1 Communication game [10 points]
Consider a modified version of the communication games described in the class: an adversary gives two players $A$ and $B$ two $n$-bit integers $a$ and $b$ respectively. The objective of the game is for $B$ to check whether $a = b$. To help $B$ decide, $A$ is allowed to send a binary string followed by an unique terminating character ($\{0, 1\}^* \perp$). $B$ is then allowed to do any computation with inputs $b$ and the string received from $A$, and either output “EQUAL” or “NOT-EQUAL”. Show that any correct deterministic protocol requires $A$ to transmit at least $n$ bits in the worst case (not counting $\perp$). Give precise arguments.

SOLUTION: First note that in any deterministic protocol, the string sent by $A$ to $B$ depends only on the integer assigned to $A$. Further, the output of $B$ depends only on the string sent by $A$ and the integer $B$ has. Suppose if possible that there is a deterministic protocol which needs at most $k$ bits ($k < n$). There are only $2^{k+1} - 1 < 2^n$ unique strings that have at most than $k$ bits. Therefore, it must be the case that $A$ sends the same string (say $s$) for at least one pair of integers, say $a_1, a_2$. Suppose that according the protocol, when $B$ sees string $s$ and its input is $b$, it answers with $Ans_s(b)$. If $Ans_s(a_1)$ is “EQUAL”, then the adversary can assign $a_2$ to $A$ and $a_1$ to $B$ and the protocol fails. If $Ans_s(a_1)$ were “NOT-EQUAL”, then the adversary assigns $a_1$ to both $A$ and $B$, and the protocol fails. Either way, there is case where the ($< n$-bit communication) protocol fails.

2 Tape Machines [10 points]
You are given a tape machine which consists of a processor, a RAM, and a read-write tape. On the tape is an unsorted, but contiguous, array of $[0.9 \times 2^K]$ words of $K$-bits each. Basic operations (arithmetic etc.) on $K$-bit words cost one unit each. The tape head is initially at the beginning of the array. To read or write a word $w$ on the tape, the tape head should be
moved to the word \( w \). To move the tape head by \( p \) \((K\)-bit\) words costs \( p \) units. Give an algorithm that outputs a \( K \)-bit word that is not in the array on the tape given the following constraints:

(a) \([2 \text{ points}]\) The machine has unlimited RAM, and your algorithm has cost \( O(2^K) \).

(b) \([8 \text{ points}]\) The machine has only 20 words of RAM, and your algorithm has expected cost \( O(2^K) \).

Example: If \( K = 2 \), and the tape had the array \([ 01 \mid 11 \mid 00 ]\), your algorithm should eventually output 10.

**SOLUTION:** (a) Algorithm:

1. First, initialize a \( 2^K \)-entry array in RAM, with all zeros.  
2. Read the word from the head’s current position on tape into RAM (in a different location than the array). 
3. Set the entry stored at the position indexed by this \( K \)-bit integer to 1. 
4. If this is not the last of the \( 0.9 \times 2^K \) words on the tape, advance the head by one and go to step 2. 
5. Sequentially check each entry in the array in RAM until finding the first one storing a 0, then output the \( K \)-bit integer index of that entry.

Correctness: Any entry in the array in RAM will be 1 at the end if and only if its index gets read from the tape at some point (in step 2), which means it is one of the \( 0.9 \times 2^K \) words on the tape. Thus, any entry in the array in RAM storing a 0 will have an index not listed on the tape, and at least \( 0.1 \times 2^K \) such entries exist, so the algorithm will output a \( K \)-bit word that is not on the tape.

Time complexity: The initialization step (step 1) takes \( O(2^K) \) cost (RAM writes and counter increments). Each word on tape gets read once, accompanied by a write operation in the RAM array, and a head move. The final step (step 5) takes an additional \( O(2^K) \) RAM read operations, counter increments, and comparison operations.

So in total we have \( O(2^K) \) operations (RAM reads, RAM writes, counter increments, comparison operations, tape reads, head moves).

(b) Algorithm:

Generate a random \( K \)-bit word (store it in one word of memory).

Read the word from the head’s current position on the tape into RAM (in a different location than the random word), and test whether it is equal to the stored random word. If it is not equal and this is the last word on the tape,
output the random word currently stored in RAM Else if it is not equal, advance the head by one more word on the tape and repeat the previous step Else if it is equal, move the head back to the start, generate a new random K-bit word (overwriting the original one), and repeat the previous step.

Correctness : We have only two words in RAM at a time, so even if the result of the comparison operation requires a bit of RAM to store, we need at most 3 words of RAM. Furthermore, the algorithm only returns if it has gone through the entire array on the tape without finding a word equal to the random one stored in RAM, so if it returns it definitely returns a word not stored on the tape.

Time complexity:
A random word has 0.1 probability of not being on the tape. So the expected number of random words we need to generate before getting one not on the tape is 10 (expectation of a Geometric(0.1) random variable). For each random word we generate, we move the head, read from tape, and compare for equality to the random word, each at most the $0.9 \times 2^K$ times; if the comparison says they are equal, then we also need to move the head back to the beginning, which takes only an additional $0.9 \times 2^K$ moves. Thus, the expected total number of moves, reads, and comparisons is at most $10 \times 4 \times 0.9 \times 2^K = O(2^K)$.

3 Treaps [10 points]
In homework 2, you had shown how to implement some fast set operations on a completely ordered set of elements by representing such sets as treaps. Give an algorithm that takes as input treaps representing sets $A$, $B$ (let $a = \min\{|A|,|B|\}, b = \max\{|A|,|B|\}$) to compute the set difference $A \setminus B$. Your algorithm must run in time $O(a \log((a+b)/a))$. You may use Cut and Splice operations and assume that their costs are as indicated (same as the one described in the homework):

- **Cut** $(x,T) \rightarrow (T_L,T_R)$: splits a random treap $T$, with priorities to each node assigned independently and randomly, into two random treaps $T_L$ and $T_R$ so that all elements in $T_L$ ($T_R$) are smaller (larger) than $x$. If $x$ is found in $T$, the node containing $x$ is also returned. Expected cost: $O(\log |T_L| + \log |T_R|)$

- **Splice** $(T_L,T_R) \rightarrow T$: given two random treaps $T_L,T_R$ such that
every element in $T_L$ is smaller than every element in $T_R$, they are merged into one treap. Expected cost: $O(\log |T_L| + \log |T_R|)$

You may use the following result for evaluating the cost of your algorithm:

Let $f$ be a function whose domain is the set of all ordered pair of sets $(X, Y)$ such that $X \cap Y = \emptyset$, $X \cup Y = \{1, 2, \ldots, q\}$ for some integer $q$. The range of $f$ is $\mathbb{N}$. If $N$ and $M$ ($|N| = n$, $|M| = m$) are such that $N \cup M = \{1, 2, \ldots, n+m\}$, and the notation $X < i$ for a set $X$ of integers denotes $\{x \in X | x < i\}$ (similarly define $X > i$), and $f$ satisfies the recurrence relation

$$(n + m)f(N, M) = C_1(n + m) + C_2(n \log m + m \log n)$$

$$+ \sum_{i=1}^{n+m} (f(N < i, M < i) + f(N > i, M > i)),$$

with the base case $f(\{1, 2, \ldots, k\}, \emptyset) < C_3$ and $f(\emptyset, \{1, 2, \ldots, l\}) < C_4$ for all natural numbers $k, l$ $(C_1, C_2, C_3, C_4$ being some constants), then $f$ satisfies $f(N, M) = O(m \log((n + m)/m))$.

4 “Spring” break [20 points]

It is a spring break morning, and you get out of your bed hoping for a pleasant bike ride to a favorite coffee place. It being Pittsburgh, you step out of your apartment to find that overnight precipitation and low temperatures have left icy patches on the road. Determined to ride your bike, you set about making your way avoiding the icy patches. Your goal is to figure out the shortest path to get to your morning cup.

We model the problem as follows: you start at point $s$ on a plane and intend to get to point $t$ on the same plane (Pittsburgh suddenly became a flat world!). On this plane are some convex polygons of finite size (representing the icy patches). Further, no two polygons share a common point. The positions of the vertices of these polygons (set $V$) are given and add up to $n$ in number. Your goal is to write an algorithm to find the length of the shortest path in the plane from $s$ to $t$ that does not overlap with the interior of any polygon (see figure for an example of a valid path). You may assume that points in $V \cup \{s, t\}$ are distinct and have integral co-ordinates. Assume that asking for the square root of an integer will give you a floating point number that is at most $\epsilon$ away from the real answer. You are allowed to do arithmetic on floating point with out loss of accuracy. Your algorithm’s answer has to be accurate to within $\pm n^2 \epsilon$. 
Figure 1: A path from $s$ to $t$ that avoids the interiors of polygons. $|V| = 16$.

You may first (not necessarily) want to solve the following problem or some variant: Given a set $A$ of $n(n-1)/2$ line segments in a plane representing the set of all line segments between some $n$ points with integral co-ordinates, and a set $B$ of $n$ line segments representing the boundaries of a set of convex polygons, identify all the segments in $A$ that do not cross any segment in $B$.

(a) [10 points] Give an algorithm that runs in time $O(n^3)$.
(b) [10 points] Give an algorithm that runs in time $O(n^2 \log n)$.
(c) [Extra credit] Give an algorithm that runs in time $O(n^2)$ or better.