Problem 1: Convex layers  

Given a set $Q$ of points in the plane, we define the convex layers of $Q$ inductively. The first convex layer of $Q$ consists of those points in $Q$ that are vertices of $\text{Convex-Hull}(Q)$. For $i > 1$, define $Q_i$ to consist of the points of $Q$ with all points in convex layers $1, 2, \ldots, i-1$ removed. Then, the $i$th convex layer of $Q$ is $\text{Convex-Hull}(Q_i)$ if $Q_i \neq \emptyset$ and is undefined otherwise. In short, the convex layers of $Q$ are defined by repeatedly computing the convex hull of $Q$, and then removing its vertices from $Q$, until $Q$ is the empty set $\emptyset$.

(a) Give an $O(n^2)$-time algorithm to find the convex layers of the set of $n$ points.

(b) You are given with an unsorted array $A$ of $n$ real numbers. For each number $A[i]$, $1 \leq i \leq n$, you are asked to add 3 points to the set $Q$ (initialized to the empty set $\emptyset$), namely, $(0,0)$, $(0,i)$ and $(A[i],0)$. So $Q$ now has $3n$ points. How many convex layers does $Q$ have? How are the convex layers related to the sorted array obtained after sorting the array $A$?

(c) Prove that $\Omega(n \lg n)$ time is required to compute the convex layers of a set of $n$ points with any model of computation that requires $\Omega(n \lg n)$ time to sort $n$ real numbers.

Problem 2: Matching the sticks  

Suppose that you are given $n$ red and $n$ green sticks. All red sticks have different lengths, as do the green sticks. Moreover, for every red stick there is a green stick that is of the same length, and vice versa. You are asked to group the sticks into pairs of red and green sticks that are of the same lengths. To do so, you may perform the following operation: pick a pair of sticks one from the red bunch and one from the green bunch, and then using a Vernier caliper measure their lengths. This will tell you if both the sticks are of the same length or the red stick is longer or the green one is longer. Assume that such a comparison takes one time unit. Your goal is to find an algorithm that makes a minimum number of comparisons to determine the grouping. Remember that you may not directly compare two red sticks or two green sticks.

a. Describe a deterministic algorithm that uses $\Theta(n^2)$ comparisons to group the sticks into pairs.

b. Prove a lower bound of $\Omega(n \lg n)$ for the number of comparisons that must be made by an algorithm solving this problem. (Hint: How many leaves does the decision tree have?)

c. Give a randomized algorithm whose expected number of comparisons is $O(n \lg n)$, and prove that this bound is correct. What is the worst-case number of comparisons for your algorithm?
Problem 3: Stack depth for quicksort  \((6 + 6 + 8 = 20\) Points\)

The \textsc{Quicksort} algorithm discussed in the class contains two recursive calls to itself. After \textsc{Quicksort} calls \textsc{Partition}, it recursively sorts the left subarray and then it recursively sorts the right subarray. The second recursive call in \textsc{Quicksort} is not really necessary. We can avoid it by using an iterative control structure. This technique, called tail recursion, is provided automatically by good compilers. Consider the following version of quicksort, which simulates tail recursion:

\begin{verbatim}
Tail-Recursive-Quicksort(A, p, r)
1  while p < r
2    q ← Partition(A, p, r)
3    Tail-Recursive-Quicksort(A, p, q - 1)
4    p ← q + 1
\end{verbatim}

(a) Argue that \textsc{Tail-Recursive-Quicksort}(A, 1, A.length) correctly sorts the array \(A\).

Compilers usually execute recursive procedures by using a \textit{stack} that contains pertinent information, including the parameter values, for each recursive call. The information for the most recent call is at the top of the stack, and the information for the initial call is at the bottom. When a procedure is invoked, its information is \textit{pushed} onto the stack; when it terminates, its information is \textit{popped}. Since we assume that array parameters are actually represented by pointers, the information for each procedure call on the stack requires \(O(1)\) stack space. The \textit{stack depth} is the maximum amount of stack space used at any time during a computation.

(b) Describe a scenario in which \textsc{Tail-Recursive-Quicksort}’s stack depth is \(\Theta(n)\) on an \(n\)-element input array.

(c) Modify the code for \textsc{Tail-Recursive-Quicksort} so that the worst-case stack depth is \(\Theta(\lg n)\). Maintain the \(O(n \lg n)\) expected running time of the algorithm.

Problem 4: The \textit{i}th order statistic  \((15\) Points\)

Suppose that you have a “black-box” worst-case linear-time median finding algorithm. Give a simple, linear-time algorithm that finds the \textit{i}th smallest element (\textit{a.k.a.}, the \textit{i}th order statistic) in an unsorted array. (\textit{Hint:} You can use the median find algorithm and the linear-time algorithm to partition an array as subroutines).

Problem 5: Weighted median  \((6 + 7 + 7 = 20\) Points\)

In the class, we have discussed a linear-time deterministic algorithm to determine the \textit{i}th smallest element in an unsorted input array. This problem is about finding the \textit{weighted median}. For \(n\) distinct elements \(x_1, x_2, \ldots, x_n\) with positive weights \(w_1, w_2, \ldots, w_n\) such that \(\sum_{i=1}^{n} w_i = 1\), the \textit{weighted (lower) median} is the element \(x_k\) satisfying

\[
\sum_{x_i < x_k} w_i < \frac{1}{2} \quad \text{and} \quad \sum_{x_i > x_k} w_i \leq \frac{1}{2}.
\]

(a) Argue that the median of \(x_1, x_2, \ldots, x_n\) is the weighted median of the \(x_i\) with weights \(w_i = \frac{1}{n}\) for \(i = 1, 2, \ldots, n\).

(b) Show how to compute the weighted median of \(n\) elements in \(O(n \lg n)\) worst-case time using sorting.

(c) Show how to compute the weighted median in \(\Theta(n)\) worst-case time using the linear-time median algorithm we discussed in the class.