Author: Ahmed Nouralla - Group: BS19-02 - a.shaaban@innopolis.university

Task:

- We have n points in 2D Cartesian coordinate system. $(n > 0)$
- We want to draw k connected lines $(0 \leq k < n)$ from point #1 to point #n.
- Total cost should be minimized. Cost = sum of SSE for all line segments + Number of lines.

Notation:

MIN(j) = minimum cost for points$(x_1, y_1), (x_2, y_2), \dots, (x_j, y_j)$.

e(i, j) = minimum sum of squared errors for points$(x_i, y_i), (x_{i+1}, y_{i+1}), \dots, (x_j, y_j)$.

Solution:

1) Sub-problems: The big problem is calculating Min(n), The minimum cost for points from 1 to n.

The solution will be to create one line from some point ‘i’ to the last point n, that’s 1 line, so we add 1 + e(i, n) to the result, then we try to solve the smaller **sub-problem** MIN(i-1), **which will be easier to solve.**

So the problem MIN(n) is solved using the results obtained by solving MIN(i-1), which will require another smaller sub-problem, and so on until we reach the base case, which will be trivial to solve. Then we unfold this recursion and combine the solutions we got so far until we reach the answer.

Important moments:

- Which point i do we have to choose when we are solving MIN(k)? Between all the points from 1 to k, we will choose the one that minimizes the final answer.
- As we saw: problem MIN(n) will call several problems MIN(k), MIN(k) will also call other problem MIN(h). These **sub-problems** can **overlap,** that is, we may need to solve the same problem twice. To avoid doing that and make the program faster we maintain a data structure to hold the solutions of previously solved problems, and whenever we need them again, we get them in O(1) instead of calculating them again. This method is called memoization and we will use it in the pseudocode below.

2) Base cases:

- MIN(0) = 0, because there is no points at all.
- **(Optional)** MIN(2) = 1, because the best scenario is when we connect points #1 and #2 with a straight line that makes the error = 0. cost = SSE + #lines = 0 + 1 = 1.

3) The recurrence:

$$MIN(j) = \begin{cases} \min_{1 \leq i \leq j}\{1 + e(i,j) + MIN(i-1)\} & j > 1 \\ 0 & j = 0 \end{cases}$$

4) Pseudo-code:

```
// memo is an array of type Double used for memoization.
Double MIN(int j){
    If j = 0 then return 0
    If memo[j] is not null then return memo[j]
    memo[j] = Minimum( e(i, j) + 1 + MIN(i-1) ) forall i from 1 to j
    return memo[j]
}
```

5) Time complexity:

- In the worst case: MIN(n) will call MIN(n-1) which will call MIN(n-2) and so on, until we reach the base case, each one will do a linear amount of work in worst case to calculate minimum of all results, so that is a quadratic amount of work.
- Each call for MIN(j) will need additional linear amount of work in the worst case for calculating e(i, j).

Overall we will have a worst case time complexity as **$O(n^3)$.**

Optimization: We can pre-calculate values of e(i, j) and get them in O(1) while doing recursion, then the time complexity will be **$O(n^2)$.**

In practice: it will be faster because we are doing memoization so the recursion tree will terminate sometimes**.**