NANYANG TECHNOLOGICAL UNIVERSITY

SEMESTER II EXAMINATION 2017-2018 SUGGESTED SOLUTION

MH1402 – Algorithms & Computing II

MAY 2018 TIME ALLOWED: 2 HOURS

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **FOUR (4)** questions and comprises **FIVE (5)** printed pages.
2. Answer **all** questions. The marks for each question are indicated at the beginning of each question.
3. Answer each question beginning on a **FRESH** page of the answer book.
4. This **IS NOT** an **OPEN BOOK** exam.

Solutions provided by: Brandon Goh – bgoh008@e.ntu.edu.sg

Question 1.

(a) Show by mathematical induction that $T(n) = 2^{n+1} - 1$, given the following definition: **(6 marks)**

$$T(n) = \begin{cases} 1 & \text{if } n = 0, \\ T(n-1) + 2^n & \text{otherwise.} \end{cases}$$

(b) What is the complexity of the following piece of Python code in Big-Oh notation? **5 marks**

```
sum = 0
for i in range(n**2):
    for j in range(i):
        sum += j
```

Answer

(a) Take the LHS as the given equation and RHS as $T(n) = 2^{n+1} - 1$.

Base Case $n = 0$

(LHS) $T(0) = 1$
(RHS) $T(0) = 2^1 - 1 = 1$

LHS=RHS, $\therefore$ Base case is true.

Hypothesis $n = k$

Assume that $n = k$ is true, then

$$T(k) = T(k-1) + 2^k = 2^{k+1} - 1$$

Induction $n = k + 1$

To prove that $n = k + 1$ is true, LHS must be equal to RHS.

$$\begin{aligned}
\text{(RHS) } T(k+1) &= 2^{k+2} - 1 \\
\text{(LHS) } T(k+1) &= T(k) + 2^k \\
&= (2^{k+1} - 1) + 2^{k+1} \\
&= 2 \cdot 2^{k+1} - 1 \\
&= 2^{k+2} - 1 \\
&= \text{(RHS)}
\end{aligned}$$

From the above, $n = 0$ and $n = k$ are true
$\Rightarrow n = k + 1$ is also true. Then by mathematical induction, the equation is true for all $n \in \mathbb{N}_0$. $\square$

(b) Break the question into 2 blocks:

Block 1:
`sum = 0` $\rightarrow$ 1 operation.

Block 2:
```
for i in range(n**2):
   for j in range(i):
      sum += j
```

It is important to note that `sum += j` constitutes as 1 operation per execution.

Start by realising that n is fixed.

(1) When $i = 0$, there is no execution of the **inner** loop.

(2) When $i = 1$, $j = 0$ only $\Rightarrow$ inner loop is executed **once**

(3) When $i = 2$, $j \in \{0, 1\} \Rightarrow$ inner loop is executed **twice**

(4) $\vdots$

(5) When $i = n^2 - 1$, $j \in \{0, 1, \cdots, n^2 - 2\} \Rightarrow$ inner loop is executed $n^2 - 1$ times.

Total number of executions: $\sum_{x=0}^{n^2-2} x = \frac{(n^2-2)(n^2-1)}{2} = O(n^4)$

The loop gives us a time complexity of $O(n^4)$, so the addition of the 1 operation from outside the loop ($O(1)$) will not change the time complexity of the algorithm. □

Question 2. Given the following binary search tree

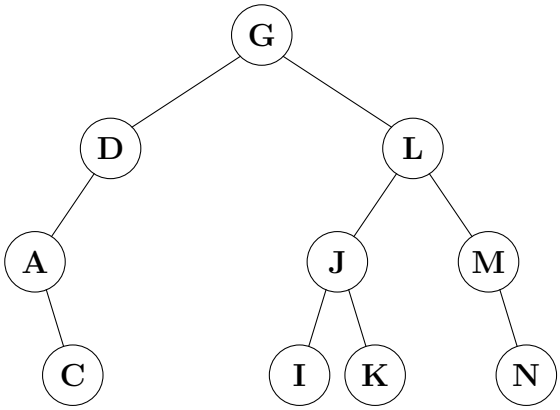


Figure 1: Binary Search Tree

(a) What is the sequence of nodes by Euler's Traversal? **(8 marks)**

(b) Draw the resultant tree after removing node "L". **(6 marks)**

Answer

(a) $G \to D \to A \to C \to A \to D \to G \to L \to J \to I \to J \to K \to J \to L \to M \to N \to M \to L \to G$ □

(b) Step 1: L has 2 child nodes, so the key must be swapped with its **successor**.

