**Edit distance:**

The edit distance between two strings is the minimum number of insertions, deletions, and replacements required to transform one string into the other. We define the edit distance as:

$$\text{Edit}(i, j) = \text{edit distance from } A[1..i] \text{ to } B[1..j]$$

Algorithm:

```pseudo
EDITDISTANCE(A[1..m], B[1..n]):
    for j ← 0 to n
        Edit[0, j] ← j
    for i ← 1 to m
        Edit[i, 0] ← i
        for j ← 1 to n
            ins ← Edit[i, j - 1] + 1
            del ← Edit[i - 1, j] + 1
            if A[i] = B[j]
                rep ← Edit[i - 1, j - 1]
            else
                rep ← Edit[i - 1, j - 1] + 1
            Edit[i, j] ← min{ins, del, rep}
    return Edit[m, n]
```

The time complexity is $O(mn)$. The basic idea is to compute the Edit distance for all substrings and choose the minimum. This can be implemented using a matrix where each cell stores the minimum cost to transform the substrings ending at those indices.
Li Russians $\rightarrow O(n\log n)$ time
Faster?? IDK, but prob not.
Given array L[1..n] where

\[ L(i) \] is length of \( i \)th board from left

\[ \text{OptCost}(i,k) = \min \text{ possible cost of cutting} \]

\[ \text{long plank consisting of short boards } i+1 \ldots k \]

We want \( \text{OptCost}(0,n) \)

\[ \text{OptCost}(i,k) = \begin{cases} 
0 & \text{ if } k-i = 1 \\
\sum_{j=i+1}^{k} L[j] + \min_{1 \leq j < k} \{ \text{OptCost}(i,j) + \text{OptCost}(j,k) \} & \text{ otherwise}
\end{cases} \]
\[
\begin{array}{cccc}
  i+1 & j & \cdots & i+1 \\
  \vdots & \vdots & \ddots & \vdots \\
  k & \cdots & \cdots & k
\end{array}
\]

\text{OptCost}(i, n)

\text{For } k = n \text{ to } 0
\text{For } i = 0 \text{ to } n
\text{Compute } \text{OptCost}(i, k) \leftarrow O(n) \text{ time}
\text{For } j = i+1 \text{ to } k-1
\text{case}
\text{O}(n^3) \text{ time} \quad \rightarrow \quad O(n^2) \text{ time}
\text{\rightarrow } O(\log n) \text{ time}