Date: 6th September, 2021

Design and Analysis of Algorithm Lab

2019BTECS00058 Devang
Batch: T7

Assignment: Week 4
Divide and Conquer Strategy Part 2

Q) Strassen's Matrix Multiplication.

A) Implement the Naive method to multiply 2 matrices and justify the $O(n^3)$ complexity.

Algorithm:

1. We implement the Brute Force algorithm
2. We check if the matrix multiplication is possible. (a*b) X (b*c) form
3. We then perform individual multiplication for every term, adding all cross-multiples and inserting the value in a new matrix - the final result

Program:

```
# Naive method for matrix multiplication

m1 = [[2, 3, 1], [2, -7, 4]]
m2 = [[3, 4, 5], [1, 1, 4], [2, 1, 4]]

def matrixMultiply(m1, m2):
    if len(m1[0]) != len(m2):
        print("Matrix Multiplication is not possible for this matrix.")
        return []
    res = [[0 for i in range(len(m2[0]))] for j in range(len(m1))]
```

```python
    for i in range(len(m1)):
        for j in range(len(m1[0])):
            for k in range(len(m1[0])):
                res[i][j] += m1[i][k]*m2[k][j]
    return res

ans = matrixMultiply(m1, m2)
for i in ans:
    for j in i:
        print(j, end=' ')
    print()
```

Output:

```python
m1 = [[2, 3, 1], [2, -7, 4]]
m2 = [[3, 4, 5], [1, 1, 4], [2, 1, 4]]

def matrixMultiply(m1, m2):
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4> py .\Q1a.py
11 12 26
7 5 -2
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4>
```

```python
m1 = [[3, 4]]
m2 = [[3, 1], [5, 6]]

def matrixMultiply(m1, m2):
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4> py .\Q1a.py
29 27
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4>
```

Complexity of proposed algorithm (Time & Space):
Space Complexity: O(N)
Time Complexity: $O(N^3)$

Your comment (How is your solution optimal?)
The approach is intuitive but inefficient. The solution is not the most optimal.

B) Implement the Divide and Conquer method to multiply 2 matrices and justify the $O(n^3)$ complexity.

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}_{C}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2

Algorithm:

1. We use recursive Divide and Conquer by dividing the matrix into 4 quarters and solve for each and merge
2. Base case is when we have a quarter with 1 element where we return the multiplied value
3. We then recursively multiply as in the figure C and take the result by combining

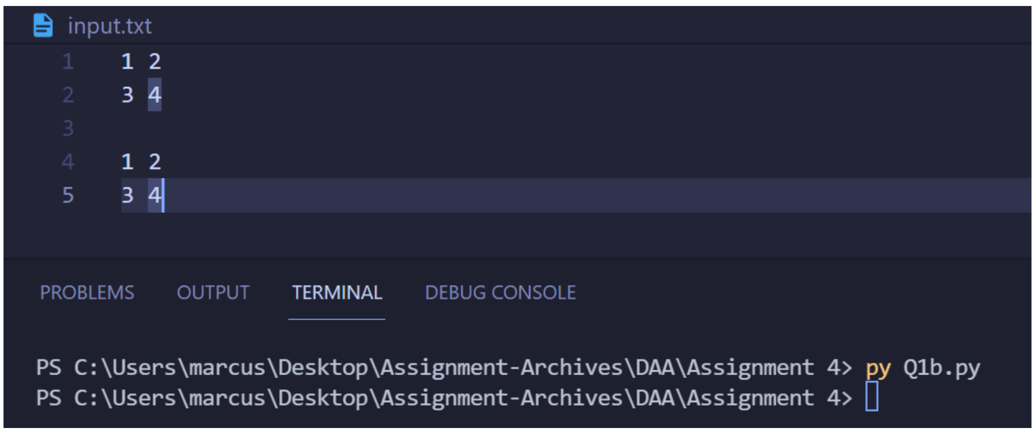
Program:

