Problem 1: Use the substitution method to show that the recurrence

\[
T(n) = \begin{cases} 
\sqrt{n} \cdot T(\sqrt{n}) + an & \text{if } n < 2 \\
1 & \text{otherwise}
\end{cases}
\]

where \(a > 0\) has the solution \(T(n) = O(n \lg \lg n)\).

Problem 2: Recall that we can quick-sort an array of \(n\) elements in \(O(n \lg n)\) time if we use an \(O(n)\) time partition algorithm to partition an array of \(n\) elements into two equally sized arrays such that all elements in one array are smaller than all elements in the other array. We use linear time selection to find the median in \(O(n)\) time and partition around this median.

Describe an \(O(n \log k)\) time \(k\)-partitioning algorithm that partitions an array \(A\) of \(n\) elements into \(k\) equally sized arrays \(A_1, A_2, \ldots, A_k\) such that elements in array \(A_i\) are smaller than elements in array \(A_{i+1}\) for all \(i, 1 \leq i < k\). You may assume that \(k\) is a power of two (meaning you can ignore floors and ceilings.)

Problem 3: The median of a sorted list \(L = x_1, x_2, \ldots, x_n\) of \(n\) integers is the element with index \(\left\lceil \frac{n}{2} \right\rceil\) and can be found in \(O(1)\) time.

Given two sorted lists \(L_1\) and \(L_2\) of \(n\) integers each, the median of all \(2n\) elements can be found in \(O(n)\) time by merging the two lists and returning the element with index \(n\). Design a faster algorithm and bound its asymptotic running time.
Problem 4: Consider two balanced search trees $T_1$ and $T_2$ representing two sets of integers $S_1$ and $S_2$ with $n_1$ and $n_2$ elements, respectively. Consider the problem of determining if $S_1 \subseteq S_2$ using $T_1$ and $T_2$.

- Describe an $O(n_1 \log n_2)$ time algorithm for determining if $S_1 \subseteq S_2$ using $O(1)$ extra space.

- Describe an $O(n_1 + n_2)$ time algorithm for determining if $S_1 \subseteq S_2$ using $O(n_1 + n_2)$ extra space.

- Describe an $O(n_1 + n_2)$ time algorithm for determining if $S_1 \subseteq S_2$ using $O(\log n_1 + \log n_2)$ extra space.
**Problem 5:** In this problem we design a divide-and-conquer algorithm for computing the skyline of a set of $n$ buildings.

A *building* $B_i$ is represented as a triplet $(L_i, H_i, R_i)$ where $L_i$ and $R_i$ denote the left and right $x$ coordinates of the building, and $H_i$ denotes the height of the building (note that the $x$ coordinates are drawn boldfaced.)

A *skyline* of a set of $n$ buildings is a list of $x$ coordinates and the heights connecting them arranged in order from left to right (note that the list is of length at most $4n$).

Example: The skyline of the buildings

$\{(3, 13, 9), (1, 11, 5), (12, 7, 16), (14, 3, 25), (19, 18, 22), (2, 6, 7), (23, 13, 29), (23, 4, 28)\}$

is

$\{1, 11, 3, 13, 9, 0, 12, 7, 16, 3, 19, 18, 22, 3, 23, 13, 29, 0\}$

(note that the $x$ coordinates in a skyline are sorted).

1. Let the size of a skyline be the total number of elements (coordinates and heights) in its list.

   Describe an algorithm for combining a skyline $A$ of size $n_1$ and a skyline $B$ of size $n_2$ into one skyline $S$ of size $O(n_1 + n_2)$. Your algorithm should run in time $O(n_1 + n_2)$.

2. Describe an $O(n \log n)$ algorithm for finding the skyline of $n$ buildings.