

1. Recall Karger's min-cut algorithm:

```

While number of vertices > 2:
    Select edge  $e = (u, v)$  uniformly at random.
    Merge  $u$  and  $v$  into a single vertex.
Return remaining edges.
    
```

We proved that this returns a minimum cut in the original graph with probability at least  $\frac{1}{n^2}$ , where  $n = |V|$ .

- (a) Describe a data structure that allows each iteration of the while loop to be performed in  $O(n)$  time, and justify this bound. Describe how this allows us to find a minimum cut with probability at least  $1 - \epsilon$  in time  $O(n^4 \log \epsilon^{-1})$ .

We next consider how to improve on this run time. The idea behind this improvement is that the probability of the min-cut algorithm hitting an edge in the minimum cut is fairly small for the first few edges that are chosen, but increases with each merge that is performed. Thus, we should be repeating the merges at the later stages of the algorithm more times than the merges at the early stages.

Let us consider how many merges we can perform before the *total* probability of hitting a particular minimum cut  $C$  is  $\leq \frac{1}{2}$ . From the argument in lecture, the probability that  $C$  survives down to  $\ell$  vertices is at least  $\frac{\ell(\ell-1)}{n(n-1)}$ . You should verify this (but you don't need to turn this in.) If we take  $\ell = \lceil n/\sqrt{2} \rceil$ , this probability is  $\frac{1}{2} - O(1/n)$ . Therefore, we need only to repeat these early stages  $\log \epsilon^{-1}$  times (rather than  $n^2 \log \epsilon^{-1}$  times, as in part (a)) for  $C$  to survive down to  $\lceil n/\sqrt{2} \rceil$  vertices at least once with probability  $\epsilon$ . To finish off the process, we can then apply the same procedure recursively to the remaining  $\lceil n/\sqrt{2} \rceil$  vertex graph. The resulting algorithm, Rec-MinCut, is as follows:

```

Rec-MinCut(G)
    If number of vertices = 2 return edges of  $G$ .
    Else repeat twice
        By repeated merging, reduce  $G$  to an  $\lceil n/\sqrt{2} \rceil$  vertex graph  $G'$ .
        Rec-MinCut( $G'$ )
    Return the smaller of the two cuts returned by the recursive calls.
    
```

- (b) Write down a recurrence relation for the running time,  $T(n)$  of Rec-MinCut on a graph with  $n$  vertices. Use this recurrence relation to show that  $T(n) = O(n^2 \log n)$ . To do so, you can ignore floors and ceilings.
- (c) In an  $n$ -vertex graph  $G$ , let  $P(n)$  be the probability that a particular minimum cut survives to the end of Rec-MinCut( $G$ ). Show that  $P(n)$  satisfies the recurrence relation

$$P(n) = P\left(\left\lceil \frac{n}{\sqrt{2}} \right\rceil\right) - \frac{1}{4}P\left(\left\lceil \frac{n}{\sqrt{2}} \right\rceil\right)^2 - O\left(\frac{1}{n}\right), \text{ for } n > 2,$$

and  $P(2) = \Theta(1)$ .

(d) The solution to the above recurrence is  $P(n) = \Theta(1/\log n)$ . (You do not need to show this.) Use this fact to show that Rec-MinCut can be used to find a minimum cut in an  $n$ -vertex graph with probability  $1 - \epsilon$  in time  $O((n \log n)^2 \log \epsilon^{-1})$ .

2. Say we are given  $n$  distinct numbers  $i_1, \dots, i_n$ , and we wish to obtain an ordering on these numbers. However, instead of being able to compare the numbers directly, we are only given a set of constraints of the following form:  $i_a$  is between  $i_b$  and  $i_c$ . In other words, we are told that either  $i_b < i_a < i_c$  or  $i_c < i_a < i_b$ , but the constraint does not specify which. Given a set of such constraints, it is not always possible to satisfy all of them simultaneously. Thus, our goal is to determine an ordering on  $i_1, \dots, i_n$  that satisfies as many of them as possible. Unfortunately, given  $C$  constraints and a value  $C^* \leq C$ , deciding if there is an ordering that satisfies  $C^*$  constraints is an NP-Complete problem. (This means that we don't expect to be able to design an efficient algorithm for the problem of maximizing the number of constraints satisfied.)

Provide an algorithm for ordering  $i_1, \dots, i_n$  with the following property: for a given set of constraints, if it is possible to satisfy  $T$  constraints, then your algorithm satisfies at least  $T/4$  constraints. Your algorithm can use randomization, but should guarantee (with probability 1) that  $T/4$  constraints are satisfied, and should have an expected running time that is polynomial.

3. Two rooted trees  $T_1$  and  $T_2$  are said to be *isomorphic* if there exists a one-to-one and onto mapping  $f$  from the vertices of  $T_1$  to those of  $T_2$  satisfying the following condition: for each internal vertex  $v$  of  $T_1$  with children  $v_1, \dots, v_k$ , the vertex  $f(v)$  has as children exactly the vertices  $f(v_1), \dots, f(v_k)$ . Note that no ordering is assumed on the children of any internal vertex. Devise an efficient randomized algorithm for testing the isomorphism of rooted trees, and analyze its performance. Your algorithm should be based on the following idea: associate a polynomial  $P_v$  with each vertex  $v$  in a tree. The polynomials are defined recursively, the base case being that the leaf vertices all have  $P = x_0$ . An internal vertex  $v$  of height  $h$  with children  $v_1, \dots, v_k$  has its polynomial defined to be

$$(x_h - P_{v_1})(x_h - P_{v_2}) \dots (x_h - P_{v_k}).$$

Note that there is exactly one variable for each level in the tree.

4. Consider again the problem of placing  $n$  balls into  $n$  bins, where each ball is placed independently and uniformly at random. In this question, you can use (without proof) the fact that  $\lim_{k \rightarrow \infty} (1 - 1/k)^k = \frac{1}{e}$ .
- (a) Let  $E(n)$  be the expected number of bins that do not receive any balls. What is the limit  $\lim_{n \rightarrow \infty} E(n)/n$ ? Prove your answer.
- (b) We now assume that the balls are placed sequentially, and if a ball chooses a bin that is already occupied, it does not get placed in that bin, and is lost from the system. Let  $L(n)$  be the expected number of balls that are lost in this way. What is the limit  $\lim_{n \rightarrow \infty} L(n)/n$ ? Prove your answer.
- (c) Next, assume that each bin can hold two balls. Thus, the first two balls that choose a bin are successfully placed there, but any further balls that choose that bin are lost from the system. Let  $L'(n)$  be the number of balls that are lost in this way. What is the limit  $\lim_{n \rightarrow \infty} L'(n)/n$ ? Prove your answer.