you can look at the table and see that I got precise time complexities.

O(pqr).

3. Algorithm_3:

Algorithm 3 step	Cost	Exec	Total
Base Case Test	6	1	6
Split Axis if statements	15	1	15
Splitting Matrices	8n^2+5	1	8n^2+5
Recursive Step	4T(n/2)	1	4T(n/2)
Combine results into C	3n^2	1	3n^2

T(n) = 4T(n/2) + 11n^2+26

Recurrence relation: T(n) = { 4T(n/2) + Θ(n^2) n>=8

19prq+19pr+12p+8 n < 8

master method: a= 4, b = 2, k = 2 i = 0 ->

COMPLEXITY ORDER: Θ(n^2 log n)

4. Algorithm_4:

//The Code here used to calculate was not the complete finished code, but it is the same format, just with syntax error fixes. It is almost identical to form in our code

```python
def algorithm_4(A, B, C):
    n = len(A) # 3
    if n == 1: # 2
        C[0][0] = A[0][0] * B[0][0] # 11
    else:
        mid = n // 2 # 3

        # Partition A and B into 4 equal-sized blocks
        A11 = A[:mid, :mid] # 3(n^2/4)
        A12 = A[:mid, mid:] # 3(n^2/4)
        A21 = A[mid:, :mid] # 3(n^2/4)
        A22 = A[mid:, mid:] # 3(n^2/4)

        B11 = B[:mid, :mid] # 3(n^2/4)
        B12 = B[:mid, mid:] # 3(n^2/4)
        B21 = B[mid:, :mid] # 3(n^2/4)
        B22 = B[mid:, mid:] # 3(n^2/4)

        # Compute intermediate matrices using block partitioning
        P1 = algorithm_4(A11, B11) + algorithm_4(A12, B21) # 2T(n/2) + 2
        P2 = algorithm_4(A11, B12) + algorithm_4(A11, B11) # 2T(n/2) + 2
        P3 = algorithm_4(A21, B11) + algorithm_4(A21, B12) # 2T(n/2) + 2
        P4 = algorithm_4(A21, B12) + algorithm_4(A22, B22) # 2T(n/2) + 2

        # Combine intermediate results into the result matrix C
        C[:mid, :mid] = P1 # 3(n^2/4)
        C[:mid, mid:] = P2 # 3(n^2/4)
        C[mid:, :mid] = P3 # 3(n^2/4)
        C[mid:, mid:] = P4 # 3(n^2/4)

    return C
```

3+2+11+3+3(n^2/4)+3(n^2/4)+3(n^2/4)+3(n^2/4)+3(n^2/4)+3(n^2/4)+3(n^2/4)+3(n^2/4)+4[2T(n/2)+2]+3(n^2/4)+3(n^2/4)+3(n^2/4)+3(n^2/4) -> T(n) = 8T(n/2)+9n^2+27

Recurrence Relation: T(n) = { 8T(n/2) + Θ(n^2) n>1 Θ(1) n = 1 (this is 16 but generalized to constant)

COMPLEXITY ORDER: master method a = 8, b = 2, f(n) = n^2 -> Θ(n^3)

5. Algorithm_5:
   //For m1-m7 in this case, recursion method was implemented on a different computer so the comments next to are the times for the recursion method as seen in comments

   T(n)=T(log2)+7T(n/2)+54(n^2/4)+3

recurrence relation: T(n) = 7T(n/2) + n^2 n >= 8 19prq+19pr+12p+8 n < 8

Master Method: a = 7, b = 2, f(n) = n^2 ->

COMPLEXITY ORDER: Theta(n^log2 (7)) -> approx theta(n^2.8)

