**Question 1.** Suppose you are choosing between the following three algorithms:

1. Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
2. Algorithm B solves problems of size $n$ by recursively solving two subproblems of size $n-1$ and then combining the solutions in constant time.
3. Algorithm C solves problems of size $n$ by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in asymptotic notation) and which would you choose?

**Question 2** (K&T 5.1)**.** You are interested in analyzing some hard-to-obtain data from two separate databases. Each database contains $n$ numerical values—so there are $2n$ values total—and you may assume that no two values are the same. You'd like to determine the median of this set of $2n$ values, which we define to be the $n$th smallest value.

However, the only way you can access these values is through *queries* to the databases. In a single query, you can specify a value $k$ to one of the two databases, and the chosen database will return the $k$th smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most $O(\log n)$ queries.

**Question 3.** Recall that the *Fibonacci numbers* $F_n$ are defined recursively by

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2}, \end{aligned}$$

with the first few given by $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$ You may assume the following facts:

- $F_n$ is $O(\phi^n)$ (where $\phi$ is the golden ratio, $(1+\sqrt{5})/2$).
- In addition to their definition, Fibonacci numbers satisfy the following recurrences:

$$\begin{aligned} F_{2n+1} &= F_n^2 + F_{n+1}^2 \\ F_{2n} &= 2F_nF_{n+1} - F_n^2 \end{aligned}$$

Design an efficient algorithm to compute $F_n$, and analyze its running time. Be careful to include the time needed for any multiplications and additions; since the algorithm may need to deal with very large numbers, you may *not* assume that arithmetic operations take $O(1)$ time.

3

**Question 4.** An array $A[1 \ldots n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are *not* necessarily from some ordered domain like the integers, and so there can be no comparisons of the form $A[i] > A[j]$. You should think of the array elements as, say, JPEG files. However, you *can* answer questions of the form: $A[i] = A[j]$ in $O(1)$ time.

4a

(a) Show how to solve this problem in $O(n \log n)$ time. Make sure to prove that your algorithm is correct (via induction) and give a recurrence relation for the running time of your algorithm.

(b) Can you give a linear-time algorithm?

*Extra Credit*

This question adapted from the Design and Analysis of Algorithms course at the University of Konstanz by Dr. Ulrik Brandes and Dr. Sabine Cornelsen.

4b

**Question 5** (Least Common Ancestor)**.** Let $T = (V, E)$ be an oriented tree with root $r \in V$ and let $P \subseteq \{\{u, v\} \mid u, v \in V\}$ be a set of unordered pairs of vertices. For each $v \in V$ let $\Pi_v = \{r, ..., v\} \subseteq V$ denote the sequence of vertices along the path from $r$ to $v$ and let $d(v) = |\Pi_v| - 1$ denote the depth of $v \in T$. The *least common ancestor* of pair $\{u, v\} \in P$ is defined as $\overline{w} \in V$ with $\overline{w} \in \Pi_v \cap \Pi_u$ and $d(\overline{w}) > d(w)$ for all $w \in \Pi_v \cap \Pi_u$.

Algorithm LCA($r$) traverses $T$ to determine the least common ancestors of all pairs $\{u, v\} \in P$ . At the beginning, all vertices are unmarked.

**Algorithm 1:** LCA($u$)

```
1  Makeset (u)
2  ancestor[Find(u)] ⟵ u
3  foreach child v of u in T do
4  |  LCA(v)
5  |  Union(Find(u),Find(v))
6  |  ancestor[Find(u)] ⟵ u
7  end
8  mark u
9  foreach v with {u, v} ∈ P do
10 |  if v is marked then
11 |  |  print "LCA(" + u + "," + v + ") is" + ancestor[Find(v)]
12 |  end
13 end
```

(a) Show that, when Line 8 in LCA($u$) is executed, the set FIND($u$) contains all vertices of the subtree $T_u \subseteq T$ with root $u$.

(b) Show that the number of sets in the Union-Find data structure at the time of a call to LCA($v$) equals $d(v)$.

(c) Prove that LCA($r$) determines the least common ancestors of all $\{u, v\} \in P$ correctly.

(d) Analyze the running time of LCA($r$).

*5a*

*5b*

*5c*