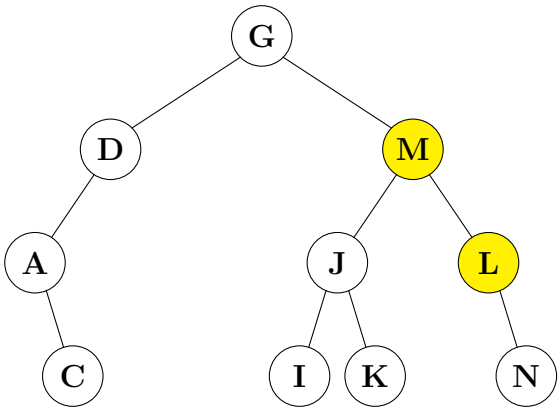


Figure 2: Binary Search Tree After Step 1

Step 2: L now has 1 child node, so swap the key with the child.

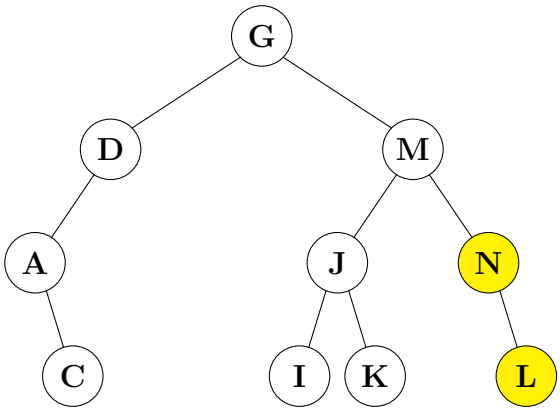


Figure 3: Binary Search Tree After Step 2

Step 3: L has no child nodes, so remove the node from the BST.

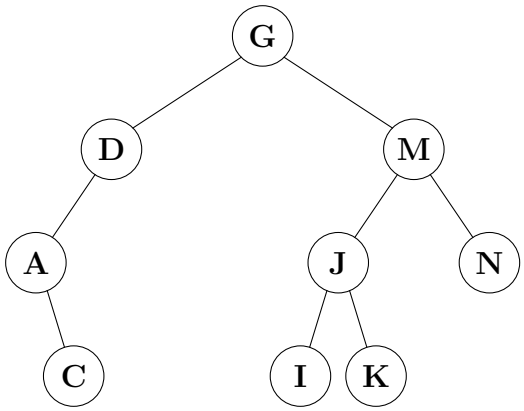


Figure 4: Final Binary Search Tree

□

Question 3. Given the node class implementation below:

```
class Node:
    def __init__(self, initElement):
        self.left = None              #left child
        self.right = None             #right child
        self.parent = None            #parent
        self.element = initElement
```

write in Python a *function* to print the element of all nodes of a binary tree in the order of depth, i.e., the node of depth 0 first (the root), followed by all the nodes of depth 1 (children of the root), then all the nodes of depth 2 (grandchildren of the root) ..., until all the nodes are printed. The function takes the root node of the binary tree as input. You may assume the Queue class is available for use, including functions like:

- Queue(): instantiate an empty queue,
- enqueue(Node x): enqueue node x,
- dequeue(): dequeue and returns the node removed from queue,
- isEmpty(): returns *True* if queue is empty, *False* otherwise.

Note the input binary tree may or may not be a proper binary tree.

(12 marks)

Answer

Since the question only requires us to print the nodes in order without any particular formatting, the code below satisfies the requirements.

```
def TreeOrder(root):
    A=Queue()                     #Create empty queue A
    A.enqueue(root)               #Queue the root node
    while not A.isEmpty():        #The queue is not-empty,
                                  #so there are nodes that
                                  #still need to be processed
```

```
        P=A.dequeue()              #Remove node for processing
        print(P.element)           #Print element in the node
        if P.left!=None:
            A.enqueue(P.left)      #Queue left child node if exists
        if P.right!=None:
            A.enqueue(P.right)     #Queue right child node if exists
    return
```

□

Question 4. Consider the list L = [1, 4, 2, 8, 6, 3, 5].

