Exam

Notation.

- $[a, b]$ denotes the integer interval $\{a, a+1, \ldots, b\}$.
- $x \leftarrow X$ means that x is sampled uniformly at random from the set X.

Exercise 1. Range proof. Let $\mathbb{G}$ be a cyclic group of prime order p, generated by g. Both $\mathbb{G}$ and g are public. Let us recall the Schnorr protocol, which allows a prover Alice to prove to a verifier Bob that she knows the (secret) discrete logarithm x of a (public) element $h = g^x$.

1. Alice samples $r \leftarrow \mathbb{Z}_p$ and sends $t = g^r$ to Bob.
2. Bob samples $c \leftarrow \mathbb{Z}_p$ and sends c to Alice.
3. Alice computes $s = r - xc$ and sends s to Bob. Bob accepts the proof iff $t = g^s h^c$.

As a reminder, to prove that this protocol is zero-knowledge, we can build the following simulator : the simulator first picks c and s independently and uniformly at random, then computes $t = g^{s+xc}$. The simulated transcript is (t, c, s).
Now suppose we have two group elements $h_1 = g^{x_1}$ and $h_2 = g^{x_2}$. Alice wants to prove to Bob that she knows at least one of the two discrete logarithms, *i.e.* she knows either x_1 or x_2 (possibly both). To do that, Alice uses the following protocol. Unfortunately, someone spilled coffee on the description of the protocol, and some parts of it are not readable.

1. Alice picks r_1, r_2 in $\mathbb{Z}_p$ as follows: [unreadable]. She sends $t_1 = g^r$ and $t_2 = g^{r_2}$ to Bob.
2. Bob samples $c \leftarrow \mathbb{Z}_p$ and sends c to Alice.
3. Alice computes (c_1, s_1, c_2, s_2) in the following way: [unreadable]. Alice sends (c_1, s_1, c_2, s_2) to Bob. Bob accepts the proof iff $c = c_1 + c_2$ and $t_1 = g^{s_1} h^{c_1}$ and $t_2 = g^{s_2} h^{c_2}$.

Question 1.1

a. Fill the unreadable part: how does Alice compute (r_i, c_i, s_i) so that the protocol is complete?
Hint. Recall that Alice knows the discrete logarithm of h_1 or of h_2. For the one she doesn't know, use the Schnorr simulator recalled earlier.

b. Sketch a proof that the (completed) protocol is zero-knowledge.

c. Sketch a proof that the (completed) protocol is knowledge-sound.

At this point, we have a protocol that allows Alice to prove that she knows the discrete logarithm of one of two group elements.
Now, assume that a trusted authority picks two generators f and g of $\mathbb{G}$. The generators f and g are public. For x and r in $\mathbb{Z}_p$, define:

$$C(x, r) = f^x g^r.$$

We say that Alice has *commited* to a value $x \in \mathbb{Z}_p$ when she has sent to Bob $C(x, r)$ for some $r \leftarrow \mathbb{Z}_p$, and Alice knows both x and r. Note that Bob doesn't know a priori which value x Alice has commited to.

Question 1.2. Build a protocol that allows Alice to commit to a value x, and prove in zero-knowledge that x is either zero or one. *Hint.* Use the previous protocol.
Now Alice wants to commit to an arbitrary value $x \in [0, 2^{k-1}]$, and prove (in zero-knowledge) that the committed value x lies in that interval. (The previous question was the special case $k = 1$.)
Question 1.3. Sketch a way to build the desired protocol, with a proof of size exponential in k.
We want to do better than that. Towards that end, define the binary decomposition of x: let $x_i \in \{0, 1\}$ for $i \in [0, k-1]$ such that $x = \sum_{0 \le i < k} x_i 2^i$.
Question 1.4. Propose a protocol that achieves the desired goal, where the size of the proof is $\mathcal{O}(k)$ (where the unit for the size of the proof is one memory word; and one memory word is assumed to be

large enough to contain an element of $\mathbb{G}$, or an element of $\mathbb{Z}_p$).
Hint. If $a = C(x, r)$ and $a' = C(x', r')$ are respectively commitments to x' and x', observe that $aa' \in \mathbb{G}$ is a commitment to xx'.
Question 1.5. Build a protocol that allows Alice to commit to an arbitrary value $x \in [a, b]$, and prove in zero-knowledge that x is in $[a, b]$ ($b > a > 0$), with a proof of size $\mathcal{O}(\log b)$.

