Midterm 2 Solutions

- Do not open this quiz booklet until you are directed to do so. Read all the instructions first.
- This quiz contains 5 problems. You have 120 minutes to complete all problems. The points for each problem in the midterm roughly indicate how many minutes you should spend on the problem.
- This quiz booklet contains 15 pages, including this one and 3 sheets of scratch paper. **Please do not remove any pages from the booklet, including the scratch papers!**
- Write your solutions in the space provided. If you continue your solution on the back of any page, please make a note of it. If you run out of space, continue on a scratch page and make a note that you have done so.
- Do not waste time deriving facts that we have studied. Just cite results from class.
- When we ask you to "give an algorithm" in this quiz, describe your algorithm in plain English or if necessary, pseudocode, and provide a short argument for correctness and running time. You do not need to provide a diagram or example unless it helps make your explanation clearer.
- Do not spend too much time on any one problem.
- **Please write your name on every single page of this exam.**
- Good luck!

Problem	Title	Points	Parts	Grade	Initials
0	Name	1	1		
1	Short Answer Questions	15	2		
2	Graph Completeness	20	2		
3	Perfect Power Mod P	25	4		
4	Funky Sorting Network	25	4		
5	$\frac{3}{4}$-SAT	20	3		
Total		106			

Name: ______________________________

Circle your recitation:

F10	F11	F11	F11	F12	F12	F1	F1	F2	F3
Shalom	Shraman	Mayuri	Blake	Ethan	Isaac	Isaac	Akshat	Alex	Kai
R01	R02	R03	R04	R05	R06	R07	R08	R09	R10

Quiz 1-1: [15 points] Short answer questions

For each of the following questions you must provide an answer along with a **short proof** of your answer. You can reference and use any statement from the class without a proof.

(a) [5 Points] In a skip list with n items, is it possible that the "Insert" operation takes $\Omega(\sqrt{n})$ time? Briefly, explain your answer.

Solution: Yes. A skip list could be of any height, depending on its random choices, and in particular have all nodes at 2 levels, with $\sqrt{n}$ reaching the second level and having the lower level items spaced at $\sqrt{n}$ distances between them.

(b) [10 points] Sam is organizing a winter workshop for algorithms. There are n potential speakers. If Sam invites speaker i, then he has to pay the travel cost s_i to her. There are m potential participants who want to attend the workshop. Participant j will attend the workshop if all her favorite speakers, denoted by set L_j, are invited, and is willing to pay p_j to Sam for attending the workshop. Sam wants to find a set of speakers to invite so that his overall profit is maximized.

Write a maximization ILP (Integer Linear Program) to solve this problem.

Solution: Let x_i be a variable that indicates whether speaker i is invited or not. i.e. $x_i = 1$ if speaker i is invited and 0 otherwise. Let y_j be a variable that indicates whether participant j attends. i.e. $y_j = 1$ if participant j attends the workshop, and 0 otherwise. Then our ILP is as follows,

$$\begin{array}{lll}
\max & \sum_{j=1}^{m} p_j y_j - \sum_{i=1}^{n} s_i x_i & \\
\text{subject to} & \sum_{i \in L_j} x_i \geq |L_j| y_j & \forall\, 1 \leq j \leq m \\
 & x_i, y_j \in \{0, 1\} & \forall\, i, j
\end{array}$$

Quiz 1-2: [20 points] Graph Completeness

Alice is given a very large undirected graph $G = (V, E)$ with $n = |V|$. We say that a graph is complete if it contains an edge between all $\binom{n}{2}$ pairs of vertices. We say that a graph is ϵ-far from complete if it is missing at least $\epsilon\binom{n}{2}$ edges in order to make it complete.

Alice is given query access to G. Namely, for any u and v vertices in V, Alice can issue queries (u, v) to find out whether $(u, v) \in E$ or $(u, v) \notin E$.

(a) [14 points] Help Alice design a Property Testing algorithm with parameter ϵ which makes $O(1/\epsilon)$ queries to G and satisfies the following:

- If G is complete, the algorithm outputs ACCEPT with probability 1.
- If G is ϵ-far from being complete, the algorithm outputs REJECT with probability at least 2/3.

