(3) **Finding elements near the median** (35 points) Given an unsorted array $A$ of $n$ distinct numbers, and an integer $k$ where $1 \leq k \leq n$, design an algorithm that finds the $k$ numbers in $A$ that are closest in value to the median of $A$ in $\Theta(n)$ time.

(Note: the position of the elements in the array with respect to the median is irrelevant; only their is important. The numbers that are closest in value to the median may be larger or smaller than the median.)

Consider the following algorithm:

1. Find the median of $A$, call it $x$.
2. Form another array $B[1:n]$ where
3. Find the $k$th smallest element in $B$, call it $y$.

Using the linear-time $k$th smallest algorithm for Steps (1) and (3), the entire algorithm runs in $\Theta(n)$ time.
Problem (Generalized k'th smallest)

Given n distinct unsorted numbers $x_1, \ldots, x_n$ with associated positive weights $w_1, \ldots, w_n$ where $W := \sum_{i=1}^{n} w_i$, and a number $k$ with $0 \leq k \leq W$, find element $x_j$ such that

$$\sum_{x_i < x_j} w_i \leq k \quad \text{and} \quad \sum_{x_i > x_j} w_i \leq W - k,$$

in $\Theta(n)$ time.

Algorithm

Our algorithm proceeds as follows:

1. Find the median $X$ of the $x_i$ using the worst-case linear-time algorithm for finding the (ordinary) $k$'th smallest.

$\Theta(n)$

2. Partition the $x_i$ around the median $X$ into a left portion of $x_i < X$ and a right portion of $x_i > X$.

3. Evaluate $L := \sum_{x_i \leq X} w_i$ and $R := \sum_{x_i > X} w_i$.

$T(\frac{n}{2})$

4. If $L \leq k$ and $R \geq W-k$ then return $X$.

Else if $L > k$ then recursively find the generalized $k$'th smallest over the left portion with $W' := L$.

Otherwise recursively find the generalized $(k-L)$'th smallest over the right portion with $W' := R$.

Analysis

This takes time

$$T(n) = T(\frac{n}{2}) + \Theta(n) = \Theta(n).$$
(5) **(Longest balanced subsequence)** (50 points) A string of parentheses is said to be balanced if the left- and right-parentheses in the string can be paired off properly. For example, the strings "(())" and "(()" are both balanced, while the string "(()(" is not balanced.

Given a string $S$ of length $n$ consisting of parentheses, suppose you want to find the longest subsequence of $S$ that is balanced. Using dynamic programming, design an algorithm that finds the longest balanced subsequence of $S$ in $O(n^3)$ time.

(1) **(Structure of optimal solution)**

The longest balanced subsequence (LBS) of $S[1:n]$, call it $W$, must end by choosing both $S[1]$ and $S[n]$ or not, which leads to the following four cases:

**Case (i)** $W$ uses both $S[1]$ and $S[n]$, and these parentheses are paired with each other:

```
  1  2  ...  n-1  n
 S  └──┐
     │
     │ Must be an LBS of S[2:n-1].
     │
     │ Parentheses are paired off.
     │
     │ Must have $S[1] = 'c'$ and $S[n] = 'y'$.
```

**Case (ii)** We use both $S[1]$ and $S[n]$, but these parentheses are not paired with each other:

```
  1  ...  k  k+1  ...  n
 S  └──┐
     │
     │ Must be an LBS of S[1:k].
     │
     │ Must be an LBS of S[k+1:n].
```
Case (iii) \( W \) does not use \( S[i] \).

\[
S \begin{array}{c}
1 \\
2 \\
h \end{array}
\]

Must be an LBS of \( S[2:h] \).

Case (iv) \( W \) does not use \( S[h] \).

\[
S \begin{array}{c}
1 \\
2 \\
3 \\
\vdots \\
h \end{array}
\]

Must be an LBS of \( S[2:h-1] \).

(2) (Recurrence for value of an optimal solution)

In each case the subproblem that arises is to find an LBS over a substring of \( S \), call it \( S[i:j] \), which can be described by the pair \( (i,j) \).

Let

\[ L(i,j) := \text{length of LBS of } S[i:j]. \]

Then by Part (1),

\[
L(i,j) = \begin{cases} 
L(i+1,j-1) + 2 & \text{if } S[i] = 'L' \text{ and } S[j] = 'L', \\
\max \left\{ \max_{i<k<j} \left( L(i,k) + L(k+1,j) \right) \right\} & \\
L(i+1,j) & \text{if } 1 \leq i < j, \\
L(i,j-1) & \text{if } i \geq j.
\end{cases}
\]
The value of the solution to the input problem is:
\[ L(1,n). \]

(3) (Evaluation phase)
We evaluate the recurrence in a table \( L[i:n,0:n] \).
The dependencies among entries are:

These dependencies can be satisfied by filling in the upper triangle of table \( L \) in diagonal-major order:

There are \( O(n^2) \) entries in the table, and each takes \( O(n) \) time to evaluate by the recurrence, so the evaluation phase runs in \( O(n^3) \) time.
(4) (Recovery phase)

To recover the LBS, we start at the goal entry \( L[1,n] \), determine which of cases (i) - (iv) gave its value, and recurse (possibly on two subproblems in case (iii)).

The recursive calls in this process can be mapped to the nodes of a binary tree (in which nodes have 1 child in cases (i), (iii), (iv), and 2 children in case (ii)) that has \( \mathcal{O}(n) \) total nodes. Each node takes \( \mathcal{O}(n) \) time to determine which case gave its value, so the recovery phase takes \( \mathcal{O}(n^2) \) total time.