Exercise 2. Probabilistic oblivious sorting. We set out to build an oblivious sorting algorithm. The server's memory is made up of memory blocks, indexed by $\mathbb{N}^*$. The client can only store two memory blocks, plus some auxiliary variables (counters etc). Initially, the server's memory contains some items $(x_1, \ldots, x_n)$. The item x_i is stored in block i.
Let $\pi : X \to [1, n]$ be an arbitrary bijection. Let $<_\pi$ be the order on X defined by $x_i <_\pi x_j$ iff $\pi(x_i) < \pi(x_j)$. In this exercise, we view sorting algorithms as algorithms that take π as input. At the outcome of the algorithm, the first n blocks of the server's memory should contain the items X sorted according to $<_\pi$. The client wishes to perform an oblivious sorting algorithm according to some $<_\pi$. (That is, the memory accesses of the client to the server should reveal no information about $<_\pi$; or more formally, for any two linear orders $<_\pi$ and $<_{\pi'}$, the sequence of accesses induced by the sorting algorithm for $<_\pi$ should be indistinguishable from the sequence of accesses induced by the algorithm for $<_{\pi'}$.)
A *random* sorting algorithm is a sorting algorithm that does not take a bijection π as input. Instead, the algorithm sorts according to $<_\pi$, where π is sampled uniformly at random among bijections $X \to [1, n]$. An oblivious random sorting algorithm is defined in the natural way as a random sorting algorithm that is also oblivious (leaks no information about π).
Question 2.1. Let $\mathbb{G}$ be a group. Let g be an arbitrary element of $\mathbb{G}$. Let u be sampled uniformly at random from $\mathbb{G}$. Show that gu is uniform over $\mathbb{G}$.
Question 2.2. Assume we know an oblivious *random* sorting algorithm that terminates in worst-case time T. Show that there exists an oblivious (non-random) sorting algorithm that terminates in time $T + \mathcal{O}(n \log n)$.
Note: please do so without using the fact that there exists a sorting network of size $\mathcal{O}(n \log n)$, mentioned in class, since that trivializes the question.

In the remainder, we focus on building an efficient random sorting algorithm. Assume for simplicity that n is a power of two. Let $z = \mathcal{O}(\log n)$ be an integer that divides n. Let $H : X \to \{0, 1\}^z$ be a hash function, which we will model as a uniformly random function. We assume there is no collision. Let $<^h$ be the order on X define by $x <^h y \Leftrightarrow H(x) \prec H(y)$, where $\prec$ denotes the lexicographic order. Let us partition the memory blocks on the server into *bins* of $6z$ consecutive blocks each. Move around the memory blocks so that each bin initially contains z items. Encrypt all memory blocks using IND-CCA encryption whenever they are read or written to by the client.
Assume we know an algorithm $\mathsf{Compaction}$ that takes as input two bins and a function $f : X \to \{0, 1\}$, and obliviously places all items x from the two input bins such that $f(x) = 0$ in the left bin, and all items x such that $f(x) = 1$ in the right bin. (Throughout this exercise, we will assume bins never overflow, *i.e.* there are enough memory blocks in each bin to store the relevant items.)
Question 2.3. Using $\mathcal{O}(n \log n)$ calls to compaction, obliviously sort the bins according to $<^h$. That is, at the outcome of the algorithm, all items in bin i are smaller for $<^h$ than the items in bin $i + 1$.
Question 2.4.

a. At the outcome of the previous question, we get n/z bins sorted according to $<^h$. However, the bins are of size $6z$, and contain only z items on average. To satisfy the earlier definition of a sorting algorithm, we must extract the real items from each bin, to place them in the first n memory blocks of the server. Is it possible for this extracting algorithm to reveal to the server how many (real) items there are per bin, while remaining oblivious?
   Hint. Show that the distribution of the number of items per bin at the outcome of the algorithm does *not* depend on $<^h$.

b. Propose an extraction algorithm in the sense of the previous question.

Question 2.5. Based on the previous questions, propose an oblivious sorting algorithm in time $\mathcal{O}(n \log^c n)$ for some constant c (which need not be computed).