Given $\{Y_1, \ldots, Y_n\}$ and M, design an algorithm to construct a non-decreasing set $\{X_1, \ldots, X_n\}$ to minimize

$$\text{cost}(X) = \sum_{i=1}^{n} |X_i - Y_i|^2.$$

1. A subproblem is finding a non-decreasing sequence $\{X_1, \ldots, X_n\}$ such that

$$0 \leq X_1 \leq \cdots \leq X_n \leq m,$$

where $0 \leq m \leq M$, and M is the original problem parameter. Problems are ordered by increasing m. Denote the solution sequence corresponding to m by $X^{(m)}$.

2. The base case is where $m = 0$. Then the sequence $X^{(0)}$ is $\{0, \ldots, 0\}$ (i.e. all 0's), and the cost of the base case is $\text{cost}(X^{(0)}) = \sum_{i=1}^{n} Y_i^2$.

3. First we'll consider what the sequence *is*, before considering the cost of the squence. Symbolically,

$$X^{(j)} = \{X_1^{(j-1)}, \ldots, X_k^{(j-1)}, \underbrace{j, \ldots, j}_{n-k}\}.$$

So the non-decreasing sequence minimizing cost with upper limit j is equal to the first k terms of the sequence with upper limit $j-1$, followed by $n-k$ elements of value j, where k is chosen by inspection to minimize the cost of $X^{(j)}$. Note that k may be n, so it is possible that $X^{(j)}$ does not contain a term with value j, and k may be 0, so that $X^{(j)}$ does not contain any of the elements from $X^{(j-1)}$.

Thus the recurrence relation for the cost of the sequence is:

$$\text{cost}(X^{(j)}) = \min(\text{cost}(\{X_1^{(j-1)}, \ldots, X_k^{(j-1)}, \underbrace{j, \ldots, j}_{n-k}\})|k \in \{0, \ldots, n\})).$$

4. Now we prove that the recurrence relation is correct.

Proof. Note that when $m = 0$, there is only one feasible sequence of elements, that with all 0's. The cost of this sequence is therefore the solution to the subproblem for $m = 0$.

So we know that for some i, we can correctly solve the subproblem for $m = i$, so we know the sequence $X^{(i)}$. We claim that we can therefore solve the subproblem for $m = i+1$. Since we know the sequence $X^{(i)}$, we can construct feasible solutions to the subproblem for $m = i+1$ by constructing each of the sequences

$$\{X_1^{(i)}, \ldots, X_k^{(i)}, \underbrace{i+1, \ldots, i+1}_{n-k}\}) \text{ for } k \in \{0, \ldots, n\}.$$

We know that the optimal sequence for $m = i+1$ must be one of these, since the only way we could possibly improve the sequence from the solution for $m = i$ is to increase the value of some of the elements in the sequence. Then we simply set $X^{(i+1)}$ to the sequence with minimum cost. It follows that

$$\text{cost}(X^{(i+1)}) = \min(\text{cost}(\{X_1^{(i)}, \ldots, X_k^{(i)}, \underbrace{i+1, \ldots, i+1}_{n-k}\})|k \in \{0, \ldots, n\})).$$

Noting that this is the same relationship as our recurrence relation, we conclude that our recurrence relation is correct. □

5. Now we give an algorithm for finding a solution to the subproblem.

   Suppose we have correctly constructed $X^{(i-1)}$, let this be denoted by an array `x` of length n, and let `best` be positive infinity. We're trying to construct $X^{(i)}$. Let `k` be n, let `min_index` be n. Save `x` in an array `keep`, so we remember the correct solution for $m = i - 1$.

   Change the `k`th element of `x` to i, and compute the cost of the new `x`. If this cost is less than `best`, then let `best` be this cost, and let `min_index` be `k`. Decrement `k`, and repeat until `k` is 0.

   Finally reset `x` so that it is the first `min_index` elements of `keep`, followed by $n -$ `min_index` elements of value i.

6. Finally we give an algorithm for the original problem.

   Given sequence `y` of length `n` and `M`, first let `x` be an array of zeroes of length `n`. Let `best` be positive infinity.

   `for (m=0; m <= M; m++) {`

   Copy `x` into `keep`. Let `i` be `n`, let `m_best` be positive infinity, let `min_index` be `n`.

   Change the `i`th element of `x` to `m`. Compute the cost of the new `x`. If this cost is less that `m_best`, then let `m_best` be the cost and let `min_index` be `i`. Decrement `i` and repeat until `i` is 0.

   Reset `x` so that it is the first `min_index` elements of `keep`, followed by $n -$ `min_index` elements of value `m`.

   Compute the cost of `x` again, and if it's less than `best`, then set `best` equal to the cost.

   `}`

   Return `best`.

   Thus the solution to the original problem is the cost of $X^{(M)}$, i.e. the cost of the solution sequence to the subproblem corresponding to M.

7. The running time of the algorithm is $O(nM)$. Essentially, we're solving M subproblems that correspond to constructing sequences with upper limit equal to each non-negative integer less than or equal to M; thus the outer loop in our algorithm runs M times. At each step in the loop, we search for the `min_index`, which takes $O(n)$ operations, since we have to go through the entire `x` array of length n.