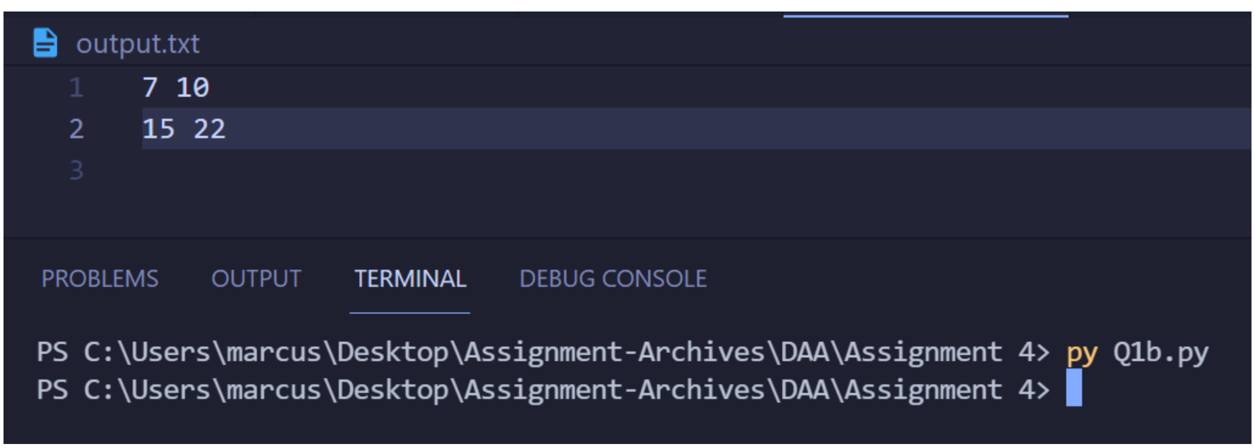
```
import numpy as np

def read_input(input):
```

```
    array = np.loadtxt(input,dtype='i',delimiter=' ')

    array_first,array_second = np.split(array,2,axis=0)
    return array_first, array_second

def save_ouput(output):
    output_array = np.savetxt("output.txt",output.astype(int), fmt='%i',
delimiter=' ')

def divide_and_conquer(array_first,array_second):
    n = len(array_first)
    if n == 1:
        return int(array_first * array_second)
    else:
        a11 =
array_first[:int(len(array_first)/2),:int(len(array_first)/2)]
        a12 =
array_first[:int(len(array_first)/2),int(len(array_first)/2):]
        a21 =
array_first[int(len(array_first)/2):,:int(len(array_first)/2)]
        a22 =
array_first[int(len(array_first)/2):,int(len(array_first)/2):]

        b11 =
array_second[:int(len(array_second)/2),:int(len(array_second)/2)]
        b12 =
array_second[:int(len(array_second)/2),int(len(array_second)/2):]
        b21 =
array_second[int(len(array_second)/2):,:int(len(array_second)/2)]
        b22 =
array_second[int(len(array_second)/2):,int(len(array_second)/2):]

        c11 = divide_and_conquer(a11,b11) + divide_and_conquer(a12,b21)
        c12 = divide_and_conquer(a11,b12) + divide_and_conquer(a12,b22)
        c21 = divide_and_conquer(a21,b11) + divide_and_conquer(a22,b21)
        c22 = divide_and_conquer(a21,b12) + divide_and_conquer(a22,b22)

        result = np.zeros((n,n))
        result[:int(len(result)/2),:int(len(result)/2)] = c11
        result[:int(len(result)/2),int(len(result)/2):] = c12
```

```python
        result[int(len(result)/2):,:int(len(result)/2)] = c21
        result[int(len(result)/2):,int(len(result)/2):] = c22
    return result

if __name__ == "__main__":
    array_first,array_second = read_input('input.txt')
    output = divide_and_conquer(array_first,array_second)
    save_ouput(output)
```

Output:





```
input.txt
1   5 2 6 1
2   0 6 2 0
3   3 8 1 4
4   1 8 5 6
5
6   7 5 8 0
7   1 8 2 6
8   9 4 3 8
9   5 3 7 9

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4> py Q1b.py
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4>
```

```
output.txt
1   96 68 69 69
2   24 56 18 52
3   58 95 71 92
4   90 107 81 142
5

PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4> py Q1b.py
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4>
```

Complexity of proposed algorithm (Time & Space):
Space Complexity: O(N)
Time Complexity: $O(N^3)$

Your comment (How is your solution optimal?)
The approach uses divide and conquer but is still inefficient - we do 8 multiplications for matrices of size N/2 x N/2 and 4 additions. Addition of two matrices takes $O(N^2)$ time. The algorithm is still inefficient with respect to time.