2**Note:** The following inequality may be helpful: $(1 + x) \leq e^x$ for any x. **Solution: Version 2**

```
COMPLETENESS-TESTER($G, \epsilon$)
1  $m \leftarrow 2\epsilon^{-1}$
2  for $i = 1, \dots m$
3        do Pick random $u, v \in V$ uniformly at random.
4           if $(u, v) \notin E$
5              then REJECT
6  ACCEPT
```

It is not hard to see that if G is complete, we always accept.

Now suppose G is ϵ-far from being complete. That means the probability of picking a random u, v such that $(u, v) \notin E$ is at least ϵ. Therefore, the probability that all (u, v) are in E is at most $(1 - \epsilon)^m \leq e^{-\epsilon m} \leq 1/3$ for $m = 2/\epsilon$.

Therefore, with probability at least $2/3$, the algorithm will reject.

(b) [6 points] Modify your algorithm from part (a) to reduce the error probability $1/3$ to less than $1/\sqrt{n}$ while keeping the number of queries sublinear.

Note: You can do this part without solving part (a).

Solution: From part (a), we have an algorithm that has an error probability of at most $1/3$. We can simply repeat the algorithm $O(\log_3 \sqrt{n})$ to get the error probability to be at most $(1/3)^{\log_3 \sqrt{n}} = 1/\sqrt{n}$. Which means the algorithm is correct with probability at least $1 - 1/\sqrt{n}$.

This is still sublinear since the total number of queries is now $(2/\epsilon) \log_3 \sqrt{n} = O(\log n)$.

Quiz 1-3: [25 points] Perfect Power Mod P

In this problem we are given that b is a perfect power of a (mod p), and we want to determine that power. Formally, given the integers $a, b > 1$ and the fact that $b = a^c$ (mod p) for some integer $c \in \{0, 1, 2, \ldots B\}$, we want to find c.

In this problem, assume that the only operations that we can perform are additions, multiplications, finding x (mod p) given x, computing the inverse x^{-1} (mod p) given x, and comparisons, which can all be done in $O(1)$ time.

In particular, we can **not** take k-th roots, exponents, or logs in $O(1)$ time.

Note: The original version of the problem had a typo where a and b were switched in the problem statement.

(a) [5 points] Devise an algorithm that finds c in $O(B)$ multiplications.

Solution: Compute $1, a, a^2, \ldots, a^B$ (mod p) and compare each one to b. One of these terms a^n must equal b. This algorithm takes $O(B)$ multiplications because we can just multiply a^{n-1} by a to get a^n each time.

(b) [7 points] Let $B' = \lceil \sqrt{B} \rceil$. Show that there exist integers $c_1, c_2 \in \{0, 1, 2, \ldots, B'\}$ such that

$$ba^{-c_2} = (a^{B'})^{c_1} \mod p$$

Solution: The equation rearranges to $b = a^{B'c_1 + c_2}$. Thus, we just have to show that $B'c_1 + c_2 = c$ has solutions for c_1, c_2. Note that $B'c_1 + c_2$ can represent any number between 0 and $B'^2 + B' > B'^2 \geq B$. Since c is between 0 and B, it can be represented in the desired form.

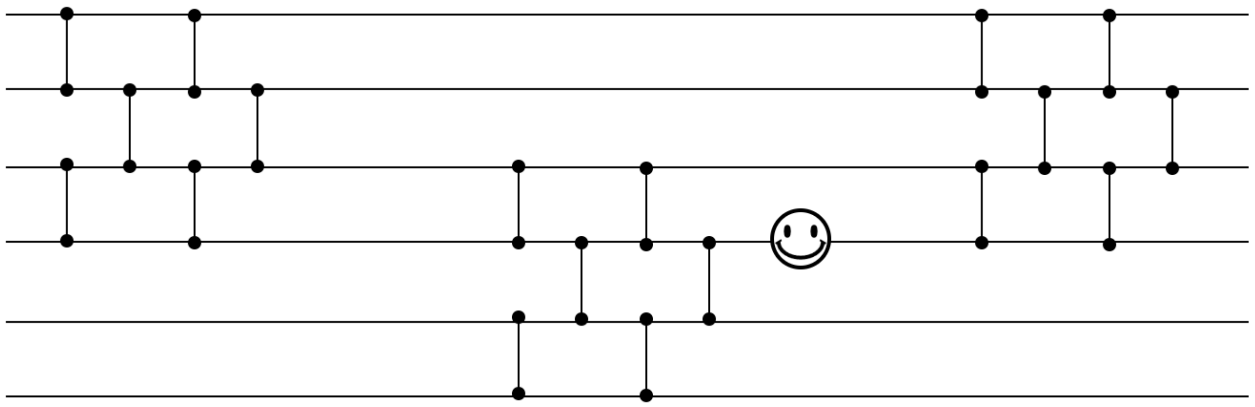
(c) [3 points] Let us construct a sorted list L that contains the elements $b, ba^{-1}, ba^{-2}, \ldots, ba^{-B'}$ (mod p). How many multiplications and comparisons, respectively, does it take to construct this sorted list?

