Main Idea and Proof

The main idea to solve this problem is divided into two parts. The first part deals with computing the cost i.e sum of squared errors(SSE) while the second part deals with solving the problem by using this calculated error.

Part A (Calculating Cost):

$$a = \frac{n \sum_{i=1}^{n} x_i y_i - (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2} \quad \text{and} \quad b = \frac{\sum_{i=1}^{n} y_i - a \sum_{i=1}^{n} x_i}{n}.$$

$$\text{Error}(L, P) = \sum_{i=1}^{n} (y_i - a x_i - b)^2.$$

The above formula gives the error from point i =1, …. n.
Let the point be denoted by $p_i = (x_i , y_i)$.
Now, to calculate the error from point j to k, the value of i = j, …. k
To calculate the error, we need 4 summations. $\Sigma x_i y_i$, Σx_i , Σy_i , Σx_i^2 where i = 0, …. n
We can represent the summations as prefix sum arrays.

Let
X denote array of x coordinates
Y denote array of y coordinates
XY denote array such that XY = [$x_1 * y_1$, ……. $x_n * y_n$]
X^2 denote [x_1^2 , …. x_n^2]

We can calculate the prefix sum of arrays such that for any array A = [a_1 , a_2, ... a_n]

Prefix_sum_of_A = [a_1, $a_1 + a_2$, $a_1 + a_2 + \dots a_n$].

Now, to calculate Σx_i we have

Σx_i = prefix _sum_of_X[k] - prefix _sum_of_X[j-1] such that i = 0 <= j ,….. k <= n

Similarly we can calculate the other summations using the prefix_sum of respective arrays.

Now, using the above formula, we can calculate the sum of squared error for the particular section.

To calculate the total cost, add the penalty C to return the total cost.

Part B

Suppose that there is a set P of n points in the two-dimensional plane, denoted by $(x1, y1), (x2, y2), \cdot\cdot\cdot, (xn, yn)$, and suppose $x1 < x2 < \cdot\cdot\cdot < xn$.

The cost for a single line that passes through these points is given by $\varepsilon P + C$. where C is the penalty per line.

Suppose we make the n points into k segments such that the total cost will be given by

$$\text{cost}(P,C) = \varepsilon(P1) + \varepsilon(P2) + \cdot\cdot\cdot + \varepsilon(Pk) + Ck.$$

Now, Consider set of points from index i to j. Let the cost for points from i^{th} index to j^{th} index be denoted by COST(i,j).

Let the optimum cost until index j from the first index be denoted by OPT_COST[j]

Consider a trivial condition, where j = 0.
As there is only a single point, the optimum cost is OPT_COST[0] = 0

For j =1, Single line passes through Point 0 and Point 1.

Therefore, the cost will be given by COST(0,1)

For j = 2 We have 2 cases,

Case 1: We can have a partition at j =1

Therefore cost can be given as sum of cost till j = 1 i.e COST(0,1)and COST(1,2)
Therefore,

cost = COST(0,1) + COST(1,2)

But from the above relation we already know that,

COST(0,1) = OPT_COST[1]

Therefore, we have the following result,

cost = OPT_COST[0] + COST(1,2)

Case 2: We can have a single line from point 0 to point 2

In this case,

cost = COST(0,2)

Now, the minimum among these costs will be the value of OPT_COST[2]. Therefore,

OPT_COST[2] = min(COST(0,2) , OPT_COST[1] + COST(1,2))

Therefore, for any given value of j, we can generalize that the starting point for a line can range from 1 to j.

Therefore, by **Induction,**

We can write the equation as

$$\mathrm{OPT\_COST}[j] = \begin{cases} 0 & \text{for } j=0 \\ \mathrm{MIN}\left(\mathrm{COST}(i,j) + \mathrm{OPT\_COST}(i-1) \right) & \text{for } j>0 \text{ and } 1 \le i \le j \end{cases}$$

Therefore, from the above property its clear that we have a **Optimal Substructure Property.**
Hence, we can use **Dynamic Programming** to store the results in a table and resuse it.

Pseudo Code

The pseudo code provided is applicable to ONLY ONE instance from the group of instances.

prefix_sum_X -> prefix sum array of array X
prefix_sum_Y -> prefix sum array of array X
prefix_sum_XY -> prefix sum array of array X * Y
prefix_sum_X^2 -> prefix sum array of array X^2
N = length of X
C = penalty per line
cost_matrix[N][N] = cost of line from point i to point j eg cost_matrix[i][j]. Initialize all indices to -1.

CALCULATE_COST (i,j)   // i starting index, j ending index

1. IF cost_matrix[i][i] != -1 THEN RETURN cost_matrix[i][j]
2. n = j - i + 1
3. a = ((n * (prefix_sum_XY[j]-prefix_sum_XY[i-1])) - ((prefix_sum_X[j]-prefix_sum_X[i-1]) * (prefix_sum_Y[j]-prefix_sum_Y[i-1]))) / ((n * (prefix_sum_X^2[j] - prefix_sum_X^2[i-1])) - SQUARE(prefix_sum_X[j] - prefix_sum_X[i-1]))
4. b = ((prefix_sum_Y[j]-prefix_sum_Y[i-1]) - (a * (prefix_sum_X[j]-prefix_sum_X[i-1]))) / n
5. cost = SUM (SQUARE(Y[i-1 : j] - (a * X [i-1 : j]) - b)) + C
6. cost_matrix[i][j] = cost
7. return cost

MULTI_LINE_FIT()

```
1.  opt_cuts = 2D array which contains partitions until point j
2.  opt_costs = 2D array which contains optimum costs until point j
3.  FOR j = 0 TO N
4.      opt_cost = CALCULATE_ERROR(0,j)
5.      opt_cut = -1
6.      FOR i = 1 to j
7.          current_cost = opt_costs [ i -1] + CALCULATE_COST (i,j)
8.          IF current_cost < opt_cost
9.              opt_cost = current_cost
10.             opt_cut = i -1
11.     opt_costs [j] = opt_cost
12.     IF opt_cut != -1
13.         opt_cuts [j] = opt_cuts [opt_cut] + [opt_cut]
14.  K = length of opt_cuts[N-1] + 1
15.  LAST_POINTS_LIST = opt_cuts[N-1] + [N-1]
16.  OPT_COST = opt_cost[N-1]
17.  RETURN
```

Time Complexity

Part A

To find Prefix Sum, we need to iterate the whole array once. The time complexity to iterate through 4 arrays is given by

prefix_sum_X -> O(N)
prefix_sum_Y -> O(N)
prefix_sum_XY -> O(N)
prefix_sum_X^2 -> O(N)

The time complexity of precomputing the prefix sums is O(N)

To calculate the error, we are accessing the prefix sum arrays. The time complexity for accessing the array elements is O(1)

Therefore, time complexity to to calculate error is O(1).

Part B

In the dynamic programming approach, we are iterating through the array of points in a nested fashion.

opt_costs [i] + CALCULATE_ERROR(i,j) for each i …… O(N)

As proved earlier, the time complexity of calculating the error is O(1). Therefore the time complexity of this step is O(N)

This step is performed for N times i.e for values of j from 1 to N.

Therefore, the **Total Time Complexity is O(N^2).**