(a) Apply merge-sort algorithm on the list L. Explain with a binary tree the successive recursive calls, and another binary tree the successive merge processes. Assume the left part takes half or one element less, i.e. $\lfloor n/2 \rfloor$ when dividing a sub-problem of size n. **(12 marks)**

(b) Apply quick-sort algorithm on the original list L. Explain with a binary tree the sort process and underline the pivots. Assume it is always the second element of current sub-problem selected as the pivot. **(12 marks)**

Answer

(a) First Binary Tree for Successive Recursive Calls

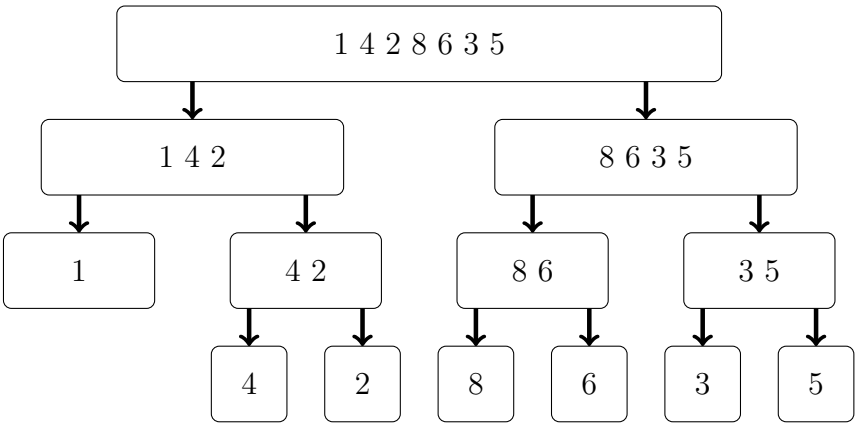


Figure 5: Binary Tree for Mergesort (Splitting)

Second Binary Tree for Sorting

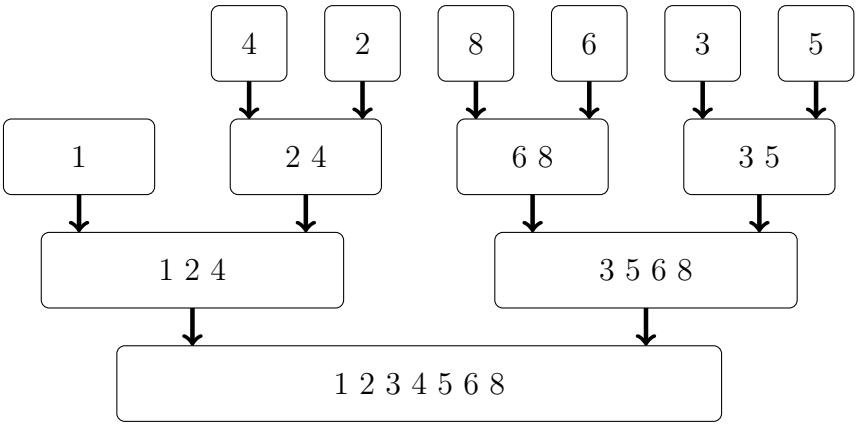


Figure 6: Binary Tree for Mergesort (Merging)

(b) Binary Tree for quicksort

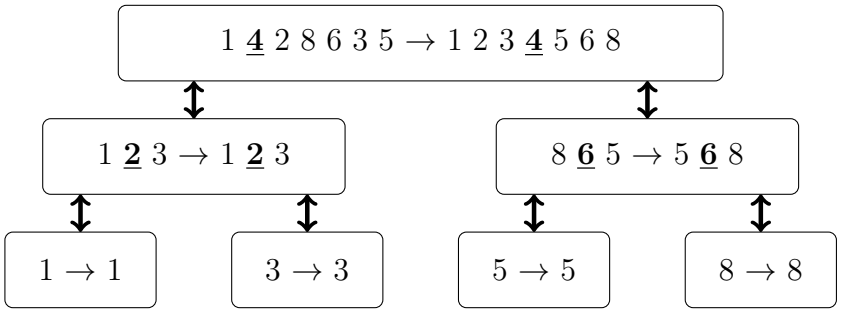


Figure 7: Binary Tree for Quicksort

Step (1)
Identify pivot

Step (2)
Split the values into the left and right subtrees respectively depending on whether the values are less than (<) or greater than (>) the pivot (while preserving the order).

Step (3)
Arrays of length 1 are already sorted (trivial).