Solution: It takes $O(B')$ multiplications to compute all of the elements in L, and it takes $O(B' \log B')$ comparisons to sort the elements of L.

(d) [10 points] Finally, using the equation from (b) and the sorted list L from (c), devise an algorithm that solves the original problem of finding c such that $b = a^c$ (mod p) in $O(B')$ multiplications and $O(B' \log B')$ comparisons.

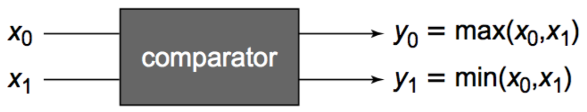
Solution: Construct the sorted list L, as before, which takes $O(B')$ multplications and $O(B' \log B')$. Then, compute $x = a^{B'}$ (mod p) in $O(B')$ multiplications. Then, iteratively compute x^{c_1} (mod p) for $c_1 \in \{0, 1, 2, \ldots, B'\}$. For each value of c_1, we want to check if x^{c_1} (mod p) is in the list L. If we can find x^{c_1} (mod p) in the list L, then we will have found a solution (c_1, c_2) to the equation from (b), which is equivalent to finding c. In order to check if x^{c_1} (mod p) is in L, we simply need to binary search through L to look for x^{c_1} (mod p), which takes $O(\log B')$ comparisons. Because there are B' possible values of c_1, the total number of operations needed is $O(B')$ multiplications and $O(B' \log B')$ comparisons.

Quiz 1-4: [25 points] Funky Sorting Network



Shown above is a sorting network with 6 input/output wires. The vertical lines in the network represent comparators.

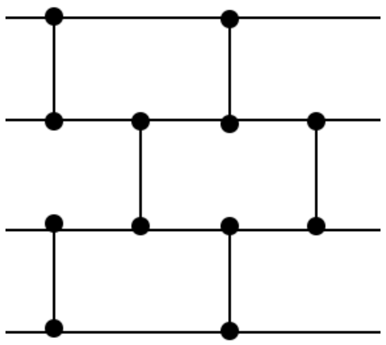
As a reminder, comparators take in two inputs x_0 and x_1 and output the maximum of those two inputs on the top wire and the minimum of those two inputs on the bottom wire, as shown below.



In this problem, we will analyze the above sorting network. The smiley face :) in the network doesn't change the network in any way, it's just there because it is happy.

(a) **[5 Points]** Suppose the inputs to the 6 wires, from top to bottom, are 1, 3, 6, 2, 5, 4. What are the final 6 outputs? What number passes through the location of the smiley face?

Solution: The final output, from top to bottom, is 6,5,4,3,2,1. The number at the smiley face is 4.



(b) **[5 Points]** Let's zoom in a bit on the comparators in the upper left section of the network (shown in figure above). Prove that this section of the network is itself a sorting network.

Solution: This can be analyzed by showing that the network sorts all 0-1 sequences. One way to analyze this is to do casework on the number of 0's and 1's on the 4 input wires. If the sequence has all 1's or 0's, the result is obvious. If the sequence has a single 1 or a single 0, it's easy to see that that single 1 or 0 will end up at the top or the bottom, respectively. Finally, you can enumerate the 6 remaining cases where there are two 1's and two 0's. Alternatively, you can notice that is a special case of the OddEven sorting network.

