1. **(Matroids and the Greedy Algorithm)** Given a universe U of elements, a collection of subsets $\mathcal{I} \subseteq 2^U$ is subset-closed if $B \in \mathcal{I}$ and $A \subseteq B \subseteq U$ implies $A \in \mathcal{I}$: the set system $(U, \mathcal{I})$ is called an *independence system* if $\mathcal{I}$ is subset closed and $\emptyset \in \mathcal{I}$. (In that case we call a set A *independent* if $A \in \mathcal{I}$.) An independence system is called a *matroid* if it satisfies the additional property (*) that if $A, B \in \mathcal{I}$ and $|A| < |B|$ there is an element $x \in B \setminus A$ such that $A \cup \{x\} \in \mathcal{I}$.

   Some more notation: a maximal independent set is called a *base*. A cycle (or circuit) is a minimal *dependent* set. A *cut* is a minimal set that intersects every base. The rank of a matroid is the cardinality of any maximal (and hence *maximum*) independent set in the matroid.

   (a) (Don't hand in) Show that the following set systems are matroids: (a) $\mathcal{I}$ is the collection of all subsets of cardinality at most r, (b) $U = U_1 \uplus U_2 \uplus \ldots \uplus U_\ell$ and $\mathcal{I} = \{A \subseteq U \mid \forall i \in [\ell] |A \cap U_i| \leq r_i$, (c) U is the set of edges of some graph G and $\mathcal{I}$ contains all acyclic subsets of edges, (d) $U = \mathbb{F}^r$ for some field $\mathbb{F}$ and $\mathcal{I}$ contains every set of r-bit vectors that are linearly independent over the field $\mathbb{F}$. What are the ranks of these matroids?

   (b) (Do not hand in) Verify that the following independence system does not form a matroid in general by giving a counterexample: the elements are the edges of an undirected graph $G = (V, E)$ and the independent sets are matchings in G (i.e., subsets of edges such that no two edges share a common endpoint). Show this is true even when G is bipartite.

   (c) (Do not hand in) Show that the cardinality of any two bases in a matroid is the same.

   (d) Given $M = (U, \mathcal{I})$, define the *dual* matroid M^* to have the same set of elements; the independent sets $\mathcal{I}^*$ are the complements of bases in M, as well as all subsets of these bases. Show that M^* is indeed a matroid.

   (e) Given element weights $w : U \to \mathbb{R}$, the min-weight basis problem seeks to find a basis B of minimum weight $w(B) = \sum_{e \in B} w(e)$. Show that repeatedly applying the cut rule (blue rule) and the cycle rule (red rule) to any matroid colors all the edges, and the resulting blue edges form a min-weight basis.

   (f) Consider the natural generalization of Kruskal's algorithm: let $S \leftarrow \emptyset$, sort elements by weight and consider them in non-decreasing order, when considering an element e add it to the S if $S \cup \{e\}$ is independent. Show that this algorithm solves the min-weight basis problem. (You may assume an oracle that answers membership queries for $\mathcal{I}$ in constant time.)

   (g) Show that for an independence system $(U, \mathcal{I})$, if the greedy algorithm correctly solves the min-weight basis problem for any setting of weights, then the independence system is a matroid. Hence the greedy algorithm characterizes matroids.

2. **(Yao's MST Algorithm)** In this problem we will illustrate some of the ideas behind Yao's $O(m \log \log n)$-time algorithm. The essential idea behind this algorithm is to note that if we implement Boruvka's algorithm we scan all the edges in the graph in each pass, and avoiding this repetition should get us an improvement. Assume that the graph G is connected and hence $m \geq n - 1$.

   (a) Suppose for each vertex, the edges adjacent to that vertex are stored in non-decreasing order of weights. Show a slight variant of Boruvka's algorithm that runs in time $O(m + n \log n)$ time.

   (b) (*k-partial sorting*) Given a parameter k and a list of N numbers, give an $O(N \log k)$-time algorithm that partitions this list into k groups $g_1, g_2, \ldots, g_k$ of size at most $\lceil N/k \rceil$ each such that all elements in g_i are smaller than those in g_{i+1} for each i.

(c) Adapt your algorithm from the first part above to handle the case where the edges adjacent to each vertex are not completely sorted but only k-partially-sorted. Ideally, your run-time should be $O(\frac{m}{k}\log n + n\log n)$.

(d) Use the two parts above (setting $k = \log n$), preceded by some additional rounds of Boruvka, to give an $O(m \log\log n)$-time MST algorithm.

3. **(Loose Ends)** In this problem, we will tie up some loose ends from lecture

   (a) *(Linear-time Cleanup)*Show how to implement the *cleanup* algorithm in $O(m)$ time. This algorithm takes as input a graph $G = (V, E)$ with some of the edges colored blue, and outputs a new graph $G' = (V', E')$ where each vertex $v' \in V'$ corresponds to a blue connected component in G, there is an edge $e' = (u', v')$ if there is an edge between the corresponding components in G; the weight of such an edge is the minimum-weight over all such edges in G.

   (b) Given the union-find data structure with n elements, show that doing $\Theta(n)$ unions followed by $\Theta(n)$ finds takes $\Theta(n)$ runtime.

4. **(Spreading Balls)** There are n balls arranged into bins. A step is the following:

   An adversary picks a bin. (Let k be the number of balls in that bin.) The adversary then distributes these k balls arbitrarily to bins subject to the restriction that all of these k balls must be in different bins after the step. The cost of the step is k.

   (a) State and prove a lower bound on the amortized cost of a step.

   (b) State and prove an upper bound on the amortized cost of a step. In class we used "tokens" to analyze Fibonacci Heaps. For this problem it may be more natural to use a potential function, as described here. www.cs.cmu.edu/afs/cs/academic/class/15451-s04/www/Lectures/amortize.pdf.

   These bounds should be functions of n, and they should be close.

5. **(Same or Different)** There are n balls numbered 1 to n. The balls are of two colors. Information about the colors of the balls is revealed gradually in the form of assertions like this:

   same(i, j) asserts that balls i and j are the same color.
   different(i, j) asserts that balls i and j are not the same color.

   If an assertion is inconsistent with what is already known, the algorithm should note this and ignore the assertion. (for example after same(1,2), different(2,3), same(1,3) is inconsistent.) Intermixed with these assertions are querys of this form:

   query(i, j) returns "same" or "different" or "don't know", depending on what can be derived about the relative colors of balls i and j from the previous assertions.

   Give as efficient an algorithm as you can for processing such a sequence of queries. Analyze your algorithm.