Step (4)
Combine after obtaining the results from the child nodes. □

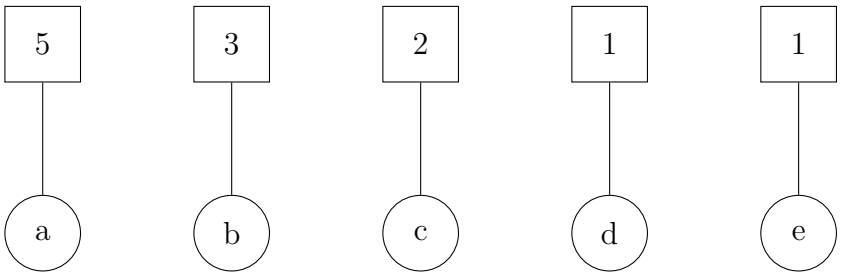
Question 5. Give the frequency table, Huffman tree, and the resulting table of code words of all characters for the string "aabaabccdbea". Note that subtree with more nodes is always placed on the right when joining two subtrees. **(15 marks)**

Answer

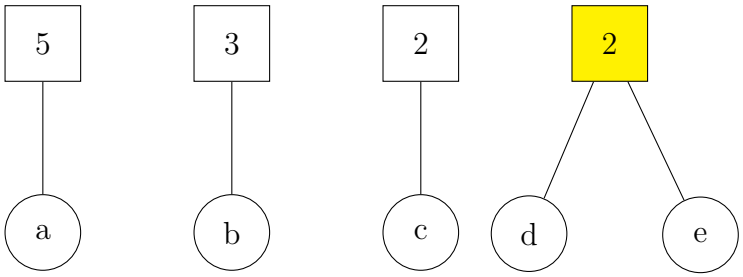
Frequency table:

a	b	c	d	e
5	3	2	1	1

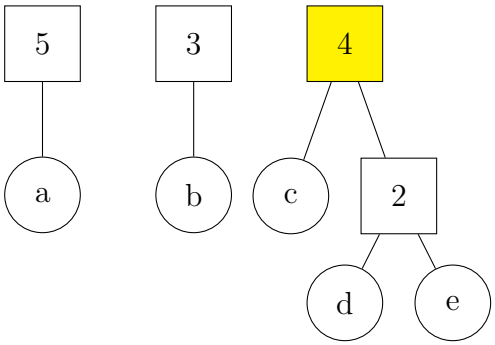
When drawn in its node form:



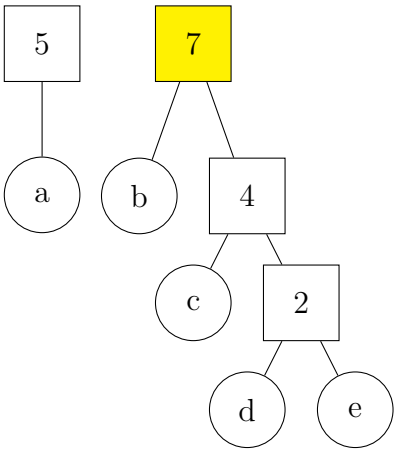
When computing the Huffman Tree, always combine the nodes of letters (or data) with the two lowest frequencies. In this case, d and e both have the lowest frequency of 1.



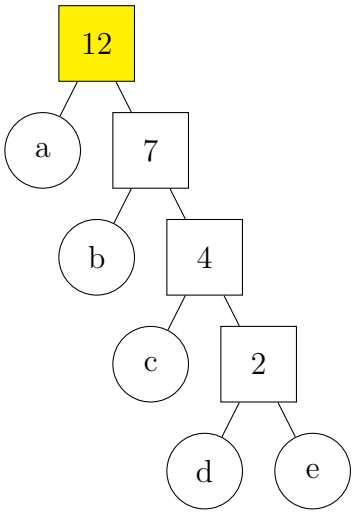
Now, c & (d & e) have the lowest frequencies. Therefore, we will combine these two branches.



Again, b & (c & (d & e)) have the lowest frequencies. Note that b has 3 nodes while (c & (d & e)) has 4 nodes. As such, b is on the left subtree.



Finally, a & (b & (c & (d & e))) are the last two roots of their respective subtrees. Both subtrees can be merged together, with a on the left subtree.