6. Matrix-Chain-Order:

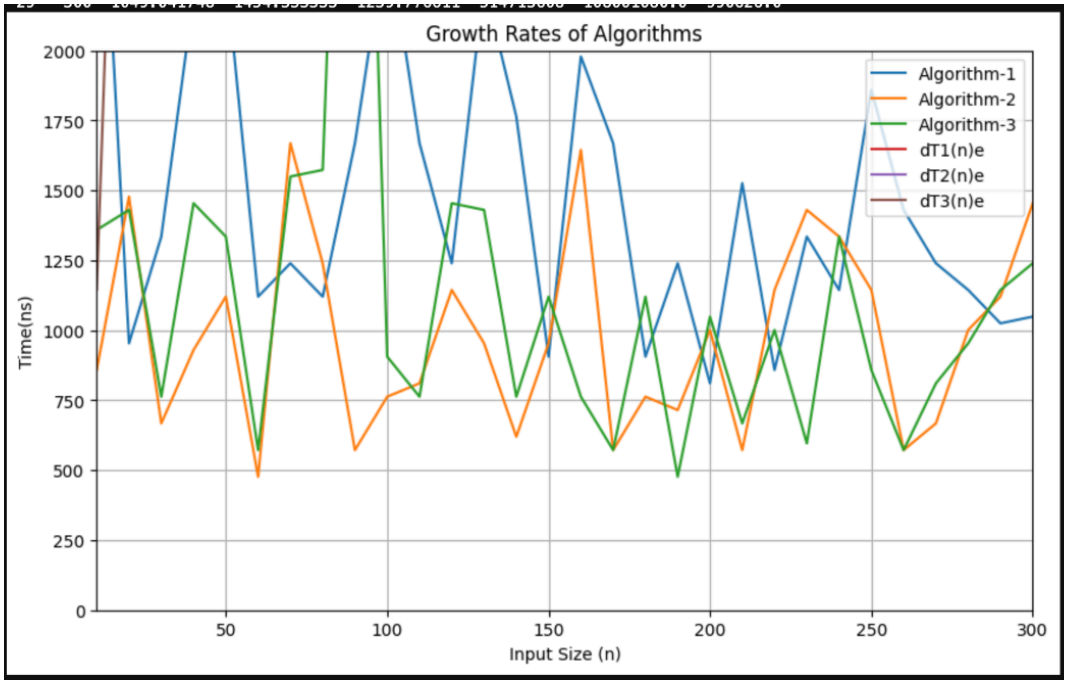
MATRIX-CHAIN-ORDER Step	Cost	Exec	Total
let m[i:n,1:n] and s[1:n-1,2:n] be new tables	2n^2	1	2n^2
for i = 1 to n	5	n+1	5n+6
m[i, i] = 0	6	n+1	6n+6
for l = 2 to n	5	n	5n+1
for i = 1 to n-1 +1	8	(n^2+n)/2+1	4(n^2+n)+9
j=1+l-1	5	(n^2+n)/2	5(n^2+n)/2
m[i,j] = inf	6	(n^2+n)/2	6(n^2+n)/2

for k = 1 to j -1	8	((n^2+n)/2)((n^2+n)/2+1)	8((n^2+n)/2)((n^2+n)/2+1)+1
q = m[i, k] + m[k+1, j] + $p_{i-1}p_kp_j$	24	(n^2+n)/2)((n^2+n)/2)^2	24(n^2+n)/2)((n^2+n)/2)^2
if q < m[i, j]	7	(n^2+n)/2)((n^2+n)/2)^2	7(n^2+n)/2)((n^2+n)/2)^2
m[i,j] = q	6	(n^2+n)/2)((n^2+n)/2)^2	6(n^2+n)/2)((n^2+n)/2)^2
s[i, j] = k	6	(n^2+n)/2)((n^2+n)/2)^2	6(n^2+n)/2)((n^2+n)/2)^2
return m and s	3	1	3

COMPLEXITY ORDER:
T(n)=2n^2+16n+26+35(n^2+n)/2+51((n^2+n)/2)^2
O(n^4)

Experiment 1:

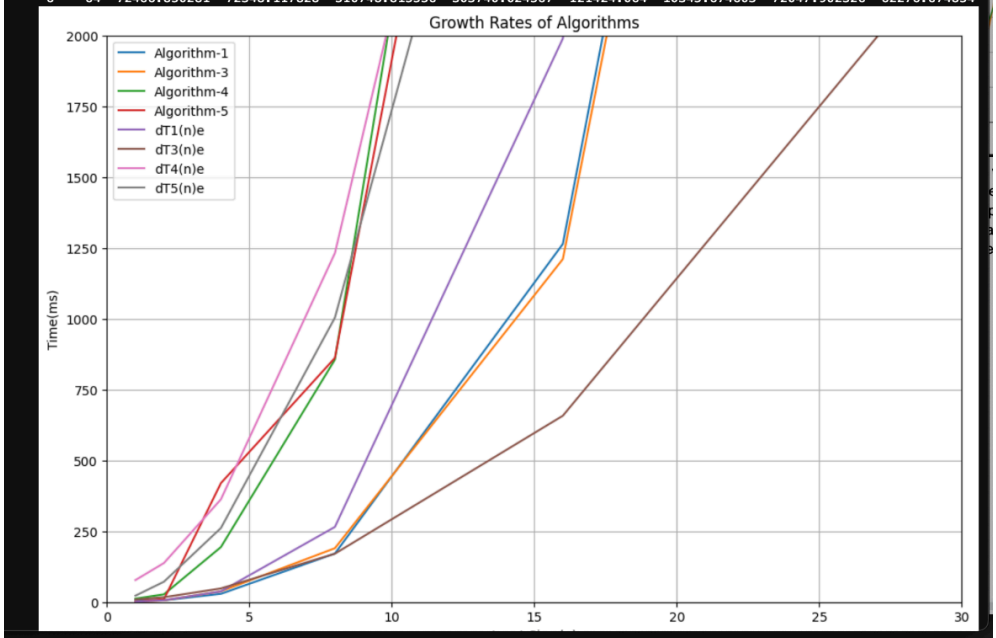
	Size	Algorithm-1	Algorithm-2	Algorithm-3	dT1(n)e	dT2(n)e	dT3(n)e
0	10	3075.599670	858.306885	1358.985901	21028	4036.0	1146.0
1	20	953.674316	1478.195190	1430.511475	159848	32072.0	4466.0
2	30	1335.144043	667.572021	762.939453	530468	108108.0	9986.0
3	40	2193.450928	929.832458	1454.353333	1246888	256144.0	17706.0
4	50	2288.818359	1120.567322	1335.144043	2423108	500180.0	27626.0
5	60	1120.567322	476.837158	572.204590	4173128	864216.0	39746.0
6	70	1239.776611	1668.930054	1549.720764	6610948	1372252.0	54066.0
7	80	1120.567322	1239.776611	1573.562622	9850568	2048288.0	70586.0
8	90	1668.930054	572.204590	4673.004150	14005988	2916324.0	89306.0
9	100	2431.869507	762.939453	905.990601	19191208	4000360.0	110226.0
10	110	1668.930054	810.623169	762.939453	25520228	5324396.0	133346.0
11	120	1239.776611	1144.409180	1454.353333	33107048	6912432.0	158666.0
12	130	2241.134644	953.674316	1430.511475	42065668	8788468.0	186186.0
13	140	1764.297485	619.888306	762.939453	52510088	10976504.0	215906.0
14	150	905.990601	953.674316	1120.567322	64554308	13500540.0	247826.0
15	160	1978.874207	1645.088196	762.939453	78312328	16384576.0	281946.0
16	170	1668.930054	572.204590	572.204590	93898148	19652612.0	318266.0
17	180	905.990601	762.939453	1120.567322	111425768	23328648.0	356786.0
18	190	1239.776611	715.255737	476.837158	131009188	27436684.0	397506.0
19	200	810.623169	1001.358032	1049.041748	152762408	32000720.0	440426.0
20	210	1525.878906	572.204590	667.572021	176799428	37044756.0	485546.0
21	220	858.306885	1144.409180	1001.358032	203234248	42592792.0	532866.0
22	230	1335.144043	1430.511475	596.046448	232180868	48668828.0	582386.0
23	240	1144.409180	1335.144043	1335.144043	263753288	55296864.0	634106.0
24	250	1859.664917	1144.409180	858.306885	298065508	62500900.0	688026.0
25	260	1430.511475	572.204590	572.204590	335231528	70304936.0	744146.0
26	270	1239.776611	667.572021	810.623169	375365348	78732972.0	802466.0
27	280	1144.409180	1001.358032	953.674316	418580968	87809008.0	862986.0
28	290	1025.199890	1120.567322	1144.409180	464992388	97557044.0	925706.0
29	300	1049.041748	1454.353333	1239.776611	514713608	108001080.0	990626.0



The graphs and chart here were very surprising to us. As we can see, Algorithm 1 usually performs the worst, over 2 and 3. Next, algorithm 3 seemed to be about the same but slightly worse at times than algorithm 2, but Algorithm 2 had less peaks where it would get much worse. The empirical time almost didnt correlate at all, you can barely see the lines in the left corner of the graph, and the algorithms seemed to not follow the predicted complexity, performing much better than anticipated

Experiment 2:

	Size	Algorithm-1	Algorithm-3	Algorithm-4	Algorithm-5	dT1(n)e	dT3(n)e	dT4(n)e	dT5(n)e
0	1	8.106232	1.192093	11.920929	1.192093	1.392	8.924603	77.581395	22.432749
1	2	5.960464	5.960464	27.179718	10.013580	6.240	16.933862	137.706977	71.784795
2	4	29.087067	37.908554	194.072723	419.855118	37.824	48.055556	362.693023	261.341520
3	8	171.184540	190.019608	855.922699	861.883163	265.152	170.711640	1231.604651	1003.865497
4	16	1264.095306	1210.689545	5782.127380	5076.885223	1989.312	657.674603	4645.186047	3942.555556
5	32	9632.110596	9417.772293	41078.090668	37913.084030	15418.560	2598.203704	18175.381395	15634.504094
6	64	72466.850281	72348.117828	310748.815536	303740.024567	121424.064	10345.674603	72047.902326	62276.674854



In experiment 2, the algorithms seemed to fit much better with their predicted times. In most of the trials we ran, Algorithm 5 performed slightly better than algorithm 4, and the theoretical vs actual values of all of the algorithms seem to fit very nicely, besides algorithm 3 performing worse. Unlike Experiment 1, here we can clearly see an increase in time as the input size increases. However, we were unable to complete the matrices up to 2^9 as the program would time out every single time.

Matrix Chain Problem: