Due on November 26th, 2014
100 points total

General Directions: If you are asked to provide an algorithm, you should clearly define each step of the procedure, establish its correctness, and then analyze its overall running time. There is no need to write pseudo-code; an unambiguous description of your algorithm in plain text will suffice.

All the answers must be typed, preferably using LaTeX. If you are unfamiliar with LaTeX, you are strongly encouraged to learn it. However, answers typed in other text processing software and properly converted to a pdf file will also be accepted. Before submitting the pdf file, please make sure that it can be opened using any standard pdf reader (such as Acrobat Reader) and your entire answer is readable. **Handwritten answers or pdf files that cannot be opened will not be graded and will not receive any credit.**

Finally, please read the detailed collaboration policy given on the course website. You are **not** allowed to discuss homework problems in groups of more than 3 students. **Failure to adhere to these guidelines will be promptly reported to the relevant authority without exception.**

Problem 1 (20 points)
Let X be a set of n points in the plane. A point p in X is *Pareto-optimal* if no other point in X is both above and to the right of p. The Pareto-optimal points can be connected by horizontal and vertical lines into the *staircase* of X, with a Pareto-optimal point at the top right corner of each step.

(i) Describe an algorithm to compute the staircase of a given set of n points in the plane in $O(nh)$ time, where h is the number of Pareto-optimal points.

(ii) Describe an algorithm to compute the staircase of a given set of n points in the plane in $O(n \log n)$ time.

In both parts, you may assume that no two points have the same x- or y-coordinates. The output should be a linked list of Pareto-optimal points in left to right order.

Problem 2 (20 points)
(Revisiting Huffman Codes) Let Σ be an alphabet. For each character $c \in \Sigma$, let p_c be the given frequency of c given as a probability; therefore, $\sum_{c\in\Sigma} p_c = 1$.

Our goal here is to compute a binary *prefix* encoding for the characters in Σ. Namely, our algorithm must map each character in c to a unique binary string b_c with length ℓ_c such that b_c is not the first ℓ_c characters of another binary string $b_{c'}$ in the encoding (this is our "prefix" constraint). The objective is to find the encoding that minimizes $\sum_{c\in\Sigma}(\ell_c \cdot p_c)$ (which in some sense is the expected length of a document written in alphabet Σ that uses the computed encoding).

One way to represent such an encoding is with a binary tree T: Each character $c \in \Sigma$ corresponds to a leaf in T. To obtain the encoding for a character c, we build b_c by repeatedly concatenating 0s and 1s based on the path from the root to c in T. Starting at the root, if we traverse to a left child we concatenate a 0, and if we traverse to a right child we concatenate a 1 (it is straightforward to show that this process does indeed give us a prefix encoding).

It is likely you saw *Huffman trees* in a previous course. In this problem, you will prove the code yielded from a Huffman tree is optimal over all binary tree encodings. Recall the algorithm for computing a Huffman tree: we first initialize the subtree set $S = \{c | c \in \Sigma\}$ (so at the outset, S is just a collection of unconnected vertices, one vertex for each character in Σ). We incrementally build Huffman tree T_H by repeatedly merging subtrees in the following way:

1. Let T and T' be the two subtrees in S with the lowest total frequencies (where the total frequency of a subtree T is $\sum_{c\in T} p_c$, i.e., the summed frequencies of the characters at the leaves of T).

2. Remove T and T' from S.

3. Create a subtree $T' \cup T$ by making the T and T' the left and right subtrees of a newly created root r.

4. Add $T \cup T'$ to S.

5. If $|S| > 1$, repeat from Step 1.

The result of this algorithm is a binary tree whose leaves are exactly the elements in Σ, which in turn gives us the encoding described earlier. Show that for all binary trees T whose leaves are exactly the elements in Σ, T_H is the one which minimizes

$$\sum_{c \in \Sigma} (p_c \cdot \ell_c),$$

where ℓ_c is the depth of c in T.

