1. **(Finding Max-Flows is Useful)** Reduce the following problems to finding max-flows or min-cuts in networks.

   (a) (Buying Sets.) You are given m subsets $S_1, S_2, \ldots, S_m$ of some universe U. Each element $j \in U$ has cost c_j and each set S_i has profit p_i. If you pick a set $A \subseteq U$, you get profit $\pi(A) = \sum_{i:S_i \subseteq A} p_i$, but it costs $c(A) = \sum_{j \in A} c_j$. Find a set A that maximizes $\pi(A) - c(A)$.

   (b) (Hoffman's Theorem.) Consider the following situation: we are given a flow network $G = (V, E)$ with each arc uv having two values $\ell_{vw} \leq u_{vw} \in \mathbb{R}$, and every vertex v having a value $b_v \in \mathbb{R}$ such that $\sum_{v \in V} b_v = 0$.
   Show that there exists a setting $f : E \to \mathbb{R}$ such that

   $$\text{excess}_f(v) = b_v, \text{ for all } v \in V \tag{1}$$
   $$\ell_e \leq f_e \leq u_e, \text{ for all } e \in E. \tag{2}$$

   if and only if for all $A \subseteq V$ it holds that

   $$u(\partial \bar{A}) \geq b(A) + \ell(\partial A).$$

   In words, the upper bound of all edges coming into A should be at least $b(A) = \sum_{v \in A} b_v$ plus the lower bound on the edges that leave A.

2. **(Preflow-Push with Scaling)** The *excess-scaling* version of push-relabel works for max-flow computations with integer capacites. It runs in phases: in phase i, we maintain the invariant that the excess at any active node is at most Δ_i. Once the excesses of all active vertices fall below $\Delta_i/2$, we set $\Delta_{i+1} = \Delta_i/2$ and begin phase $i+1$. To start, we set $\Delta_1 = 2^{\lceil \log_2 U \rceil}$, where $U = \max_{e \in E} u_e$. We maintain the fact that all flows are integral, and stop when $\Delta_i < 1$.

   Consider the following rule: From among the currently active vertices having excess more than $\Delta_i/2$, choose the vertex u with the *smallest* height. Moreover, during a push(vw) move, move flow equal to $\min\{\text{excess}_f(v), u_f(uw), \Delta_i - \text{excess}_f(w)\}$; i.e., the maximum amount of flow that will maintain the invariant.

   (a) Show that the algorithm is correct.

   (b) Show that the number of non-saturating pushes done by this algorithm is $O(n^2 \log U)$.

   (c) Infer that the algorithm runs in $O(nm + n^2 \log U)$ time.

3. **(Bounding Splices in Link-Cut Trees)** In this problem we'll develop an alternative analysis of the link-cut trees described in lecture: here we are just concerned with bounding the number of splices done.

   First consider the simpler situation where there's just one rooted tree T of n nodes. It's partitioned into solid and dashed edges, as described in class. (At most one of the edges from a node to its children is solid.) Define an operation `Access`(x) which makes all the edges from a node x to the root solid. It does this by doing splices wherever there's a dashed edge on the path. The cost of the operation is the number of splices done.

   (a) Define a potential function, and use it to prove that the amortized cost of `Access`(x) is at most $2\log(n)$.

   (b) Obtain a bound on the cost of m `Access` operations in a tree of n nodes. (The initial partitioning of the tree into solid and dashed edges is arbitrary.)

   Now consider a dynamically changing forest of trees with n nodes. In addition to `Access`() we define two additional operations:

`Link`(x, y): x is a root of a tree, and y is a node in another tree. This operation makes y the parent of x. The new edge between them is dashed.

`Cut`(x): This operation removes the edge from x to the parent of x so that x becomes the root of its tree.

(c) Extend the potential you developed in the first part to this situation. Use it to obtain an amortized cost of $\log(n)$ for `Link`() and `Cut`(). (Note that the actual cost of each of these operations is zero, because they do no splices.)

Note: We proved essentially the same thing in class. (Actually we proved $6\log(n) + 3$ splices (amortized). See `http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15859-f09/www/handouts-/link-cut-trees.txt`.) Our method was very indirect, and was built upon the analysis of splaying.

4. **(Red-Blue Separation)** Suppose you are given a set $S = B \cup R$ of m points in $\mathbb{R}^n$. You are guaranteed that there is a "consistent" half-space $H = \{x \mid p^T x \geq q\}$ with $p \in \mathbb{R}_{n\times 1}, q \in \mathbb{R}$ that contains all the "blue" points ($B \subseteq H$) and none of the "red" points ($R \cap H = \emptyset$).

   Write a linear-program to find a consistent half-space for given set of red and blue points.

5. **(PAC Learning)** In 1984, Valiant proposed the *PAC* or *probably approximately correct* model for learning. Say you want to learn an unknown "concept" c (think of c as a subset of some universe U, or a function $\chi_c : U \to \{+, -\}$ that has $\chi_c(e) = 1 \iff e \in c$), from some class $\mathcal{C} \subseteq 2^U$ of concepts. There is some (unknown) underlying probability distribution $\mathcal{D}$ over U. All you are given is a black box that draws an independent random sample $e \in U$ according to the probability distribution $\mathcal{D}$, and gives you the tuple $(e, \chi_c(e))$.

   For the purposes of this exercise, your goal is to devise a randomized algorithm that given $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, returns some concept $c' \in \mathcal{C}$ such that

   $$\Pr_{e\in\mathcal{D}}[\chi_{c'}(e) \neq \chi_c(e)] \leq \epsilon$$

   with probability $1 - \delta$. In other words, the concept returned is approximately correct (i.e., it agrees with the target concept c on $1 - \epsilon$ fraction of the probability according to $\mathcal{D}$) with probability $1 - \delta$. In this case we say the pair $(c, \mathcal{D})$ is *PAC-learnable*. A *concept class $\mathcal{C}$ is PAC-learnable* for any concept $c \in C$ and any probability distribution $\mathcal{D}$, the pair $(c, \mathcal{D})$ is PAC-learnable.

   (a) Let us consider the case where the universe is $\mathbb{R}^2$, and the concepts are axis-aligned rectangles in 2-dimensions. Consider the following algorithm:

   Take N samples $(x_1, v_1), (x_2, v_2), \ldots, (x_N, v_N)$ from the black box. Some of these have label $v_i = +$ and others $v_i = -$. Draw the smallest axis-aligned rectangle $\widehat{R}$ that contains all the $+$'s and none of the $-$'s.

   i. Infer that if R was the unknown target concept rectangle, then $\widehat{R} \subseteq R$.

   ii. Suppose $R = \{x \in \mathbb{R}^2 \mid a \leq x_1 \leq b, c \leq x_2 \leq d\}$. Consider the smallest value d' such that the rectangle $A = \{x \in \mathbb{R}^2 \mid a \leq x_1 \leq b, c \leq x_2 \leq d'\}$ contains at least ϵ probability mass according to $\mathcal{D}$. What is the probability that none of the N sample points land within A?

   iii. Use the above argument four times to infer that the rectangle $\widehat{R}$ is probably approximately correct for the target concept R, if $N = \Omega(\frac{1}{\epsilon}\log\frac{1}{\delta})$.

   If the running time is polynomial in $1/\epsilon$ and $1/\delta$, we say that the concept class $\mathcal{C}$ is *efficiently PAC-learnable*. Infer from the above parts that the concept class of axis-aligned rectangles in $\mathbb{R}^2$ is efficiently PAC-learnable.

   (b) Extend your algorithm to handle axis-aligned rectangles in $\mathbb{R}^d$. How does the number samples N depend on the dimension d?

6. **(Simple Sample)** You are given sampling access to a real-valued random variable X which takes on values in $[0, 1]$—each time you query it, you get a independent sample from this random variable. Let $\mu = E[X]$. Give an algorithm that needs $O(\frac{1}{\epsilon}^2 \log\frac{1}{\delta})$ random samples and outputs a value $\hat{\mu}$ such $\Pr[|\hat{\mu} - \mu| > \epsilon] \leq \delta$.

   How many samples do you need if the random variable takes on values in $[-a, a]$ for $a > 0$?