C) Implement the Strassen's Matrix Multiplication and justify the complexity of $O(N^{2.8})$

p1 = a(f - h)
p2 = (a + b)h
p3 = (c + d)e
p4 = d(g - e)
p5 = (a + d)(e + h)
p6 = (b - d)(g + h)
p7 = (a - c)(e + f)

The A x B can be calculated using above seven multiplications.
Following are values of four sub-matrices of result C

$$\underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}_{A} \times \underbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}_{B} = \underbrace{\begin{bmatrix} p5 + p4 - p2 + p6 & p1 + p2 \\ p3 + p4 & p1 + p5 - p3 - p7 \end{bmatrix}}_{C}$$

A, B and C are square metrices of size N x N
a, b, c and d are submatrices of A, of size N/2 x N/2
e, f, g and h are submatrices of B, of size N/2 x N/2
p1, p2, p3, p4, p5, p6 and p7 are submatrices of size N/2 x N/2

Algorithm:

1. We shall be using an optimised version of the Divide and Conquer approach
2. For the 2 matrices, the base case would be when one quarter comes to be the size of 1 - we return the multiplication of the two
3. Further, we recursively determine the values for p1, p2, p3, …, p7
4. We then determine the values of c11, c12, c21, c22 - the quarters of the result matrix
5. Finally, we combine the 4 matrices to get the final answer

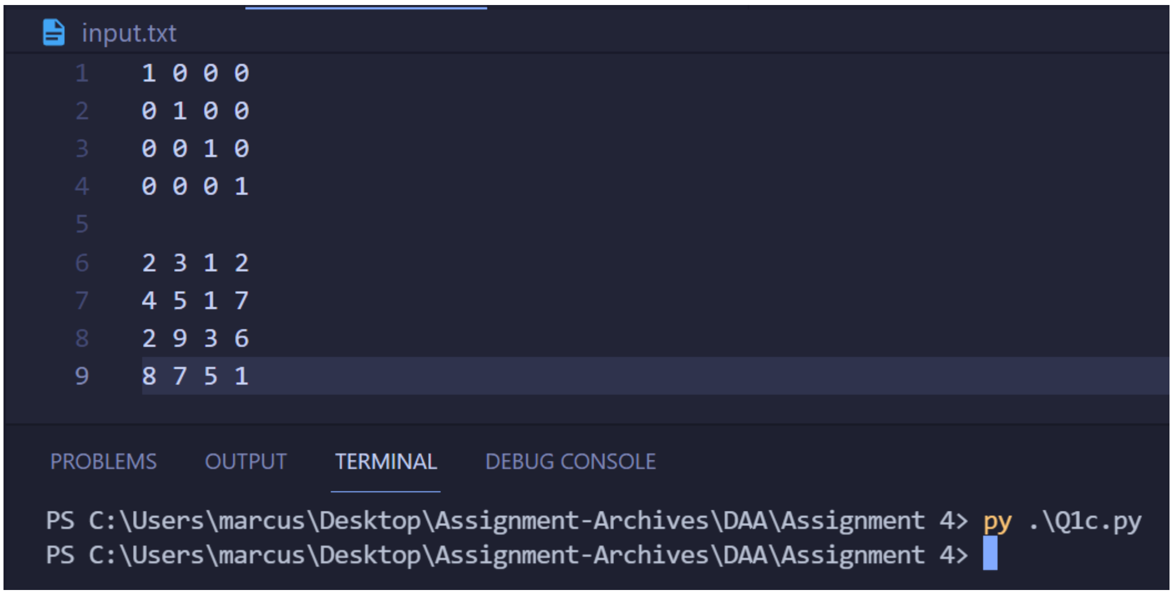
Program:

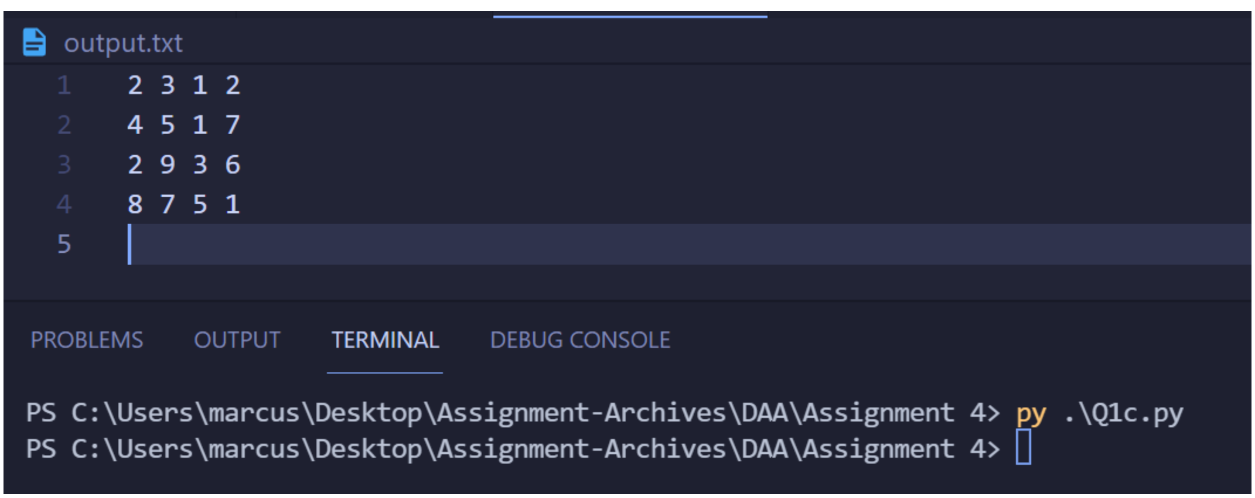
```
import numpy as np

def read_input(input):
    array = np.loadtxt(input,dtype='i',delimiter=' ')
    array_first,array_second = np.split(array,2,axis=0)
```

```python
    return array_first, array_second

def save_ouput(output):
    output_array = np.savetxt("output.txt",output.astype(int), fmt='%i',
delimiter=' ')

def splitMatrix(matrix):
    # Split matrix into quarters
    row, col = matrix.shape
    row2, col2 = row//2, col//2
    return matrix[:row2, :col2], matrix[:row2, col2:], matrix[row2:,
:col2], matrix[row2:, col2:]

def matrixMultiplicationStrassen(m1, m2):
    # Base case
    if len(m1) == 1:
        return m1 * m2

    a, b, c, d = splitMatrix(m1)
    e, f, g, h = splitMatrix(m2)

    p1 = matrixMultiplicationStrassen(a, f - h)
    p2 = matrixMultiplicationStrassen(a + b, h)
    p3 = matrixMultiplicationStrassen(c + d, e)
    p4 = matrixMultiplicationStrassen(d, g - e)
    p5 = matrixMultiplicationStrassen(a + d, e + h)
    p6 = matrixMultiplicationStrassen(b - d, g + h)
    p7 = matrixMultiplicationStrassen(a - c, e + f)

    # From the diagram
    c11 = p5 + p4 - p2 + p6
    c12 = p1 + p2
    c21 = p3 + p4
    c22 = p1 + p5 - p3 - p7

    # Combining the quadrants
    c = np.vstack((np.hstack((c11, c12)), np.hstack((c21, c22))))

    return c
```

```
array_first,array_second = read_input('input.txt')
ans = matrixMultiplicationStrassen(array_first, array_second)
save_ouput(ans)
```

Output:





input.txt

```
1 7
2 4

3 3
5 2
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4> py .\Q1c.py
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4>
```

output.txt

```
38 17
26 14
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4> py .\Q1c.py
PS C:\Users\marcus\Desktop\Assignment-Archives\DAA\Assignment 4>
```

Complexity of proposed algorithm (Time & Space):
Space Complexity: $O(N)$
Time Complexity: $O(N^{2.8})$

Your comment (How is your solution optimal?)
The approach uses modified divide and conquer and makes use of the pre-computed 7 values and recursively solve the problem. The complexity is $T(N) = 7T(N/2) + O(N^2)$. This approach is not frequently used due to the heavy constraints and it doesn't provide much advantage over brute force.