The entire algorithm using both phases takes total time \( \mathcal{O}(n^3) + \mathcal{O}(n^2) = \mathcal{O}(n^3) \).
Problem (Best approximation by lines)

Given \( n \) points \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\) where \( x_1 \leq x_2 \leq \ldots \leq x_n \),

and an integer \( k \geq 1 \), find a partition of the points into \( k \) consecutive runs along the x-axis, so that the sum of the

errors of the least-squares lines through the runs is minimum.

Algorithm

(1) (Structure of optimal solution)

An optimal solution ends with some final run of points \((x_i, y_i), (x_{i+1}, y_{i+1}), \ldots, (x_n, y_n)\) being in

the \( k \)th line. The preceding points \( 1, \ldots, i-1 \)

must be in an optimal solution with \( k-1 \) lines

(which can be shown using proof by contradiction).

(2) (Recurrence for the value of an optimal solution)

The subproblem that arises is to find an optimal solution over a prefix of the points \( 1, 2, \ldots, i \) using

some number \( l \) of lines, which can be specified by the pair \((i, l)\). Let

\[
E(i, l) := \text{total error in a best approximation of points } 1, 2, \ldots, i \text{ by } l \text{ least-squares lines.}
\]

Then by Part (1),

\[
E(i, l) = \begin{cases} 
\min_{1 \leq j < i} \frac{3}{2} E(j, l-1) + e(j+1, i) \frac{7}{8} & \text{if } l \in \mathbb{N} \text{ and } l \geq 2; \\
\infty & \text{otherwise.}
\end{cases}
\]

The solution value for the input problem is \( E(n, k) \).
Problem, continued

(3) (Evaluation phase)

We evaluate the recurrence in a table $E[1:n, 1:k]$. The dependencies among entries are:

$$
\begin{array}{c}
  E \\
  \downarrow \\
  i \\
  \downarrow \\
  n \\
\end{array}
\begin{array}{c}
  j \\
  \leftarrow \quad \leftarrow \\
  a \quad a \\
  \leftarrow \quad \leftarrow \\
  g \quad g \\
  \downarrow \\
  (i,l) \\
\end{array}
\begin{array}{c}
  \text{Goal value } E[n,k].
\end{array}
$$

Filling in the table in row-major or column-major order satisfies these dependencies. This takes time:

\[
O\left(\sum_{1 \leq l \leq k} \sum_{1 \leq i \leq n} \sum_{1 \leq j < i} \left( \Theta(i) + \Theta(i - (j+1) - 1) \right) \right)
\]

\[
= O\left(k \sum_{1 \leq i < n} \Theta(i) \right)
\]

\[
= O\left(k \sum_{1 \leq i \leq n} \Theta(i^2) \right)
\]

\[
= O(\kappa n^3).
\]

(4) (Recovery phase)

Starting from the goal entry $E[n,k]$, finding the extent of the last run of points in an optimal solution by re-evaluating the min in the recurrence, and recursively recovering the preceding portion of an optimal solution, takes time $O(\kappa n^2)$ which is dominated by the evaluation phase.
Problem, continued

Note. By precomputing $c(i,j)$ in another table, to speed up evaluation of the min in the recurrence equation, we can reduce the time for the evaluation phase to $O(n^3) + O(kn^2) = O(n^3)$. \[\square\]
Problem (Two-finger dialing)

Given an $n$-digit telephone number $T[1:n] = t_1 t_2 \ldots t_n$, dial it with two fingers starting on the `*` and `#` keys so as to minimize the total distance the fingers travel in $O(n)$ time.

Solution

(i) Consider how an optimal dialing of $T$ ends. Prior to pressing the key $T[n]$, the two fingers must have been in some final state

$$(f_1, f_2) \in \bigcup \{ (a, b) : a, b \in K \},$$

where $K = \{ 0, 1, \ldots, 9, *, # \}$.

Furthermore, the sequence that dialed the prefix $T[1:n-1]$ must be an optimal dialing that ends in state $(f_1, f_2)$ (as can be shown using proof by contradiction).

This leads to the subproblem

$$(i, a, b) \; \overset{\text{find an optimal dialing of}}{\Rightarrow} \; \text{the prefix } T[1:i] \text{ that ends in state } (a, b).$$

Let

$$D_{a,b}(i) \; \overset{\text{total distance travelled in an}}{\Rightarrow} \; \text{optimal dialing of } T[1:i] \text{ that ends in state } (a, b),$$

where $D_{a,b}(i) = \infty$ if it’s impossible to end at $(a, b)$ after dialing $T[1:i]$.

Then the solution value for the input problem is

$$\min_{a, b \in K} \left\{ D_{a,b}(n) \right\}.$$
Solution cont.

d(ab) = Euclidean distance between keys e.b.

A recurrence for D for i = 0 is

\[ D_{ab}(i) = \min_{c \in K} \{ d(ab) + D_c(i-1) \} \]

where for i = 0,

\[ D_{ab}(0) = \begin{cases} 0, & (a,b) = (\#,\#) \\ \infty, & \text{otherwise} \end{cases} \]

- (ii) For a \( \neq K \) let
- \( G\{a\} = \{ \text{for } G\{a\} \text{ if } a = T\{A\} \} \)

We can evaluate recurrence (*) in a
3-dimensional table \( D[ab,c] \) that has
\( \Theta(k^3) \) entries.

Filling in a given entry using (*) for increasing,
\( G\{k\} \) time using table look-up for c,
\( \Theta(k^2) \) total time of

Recovering the optimal dating from the D-table also takes \( \Theta(k^2) \) time.

since \( |k| = 12 = \Theta(k) \).

Therefore, \( G\{1\} = \Theta(k) \) time.

\[ G\{1\} = \{ \text{for } G\{1\} \text{ if } a = T\{A\} \} \]