In this problem, you basically have a set of \( n \) points (the account events) and a set of intervals (the “error bars” around the suspicious transactions, i.e., \([t_i - e_i, t_i + e_i]\)), and you want to know if there is a perfect matching between points and intervals so that each point lies in its corresponding interval). Without loss of generality, let us assume \( x_1 \leq x_2 \leq \ldots \leq x_n \).

A greedy style algorithm goes like this:

```plaintext
for \( i = 1, 2, \ldots, n \)
if there are unmatched intervals containing \( x_i \)
    Match \( x_i \) with the one that ends earliest
else
    Declare that there is no perfect matching
```

It is obvious that if the algorithm succeeds, it really finds a perfect matching. We want to prove that if there is a perfect matching, the algorithm will find it. We prove this by an exchange argument, which we will express in the form of a proof by contradiction.

Suppose by way of contradiction that there is a perfect matching, but that the above greedy algorithm does not construct one. Choose a perfect matching \( M \), in which the first \( i \) points \( x_1, x_2, \ldots, x_i \) match to intervals in the same way described in the algorithm, and \( i \) is the largest number with this property. Now suppose \( x_{i+1} \) matches to an interval centered at \( t_l \) in \( M \), but the algorithm matches \( x_{i+1} \) to another interval centered at \( t_j \). According to the algorithm, we know that \( t_j + e_j \leq t_l + e_l \). Suppose \( t_j \) is matched to \( x_k \ (x_k \geq x_{i+1}) \) in \( M \). Then we have

\[
    t_l - e_l \leq x_{i+1} \leq x_k \leq t_j + e_j \leq t_l + e_l,
\]

so in \( M \) we can instead match \( x_k \) to \( t_l \) and match \( x_{i+1} \) to \( t_j \) to have a new perfect matching \( M' \), which agrees with the algorithm. \( M' \) agrees with the output of the greedy algorithm on the first \( i + 1 \) points, contradicting our choice of \( i \).

To bound the running time, note that if we simply enumerate all unmatched intervals in each iteration of the `for` loop, it will take \( O(n) \) time to find the unmatched one that ends earliest. There are \( n \) iterations, so the algorithm takes \( O(n^2) \) time.