12. Consider the following analogue of Karger's algorithm for finding minimum s-t cuts. We will contract edges iteratively using the following randomized procedure. In a given iteration, let s and t denote the possibly contracted nodes that contain the original nodes s and t, respectively. To make sure that s and t do not get contracted, at each iteration we delete any edges connecting s and t and select a random edge to contract among the remaining edges. Give an example to show that the probability that this method finds a minimum s-t cut can be exponentially small.

11-4: k-universal hashing and authentication

Let \( \mathcal{H} = \{ h \} \) be a class of hash functions in which each \( h \) maps the universe \( U \) of keys to \( \{0, 1, ..., m - 1\} \). We say that \( \mathcal{H} \) is \( k \)-universal if, for every fixed sequence of \( k \) distinct keys \( \langle x^{(1)}, x^{(2)}, ..., x^{(k)} \rangle \) and for any \( h \) chosen at random from \( \mathcal{H} \), the sequence \( \langle h(x^{(1)}), h(x^{(2)}), ..., h(x^{(k)}) \rangle \) is equally likely to be any of the \( m^k \) sequences of length \( k \) with elements drawn from \( \{0, 1, ..., m - 1\} \).

a) Show that if \( \mathcal{H} \) is 2-universal, then it is universal.

b) Let \( U \) be the set of \( n \)-tuples of values drawn from \( \mathbb{Z}_p \), and let \( B = \mathbb{Z}_p \), where \( p \) is prime. For a \( n \)-tuple \( a = \langle a_0, a_1, ..., a_{n-1} \rangle \) of values from \( \mathbb{Z}_p \) and for \( b \in \mathbb{Z}_p \), define the hash function \( h_{a,b} : U \to B \) on an input \( n \)-tuple \( x = \langle x_0, x_1, ..., x_{n-1} \rangle \) from \( U \) as \( h_{a,b}(x) = \left( \sum_{j=0}^{n-1} a_j x_j + b \right) \mod p \) and let \( \mathcal{H} = \{ h_{a,b} \} \). Show that \( \mathcal{H} \) is 2-universal.

c) Suppose that Alice and Bob agree secretly on a hash function \( h_{a,b} \) from a 2-universal family \( \mathcal{H} \) of hash functions. Later, Alice sends a message \( m \) to Bob over the Internet, where \( m \in U \). She authenticates this message to Bob by also sending an authentication tag \( t = h_{a,b}(m) \), and Bob checks that the pair \( (m,t) \) he receives satisfies \( t = h_{a,b}(m) \). Suppose that an adversary Eve intercepts \( (m,t) \) en route and tries to fool Bob by replacing the pair \( (m,t) \) with a different pair \( (m',t') \). Argue that the probability that Eve succeeds in fooling Bob into accepting \( (m',t') \) is at most \( 1/p \), no matter how much computing power the adversary has.
8-4 Water Jugs

Suppose that you are given $n$ red and $n$ blue water jugs, all of different shapes
and sizes. All red jugs hold different amounts of water, as do the blue ones.
Moreover, for every red jug, there is a blue jug that holds the same amount of
water, and vice versa.

Your task is to find a grouping of the jugs into pairs of red and blue jugs that hold
the same amount of water. To do so, you may perform the following operation:
pick a pair of jugs in which one is red and one is blue, fill the red jug with water,
and then pour the water into the blue jug. This operation will tell you whether the
red or the blue jug can hold more water, or that they have the same volume.
Assume that such a comparison takes one time unit.

Your goal is to find an algorithm that makes a minimum number of comparisons
to determine the grouping. Remember that you may not directly compare two red
jugs or two blue jugs.

a Describe a deterministic algorithm that uses $\Theta(n^2)$ comparisons to group
the jugs into pairs.

b Prove a lower bound of $\Omega(n \lg n)$ for the number of comparisons that an
algorithm solving this problem must make.

c Give a randomized algorithm whose expected number of comparisons is
$O(n \lg n)$, and prove that this bound is correct. What is the worst-case
number of comparisons for your algorithm?

8-6 Lower Bound on Merging Sorted Lists

The problem of merging two sorted lists arises frequently. We have seen a
procedure for it in the context of MERGESORT. In this problem, we will prove a
lower bound of $2n - 1$ on the worst-case number of comparisons required to
merge two sorted lists, each containing $n$ items.

First we will show a lower bound of $2n - o(n)$ comparisons by using a decision tree.
Given 2n numbers, compute the number of possible ways to divide them into two sorted lists, each with n numbers.

Using a decision tree and your answer to part (a), show that any algorithm that correctly merges two sorted lists must perform at least 2n - o(n) comparisons.

Now we will show a slightly tighter 2n - 1 bound.

Show that if two elements are consecutive in the sorted order and from different lists, then they must be compared.

Use your answer to part c to show a lower bound of 2n - 1 comparisons for merging two sorted lists.

21-1: Off-line minimum

The off-line minimum problem asks us to maintain a dynamic set T of elements from the domain {1, 2, ..., n} under the operations INSERT and EXTRACT-MIN. We are given a sequence S of n INSERT and m EXTRACT-MIN calls, where each key in {1, 2, ..., n} is inserted exactly once. We wish to determine which key is returned by each EXTRACT-MIN call. Specifically, we wish to fill in an array extracted[1..m], where for i = 1, 2, ..., m, extracted[i] is the key returned by the i-th EXTRACT-MIN call. The problem is "off-line" in the sense that we are allowed to process the entire sequence S before determining any of the returned keys.

In the following instance of the off-line minimum problem, each INSERT is represented by a number and each EXTRACT-MIN is represented by the letter E:

4, 8, E, 3, E, 9, 2, 6, E, E, 1, 7, E, 5.

Fill in the correct values in the extracted array.

To develop an algorithm for this problem, we break the sequence S into homogeneous subsequences. That is, we represent S by

I_1, E, I_2, E, I_3, ..., I_m, E, I_{m+1},

where each E represents a single EXTRACT-MIN call and each I_j represents a (possibly empty) sequence of INSERT calls. For each subsequence I_j, we initially place the keys inserted by these operations into a set K_j, which is empty if I_j is empty. We then do the following.
OFF-LINE-MINIMUM($m, n$)

1. for $i \leftarrow 1$ to $n$
2.   do determine $j$ such that $i \in K_j$
3.   if $j \neq m + 1$
4.     then $\text{extracted}[j] \leftarrow i$
5.     let $l$ be the smallest value greater than $j$
6.     for which set $K_l$ exists
7.     $K_l \leftarrow K_j \cup K_l$, destroying $K_j$
8. return $\text{extracted}$

b) Argue that the array $\text{extracted}$ returned by OFF-LINE-MINIMUM is correct.

c) Describe how to implement OFF-LINE-MINIMUM efficiently with a disjoint-set data structure. Give a tight bound on the worst-case running time of your implementation.

---

∞-∞ Sorting big numbers

Consider a situation where you must sort $n$ numbers that potentially will be represented over several computer words. Assume $m$ is an upper bound on the number of blocks for each number. Analyze the complete running-times of merge-sort and radix-sort (as a function of $n$ and $m$) taking all operations into consideration: comparing, copying and so on.