(c) **[10 Points]** Prove that the entire network is a sorting network.

Note: You may assume the result described in part (b), even if you haven't solved it.

Solution: The first triangle sorts the upper 4 wires, and the next triangle sorts the bottom 4 wires. The last triangle re-sorts the upper 4. After the first sort, the 2 smallest elements will be in the bottom 4 wires because they can not end up in the top 2 wires. After the second sort, the 2 smallest elements will be properly sorted on the bottom 2 wires. Finally, the last sort sorts the upper 4 wires, which contain the 4 largest elements.

(d) [5 Points] Replace all the comparators with balancers in the funky sorting network. Provide a counterexample to show that the resulting balancing network is not a counting network.

Solution: Put three tokens through the bottom wire. The third token will end up at the bottommost wire, which should not happen in a counting network until at least 6 tokens have been inputted to the network.

Quiz 1-5: [20 points] $\frac{3}{4}$-SAT

In the $\frac{3}{4}$-SAT problem, you are given n boolean variables x_i which can be set to TRUE or FALSE. You are also given m clauses C_j composed of a logical OR of three "literals," i.e. a variable x_i or its negation $\neg x_i$. A clause is "satisfied" if it evaluates to TRUE. A "$\frac{3}{4}$-satisfying assignment" is an assignment of values to variables such that at least $\frac{3m}{4}$ clauses are satisfied.

Below is an example of a clause and a table of assignments that satisfy the clause.

$$C_j = (x_a \vee \neg x_b \vee x_c)$$

x_a	x_b	x_c
1	0	0
1	0	1
1	1	0
1	1	1
0	0	0
0	0	1
0	1	1

(a) **[5 points]** Design an $O(n)$ time Monte-Carlo algorithm that satisfies $7m/8$ clauses in expectation. You may assume clauses do not have duplicate variables (a literal or its negation).

Hint: Try a random assignment.

Solution: Our algorithm is to independently assign each variable x_i to be 0 or 1 uniformly at random. Since there are three different variables in each clause, this gives it a 1/8 probability of being in any one of the 8 possible configurations. And since only 1 out of the 8 configurations could set the clause to be false, each clause is satisfied with probability $7/8$.

Hence, by linearity of expectation, $7m/8$ clauses will be satisfied in expectation.

(b) [10 points] Show that your algorithm satisfies at least $3m/4$ clauses with probability at least 1/2.

Hint: Use Markov Bound.

Solution: Let U be the number of unsatisfied clauses. From part (a), we know that the expectation of U is $\mathbb{E}[U] = m/8$. Since U is a nonnegative random variable, we can apply Markov Bound to upper bound the probability that $U > \frac{m}{4}$.

$$Pr[U > \frac{m}{4}] \leq \frac{\mathbb{E}[U]}{m/4} = \frac{m/8}{m/4} = \frac{1}{2}$$

Therefore, with probability at least 1/2, the algorithm satisfies at least $3m/4$ clauses.

(c) [5 points] Use parts (a) and (b) to design a Monte Carlo randomized algorithm that runs in $O((m+n)\log n)$ time and outputs a correct $\frac{3}{4}$-satisfying assignment with probability at least $1 - \frac{1}{n^2}$.

Note: You may assume the results from parts (a) and (b).

Solution: Given an assignment, we can count in $O(m)$ time, how many clauses are satisfied with that assignment. Hence, our new algorithm is to repeat our algorithm from part (a) (randomly assign values to each variable) a maximum of $2\log n$ times, until at least $3m/4$ clauses are satisfied.

First, since our algorithm terminates after $2 \log n$ iterations, and each iteration takes $O(n)$ time to randomly assign variables and $O(m)$ time to count the number of satisfied clauses, it runs in deterministic $O((n + m) \log n)$ time.

The probability that none of the iterations satisfy at least $3m/4$ clauses is at most $(1/2)^{2 \log n} = \frac{1}{n^2}$. Therefore, with probability at least $1 - \frac{1}{n^2}$, at least one of the iterations will be a $3/4$ satisfying assignment.

Scratch page

Scratch page

Scratch page