To determine the table of codewords, you must refer to the Huffman Tree. The codeword for each left edge is '0', while the right edge is '1'. For example, to get to the letter 'c', you must go down the edges in this order: right $\rightarrow$ right $\rightarrow$ left. Therefore, the codeword for $c \rightarrowtail 110$. Repeat for all characters and you will get the following codeword table.

a	b	c	d	e
0	10	110	1110	1111

□

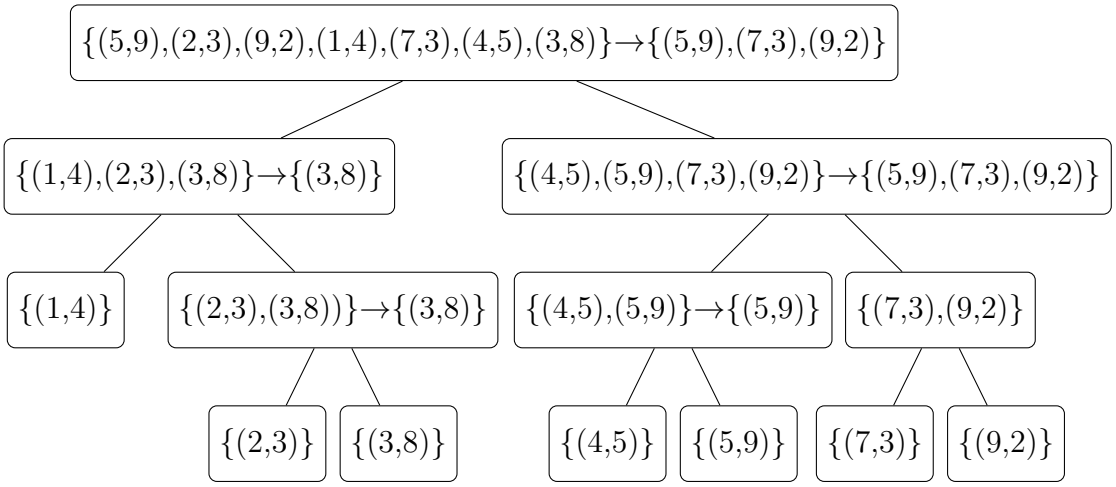
Question 6. What is the maximal set from the following set of points? Show your steps using a binary tree, that arises from the method of divide-and-conquer.

$$\{(5,9),(2,3),(9,2),(1,4),(7,3),(4,5),(3,8)\}$$

(12 marks)

Answer

Assume that the left subset will have $\lfloor n/2 \rfloor$ points. Using the first values as a pivot, (4,5) will act as the median when the list has been sorted.



Step (1)
Sort and identify median

Step (2)
Split the values into the left and right subtrees respectively depending on whether the 1^{st} values are less than ($<$) or greater than ($>$) the median.

Step (3)
No operations are required on sets of length 1 (trivial).

Step (4)
Combine after you have obtained all leaf nodes, removing all (x_i, y_i) where $(x_i, y_i) < (x_j, y_j)$, $i < j$.

Therefore, the maximal set is determined to be the following: {(5,9),(7,3),(9,2)}
□

Question 7. Let A, B, C, D be matrices with the following dimensions:

$$A : 2 \times 2 \quad B : 2 \times 1 \quad C : 1 \times 8 \quad D : 8 \times 8$$

What is the minimum number of item-wise multiplications required to compute the product $A * B * C * D$? Provide a parenthesization that achieves this minimum. **(12 marks)**

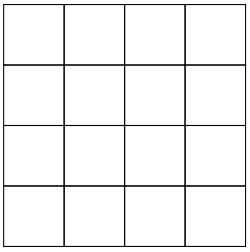
Answer

When performing the *Matrix Chain Algorithm*, we must identify the number of matrices that need to undergo multiplication. In this case, we have 4 matrices. As such, we have $\{d_0, \cdots, d_4\}$. d_i represents the the number of rows/columns of the respective matrix.

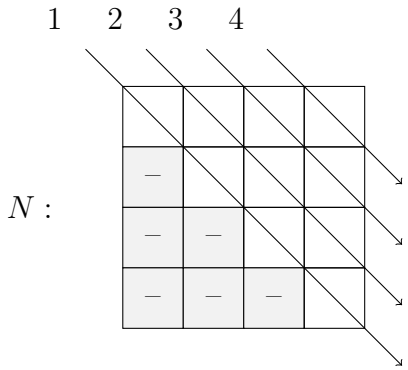
$$\therefore d_0 = 2 \quad d_1 = 2 \quad d_2 = 1 \quad d_3 = 8 \quad d_4 = 8$$

Furthermore, a 4×4 matrix N is created to illustrate the operations that need to be performed.