Hint: Try doing the following induction: Assume after i iterations/merges, each of the currently existing $n - i$ subtrees $T_1, \ldots, T_{n-i}$ (sorted by non-decreasing total frequency) are subtrees in some optimal binary tree T. With this inductive assumption, argue that the $n - i - 1$ subtrees $T_1 \cup T_2, \ldots, T_{n-i}$ are subtrees in some optimal binary tree T' (noting that is may be possible for $T' = T$).

Problem 3 (30 points)
Consider the following randomized algorithm for computing a spanning tree S for an *unweighted* undirected graph $G = (V, E)$ where $|V| = n$ and $|E| = m$ (so note this is *not* a minimum spanning tree). Denote E_v as the set of edges that are incident on vertex $v \in V$:

1. Pick an arbitrary root $r \in V$.

2. Starting at r, begin traversing the graph randomly, i.e., when sitting a given vertex v pick an edge $e = (v, u)$ from E_v uniformly at random and then travel along e to u.

3. Whenever we travel along an edge e and reach a vertex we have not yet seen, add e to our spanning tree S.

We continue this random traversal until we observe every vertex. Let p_e be the probability that edge e is added to S during the random process. The following questions ask you to analyze this algorithm.

(a) (5 points) Prove that the edges in S at the end of the algorithm in fact form a spanning tree.

(b) (10 points) Argue that $\sum_{e \in E} p_e = n - 1$.

(c) (15 points) For an edge $e = (u, v)$, define $d_e = \min(\deg(u), \deg(v))$ (recall that $\deg(v) = |E_v|$ is the *degree* of a vertex v). Prove that $\sum_{e \in E}(p_e \cdot d_e) \leq 2m$.

(Hint: in your analysis for parts b and c, consider using $p_e(u)$: the probability edge $e = (u, v)$ is added to the spanning tree as the result of traversing edge e from u to v during the random walk).

Problem 4 (30 points)
Let $G = (V, E)$ be a unit-capacity graph with n vertices and m edges. In this question, we will revisit the randomized edge contraction algorithm for the global min-cut problem we saw in class. In this setting, the contraction algorithm will do one additional contraction so that the final graph is just a single vertex (instead of being left with two vertices which define a cut).

(a) (5 points) Show that in any run of the edge contraction algorithm, the edges contracted form a spanning tree of G.

(b) (15 points) Let $\mathcal{T}$ denote all the spanning trees in G. If we run the contraction algorithm, we will get a random spanning tree in $\mathcal{T}$ formed by the contracted edges, and we denote this distribution of spanning trees by $\mathcal{D}_1$. On the other hand, if we assign a random weight in $(0, 1)$ to each edge and compute a minimum spanning tree using Kruskal's algorithm, then we obtain another distribution $\mathcal{D}_2$ over $\mathcal{T}$. Show that these two distributions are identical.

(c) (5 points) Using the analysis we saw in class for the contraction algorithm, argue that there are $O(n^2)$ *global min-cuts* in any graph.

(d) (5 points) Observe that part (c) implies nothing about the number of s-t *min-cuts* there can be for two vertices $s, t \in V$. Give an example of a graph where there are $\omega(n^2)$ s-t min-cuts for a particular pair of vertices s and t. Recall that a function $g(n) = \omega(n^2)$ if $g(n)$ is *strictly* greater than n^2 asymptotically , i.e., $g(n) = \Omega(n^2)$ but $g(n) \neq \Theta(n^2)$ (so functions like n^3, n^{10}, 2^n, etc.).

Note that this counterexample needs to be constructed based on a general parameter n since we are attempting to show an asymptotic lower bound. This means you should describe a graph with n vertices, define the edge set, and then argue why there are $\omega(n^2)$ min-cuts. Diagramming your example might be difficult. Likely, the easiest way to give your answer is to just describe the structure of your graph in words (although, if you think you can provide a clear diagram, feel free to do so).