1.Solution:

Greedy Algorithm Solution: The optimal solution is to sort the two sets A and B(both in the same order, increasing or decreasing order): Let the set $\{a_i\}$ be sorted so that $a_1 \geq a_2 \geq \ldots \geq a_n$ and set $\{b_i\}$ be sorted so that $b_1 \geq b_2 \geq \ldots \geq b_n$. And then pair a_i with b_i.

Proof:

We can suppose that the optimal payoff isn't produced from the Greedy Algorithm Solution above. Set S to be the optimal solution, in which a_1 is paired with b_i and a_j is paired with b_1. Notice that $a_1>a_j$ and $b_1>b_i$.At the same time we can consider another solution S′ in which a_1 is paired with b_1, a_j is paired with b_i, while all other pairs are the same as S. Then we have:

Payoff(S)/Payoff(S′) $=(a_1)^{b_i} (a_j)^{b_1}/(a_1)^{b_1}(a_j)^{b_i} =(a_1 / a_j)^{b_i - b_1}$

Since $a_1>a_j$ and $b_1>b_i$, then Payoff(S)/Payoff(S′)<1. This is a contradiction to the assumption that S is the optimal solution. So a_1 has to be paired with b_1. Then we can repeat the argument for the remaining elements and get the result.

Time Complexity:

If A and B are already sorted, the time complexity is O(n).

If A and B are not sorted, then sort them first and the time complexity is O(n logn).

2. Solution:

Pseudocode:

MatrixChainOrder(int A[], int n):

#Array A is the array defined above, and n is the array size, so we have a total n-1 matrix.

#We create a 2d matrix with size n*n, and we have taken one extra index so that we can omit index 0 to maintain simplicity.

```
1:int m[n][n];let i, j, k, L, q be integers.
#m[i][j] = Minimum number of scalar multiplications needed to compute the matrix from i to j where dimension of ith matrix is p[i-1]*p[i].
2:for i = 1 to n-1:
3:    m[i][i] = 0 #cost is zero when multiplying one matrix.
#now we start filling matrix m in diagonal manner
#like cost of all 2 matrxi multiplication,then cost for 3 and so on
4:for L = 2 to n-1:
5:    for i = 1 to n – L do
6:        j = i + L - 1
7:        m[i][j] = INT_MAX
#considering every intermediate matrix:
8:        for k = I to = j – 1 do
9:            q = m[i][k] + m[k + 1][j]+ p[i - 1] * p[k] * p[j] #q =scalar multiplications
10:           if q < m[i][j]:
11:               m[i][j] = q
12:return m[1][n - 1] #returing the final solution
```

3. Solution:

Pseudocode:

Pseudopolynomial:

F(0, 0) = 0 **# if b = 0, F(0, b) = 0**

for b = 1 to B do

    F(0, b) = −1

for i = 1 to n do

    for b = 0 to B do

        F(i, b) = F(i − 1, b)

 # Define F(i, b) to be the maximum profit of a subset Q ⊆ {1, 2, ..., i}such that

Σ $P_i \in Q$ a_i = b, and −∞ if no such set exists.

        if b − s_i ≥ 0 and F(i − 1, b − s_i) ≥ 0

        if F(i − 1, b − s_i) + p_i > F(i, b)

            then F(i, b) = F(i − 1, b − s_i) + p_i

Time Complexity:

The time Complexity is O(nB) since we have two nested for loops: the outer loop is exectued n times and the inner loop B times.

4. Solution:

Pseudocode:

Prrint-Neatly(n)

let P[1..n] and C[1..n] be new tables

#C[k] contains the cost of printing neatly words l_k through l_n

for k = n downto 1 do

    if $\sum_{i=k}^{n}$ l_i+n-k < M then

#If $\sum_{i=k}^{n}$ l_i+n-k < M then then put all words on a single line for an optimal solution.

        C[k] = 0

    end if

    q = ∞

    for j = 1 downto n – k do

        if $\sum_{m=1}^{j}$ l_{k+j} + j - 1 < M and (M - $\sum_{m=1}^{j}$ l_{k+j} + j - 1) + C[k + j + 1] < q then

            q = (M - $\sum_{m=1}^{j}$ l_{k+j} + j - 1) + C[k + j + 1]

            P[k] = k + j

        end if

    end for

    C[k] = q

end for

Time Complexity:

The time complexity is O(n).