16. (100 pts.) OLD Homework problem (not for submission):

Simplifying data.

A k-step function is a function of the form

\[ f(x) = b_i \quad \text{if} \quad a_i \leq x < a_{i+1} \quad (i = 0, \ldots, k-1) \]

for some \(-\infty = a_0 < a_1 < \cdots < a_{k-1} < a_k = \infty\) and some \(b_0, b_1, \ldots, b_{k-1}\).

We are given \(n\) data points \(p_1 = (x_1, y_1), \ldots, p_n = (x_n, y_n)\) and a number \(k\) between 1 and \(n\). Our objective is to find a \(k\)-step function \(f\) such that \(f(x_i) \geq y_i\) for all \(i \in \{1, \ldots, n\}\), while minimizing the total “error” \(\sum_{i=1}^{n} (f(x_i) - y_i)\) (this is the total length of the red vertical segments in the figure below).

![Graph showing a k-step function with vertical segments indicating error]

16.A. (70 pts.) Describe an algorithm, as fast as possible, that computes the minimum total error of the optimal \(k\)-step function. Bound the running time of your algorithm as a function of \(n\) and \(k\).

[Note: in dynamic programming questions such as this, first give a clear English description of the function you are trying to evaluate, and how to call your function to get the final answer, then provide a recursive formula for evaluating the function (including base cases). If a correct evaluation order is specified clearly, iterative pseudocode is not required.]

16.B. (30 pts.) Describe how to modify your algorithm in (A) so that it computes the optimal \(k\)-step function itself.

17. (100 pts.) OLD Homework problem (not for submission):

Closest subsequence

Define the \(L_1\)-distance between two sequences of real numbers \(\langle a_1, \ldots, a_m \rangle\) and \(\langle b_1, \ldots, b_m \rangle\) to be \(|a_1 - b_1| + \cdots + |a_m - b_m|\).

Consider the following problem: given two sequences of real numbers \(A = \langle a_1, \ldots, a_m \rangle\) and \(B = \langle b_1, \ldots, b_n \rangle\) with \(m \leq n\), find a subsequence of \(B\) of length \(m\) that minimizes its \(L_1\)-distance to \(A\).

17.A. (70 pts.) Describe an algorithm, as fast as possible, that computes the \(L_1\)-distance of the optimal subsequence of \(B\) to \(A\). Bound the running time of your algorithm as a function of \(m\) and \(n\).
17.B. (30 pts.) Describe how to modify your algorithm in (A) so that it computes the optimal subsequence itself.
OLD Homework problem (not for submission):

Fold it!

We are given a “chain” with \( n \) links of lengths \( a_1, \ldots, a_n \), where each \( a_i \) is a positive integer. We are also given a positive integer \( L \). We want to determine if it is possible to “fold” the chain (in one dimension) so that the length of the folded chain is at most \( L \). More formally, we want to decide whether there exists \( t \in [0, L] \) and \( s_1, \ldots, s_n \in \{-1, +1\} \) such that \( t + \sum_{i=1}^{j} s_ia_i \in [0, L] \) for all \( j \in \{0, \ldots, n\} \). (Here, \( t \) denotes the starting position, and \( s_i = \pm 1 \) depending on whether we turn rightward or leftward for the \( i \)th link.)

Example: for \( a_1 = 5 \), \( a_2 = 1 \), \( a_3 = 7 \), \( a_4 = 2 \), \( a_5 = 8 \), and \( L = 9 \), a solution is shown below.

\[ \begin{array}{c}
\text{18.A. (70 pts.) Provide an } O(nL) \text{-time algorithm to decide whether a solution exists. (Argue why the stated running time is correct.)}
\end{array} \]

Partial credit would be given to slower solutions with running time \( O(nL^2) \) or \( O(nL^3) \).

\[ \begin{array}{c}
\text{18.B. (30 pts.) Using (A) as a subroutine, describe an algorithm (as fast as possible) to find the minimum length } L \text{ such that a valid folding exists. } L^* \text{ of the best folding. What is the running time of your algorithm as a function of } n \text{ and } L^*? 
\end{array} \]