Question 1 (JE)**.** *Describe and analyze a data structure that supports the following operations on an array $A[1 \ldots n]$. Initially, $A[i] = 0$ for all i.*

- SET(i)*: Given an index i such that $A[i] = 0$, set $A[i]$ to* 1.
- GET(i)*: Given an index i, return $A[i]$.*
- SMALLEST(i)*: Given an index i, return the smallest index $j \geq i$ such that $A[j] = 0$, or report that no such index exists.*

For full credit, SET *and* GET *should run in worst-case constant time, and the amortized cost of* SMALLEST *should be equivalent to the amortized cost of the* UNION-FIND *data structure. Here is some help: imagine using a* UNION-FIND *data structure to represent maximal contiguous sets of 1s. The root element of each set may correspond to the smallest 0, but should* know *the index of its smallest 0. That is, you may want to associate a field* ZERO *with each element, which only has meaning for the root node. The implementation of* SET *requires the most care—it is natural to implement this operation as a* UNION*, which requires two* FIND *operations. However,* SET *should only take $O(1)$ time. Thus, you may want to consider adding an additional field,* LEFT*, to each element, that records the leftmost 1 in each set. If the parent of the leftmost element is always the root, then it might be possible to perform a* SET *in constant time.*

Question 2 (DPV)**.** *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 3 (DPV)**.** *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.*

(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. A hint for this problem is available at the end of the homework.

(b) Can you give a linear-time algorithm? A hint for this problem is available at the end of the homework.

Question 4 (JE)**.** *An inversion in an array $A[1 \ldots n]$ is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$. The number of inversions in an n-element array is between 0 (if the array is sorted) and $\binom{n}{2}$ (if the array is sorted backward). Describe and analyze an algorithm to count the number of inversions in an n-element array in $O(n \log n)$ time.*

Question 5. *I found this homework (a) challenging, but fun; (b) challenging, but not fun; (c) fun, but not challenging; (d) not fun and not challenging.*

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

Question 6 (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 $w \in \Pi_v \cap \Pi_u$.*

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.*
Hint: *Do an induction on the height $h(u)$ of u. ($h(u) = 0$ if u is a leaf)*

(b) Show that the number of sets in the Union-Find data structure at the time of call LCA(v) equals $d(v)$.
Hint: *The recursive calls of LCA specify a traversing order of T which implies an order on V . Do an induction on the position of v in this order.*

(c) Prove that LCA(r) determines the least common ancestors of all $\{u, v\} \in P$ correctly.
Hint: *Differentiate the two cases: 1. w.l.o.g. $v \in T_u$ and 2. $v \notin T_u$ and $u \notin T_v$.*

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

Hints

Hint for ?? (a) Split the array A into two arrays A_1 and A_2 of half the size. Does knowing the majority elements of A_1 and A_2 help you figure out the majority element of A? If so, can you use a divide-and-conquer approach?

Hint for ?? (b) Arbitrarily pair up the elements of the array. For each pair, if the two elements are different, discard both of them; if they are the same, keep just one of them. Show that after this procedure executes, there are at most $n/2$ elements left, and that they have a majority element if A does.