N :



The order of obtaining the values involves using of **diagonals** as the guides. Tha main diagonal must be calculated first and moves subsequently towards the **right** when completed. i.e. Obtaining values for the matrix must strictly follow this order.



The first part of the algorithm requires that the first diagonal be set to 0. i.e. All $N_{i,i} = 0$ for $0 \leq i <$ (Number of matrices).

$$N_{0,0} = 0$$
$$N_{1,1} = 0$$
$$N_{2,2} = 0$$
$$N_{3,3} = 0$$

0			
–	0		
–	–	0	
–	–	–	0

Calculating all other matrix index values requires the use of the formula:

$$N_{i,j} = \min_{i \leq k < j} \{N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1}\}$$

Following the numbered diagonals, the remaining indexes are worked out in order.

(Second diagonal)

$$
\begin{aligned}
N_{0,1} \Rightarrow i=0, j=1 \Rightarrow k=0 \Rightarrow & \min_{0\le k<1}\{N_{0,0}+N_{1,1}+d_0d_1d_2\} \\
&= \min_{0\le k<1}\{0+0+4\} \\
&= 4 \\
N_{1,2} \Rightarrow i=1, j=2 \Rightarrow k=1 \Rightarrow & \min_{1\le k<2}\{N_{1,1}+N_{2,2}+d_1d_2d_3\} \\
&= \min_{1\le k<2}\{0+0+16\} \\
&= 16 \\
N_{2,3} \Rightarrow i=2, j=3 \Rightarrow k=2 \Rightarrow & \min_{2\le k<3}\{N_{2,2}+N_{3,3}+d_2d_3d_4\} \\
&= \min_{2\le k<3}\{0+0+64\} \\
&= 64
\end{aligned}
$$

0	4		
–	0	16	
–	–	0	64
–	–	–	0

(Third diagonal)

$$
\begin{aligned}
N_{0,2} \Rightarrow i=0, j=2 \Rightarrow k=\{0,1\} \Rightarrow & \min_{0\le k<2}\{N_{0,0}+N_{1,2}+d_0d_1d_3, N_{0,1}+N_{2,2}+d_0d_2d_3\} \\
&= \min_{0\le k<2}\{0+16+32, 4+0+16\} \\
&= 20 \\
N_{1,3} \Rightarrow i=1, j=2 \Rightarrow k=\{1,2\} \Rightarrow & \min_{1\le k<3}\{N_{1,1}+N_{2,3}+d_1d_2d_4, N_{1,2}+N_{3,3}+d_1d_3d_4\} \\
&= \min_{1\le k<3}\{0+64+16, 16+0+128\} \\
&= 80
\end{aligned}
$$

0	4	20	
–	0	16	80
–	–	0	64
–	–	–	0

(Last diagonal)

$$
\begin{aligned}
N_{0,3} &\Rightarrow i = 0, j = 3 \Rightarrow k = \{0, 1, 2\} \\
&\Rightarrow \min_{0 \le k < 3} \{N_{0,0} + N_{1,3} + d_0 d_1 d_4, N_{0,1} + N_{2,3} + d_0 d_2 d_4, N_{0,2} + N_{3,3} + d_0 d_3 d_4\} \\
&= \min_{0 \le k < 3} \{0 + 80 + 32, 4 + 64 + 16, 20 + 0 + 128\} \\
&= 84
\end{aligned}
$$

0	4	20	84
–	0	16	80
–	–	0	64
–	–	–	0

From $N_{0,3}$ (most top right box), we are able to tell that the minimum number of item-wise multiplications required is 84.

To determine the parenthesization for the least number of item-wise multiplications, we need to work backwards. $N_{0,3}$ is obtained from $N_{0,1}+N_{2,3}+d_0 d_2 d_4$, implying that matrices 0 to 1 are enclosed within a single bracket. i.e. $(A * B) * C * D$. No further analysis of paranthesization of C and D are required since there are only two matrices. Similarly, we also see that matrices 2 to 3 are enclosed within a single bracket. i.e. $(A * B) * (C * D)$.

Note that when looking at the term $N_{i,j}$, it means that matrices i to j inclusive are enclosed within 1 bracket and should be checked to determine whether further paranthesization is required.

The answer for this question is 84 item-wise multiplications required and the respective paranthesization that gives the solution is this: $(A * B) * (C * D)$
□

END OF PAPER