1. **(10 points) List Prefix Sums**: List Prefix Sums is the task of determining the sum of all the elements before each element in a list. Let us consider the following simple variation.

- Select each element from the list randomly and independently with probability $1/\log n$ and add it to a set $S$. Add the head of the list to this set, and mark all these elements in the list.
- Start from each element $s \in S$, and in parallel traverse the lists until you find the next element in $S$ (by detecting the mark) or the end of the list. For $s \in S$, call this element found in this way $\text{next}(s)$. While traversing, calculate the sum from $s$ to $\text{next}(s)$ (inclusive of $s$ but exclusive of $\text{next}(s)$), and call this $\text{sum}(s)$.
- Create a list by linking each $s \in S$ to $\text{next}(s)$ and with each node having weight $\text{sum}(s)$.
- Compute the List Prefix Sums on this list using pointer jumping. Call the result $\text{prefixsum}(s)$.
- Go back to the original list, and again traverse from each $s$ to $\text{next}(s)$ starting with the value $\text{prefixsum}(s)$ and adding the value at each node to a running sum and writing this into the node. Now all elements in the list should have the correct prefix sum.

Analyze the work and depth of this algorithm. These should both be given with high probability bounds (one idea might be to use Chernoff bounds for getting these).

**Solution** With high probability $|S| = O(n/\log n)$. We will show that the size of $S$ is $O(n/\log n)$ with high probability using a Chernoff bound. Associate indicator variable $X_i$ with the $i$-th element of the initial list to indicate whether it has been selected for inclusion in $S$. Since we sample each element with probability $1/\log n$, we see that $X_i$’s are Bernoulli distributed where

$$X_i = \begin{cases} 1 & \text{with probability } 1/\log n, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, the $X_i$ are independent. Since our $X_i$’s are indicators, then their probability equals their expectation.$^1$ By linearity of expectations,

$$\mathbb{E}[X] = \mathbb{E} \left[ \sum_{i=1}^{n} X_i \right] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{1}{\log n} = \frac{n}{\log n}.$$  

---

$^1$This follows trivially, since $\mathbb{E}[X_i] = 1 \cdot \Pr(X_i = 1) + 0 \cdot \Pr(X_i = 0) = \Pr(X_i = 1)$. 

---
That is, in expectation $|S| = n/\log n$. Recall Chernoff’s Bound, for $\delta \in (0, 1)$,

$$\Pr (X > (1 + \delta)\mu) < e^{-\delta^2 \mu/3}.$$ 

We evaluate for $\delta = 1/2$ and $\mu = n/\log n$,

$$\Pr(X > 3n/(2 \log n)) < e^{\frac{n}{12 \log n}}$$
$$< e^{-\log n} = \frac{1}{n}$$

We have used the fact that for sufficiently large $n$, $\frac{n}{12 \log n} > \log n$. Note that the analysis can be made more tight, but this probability bound will suffice to say that with high probability, $|S| = O(n/\log n)$.

**Total Work**  We see that steps a, b, c and e all require $O(n)$ work in the worst case. In step d, we use pointer jumping to perform prefix sums of our linked list. This algorithm, on a list of size $n$, requires $O(n \log n)$ work and $O(\log n)$ depth. Therefore, step d requires $|S| \log |S|$ work. With high probability:

$$|S| \log |S| \leq \frac{3n}{2 \log n} \log \left( \frac{3n}{2 \log n} \right) = O(n).$$

Total work, then, is $O(n)$.

**Total Depth**  Now we compute the total depth. Steps a and c have depth $O(1)$ because we may do all nodes in parallel. As we stated before, computing prefix sums using pointer jumping on linked lists takes depth $O(\log |S|) = O(\log(n/\log n))$. In steps b and e we need to traverse all elements in the original list between elements in $S$, so the depth of these steps will be the maximum length between elements in $S$. We will show with high probability this maximum length will be less than $4 \log^2 n$. If this claim holds, then total depth for the algorithm will be $O(\log^2 n)$ with high probability.

The maximum length between elements in $S$ is less than $O(\log^2 n)$ with high probability. Consider dividing the original list into chunks of size $2 \log^2 n$ (so there will be $n/(2 \log^2 n)$ chunks). For every element in the original list, the probability it is not chosen to $S$ is $(1 - 1/\log n)$, independently. The probability that all of the elements in a particular chunk, $c_i$, of the original list are not in $S$ is given by:

$$\Pr(e \notin S, \forall e \in c_i) = (1 - 1/\log n)^{2 \log^2 n}$$
$$= ((1 - 1/\log n)^{\log n})^{2 \log n}$$
$$\leq (1/e)^{2 \log n} < \frac{1}{n^2}$$

\[\text{\textsuperscript{2}}\text{Please see the Wikipedia page on pointer jumping and/or http://wwwmayr.informatik.tu-muenchen.de/lehre/2013WS/pa/split/sub-Prefix-Sum-single.pdf for a discussion on pointer jumping for prefix sums.}\]
By union bound, we know the probability that all the chunks have at least one element in \( S \) is \( \leq \frac{n}{(2 \log^2 n) \cdot 1/n^2} \leq 1/n \). So with high probability the maximum length between elements in \( S \) is bounded by \( 4 \log^2 n \) (the maximum interval between points in consecutive chunks). Thus, the depth of steps b and e is \( O(\log^2 n) \) with high probability, and that is the total depth of the algorithm as well.

2. (6 points) **Random Mate on Graphs** In class we described a random-mate technique for determining graph connectivity. Each node flips a coin, and every edge from a head to a tail will attempt to hook the tail into the head (i.e., relabel the tail with the head pointer). Given a \( d \)-regular graph on \( n \) vertices, i.e., a graph in which every vertex has degree \( d \), what is the expected number of vertices after one contraction step?

**Solution** A vertex is relabeled if and only if its corresponding coin toss is Tails (probability \( 1/2 \)), and at least one of its \( d \) neighboring vertices gets a Heads (probability \( 1 - \frac{1}{2^d} \)). Since each vertex flips a coin independently, we see that a vertex is relabeled with probability:

\[
\Pr(\text{node } i \text{ relabeled}) = \Pr(\text{node } i \text{ is a follower and at least one neighbor } v \text{ of node } i \text{ is a leader})
= \frac{1}{2} \left( 1 - \frac{1}{2^d} \right).
\]

Let \( X_i \) denote the indicator variable which takes on value 1 if vertex \( i \) relabeled and 0 otherwise. Then the total number of vertices contracted is given by \( X_n = \sum_{i=1}^{n} X_i \).

Taking the expectation,

\[
\mathbb{E}[X] = \mathbb{E} \left[ \sum_{i=1}^{n} X_i \right] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \left( \frac{1}{2} \left( 1 - \frac{1}{2^d} \right) \right) = \frac{n}{2} \left( 1 - \frac{1}{2^d} \right).
\]

Hence, the expected number of vertices remaining after one contraction step is given by

\[
n - \frac{n}{2} \left( 1 - \frac{1}{2^d} \right) = \frac{n}{2} + \frac{n}{2^{d+1}} = \frac{n}{2} \left( 1 + \frac{1}{2^d} \right).
\]

3. (8 points) **Finding Parent Nodes In Tree:** Let \( T = (V, E) \) be a rooted tree with \( V = \{1, 2, ..., n\} \). Each node in \( T \) has a specific ordering of its children. In other words, we have a specific first, second, ..., \( n \)-th children of a node. We can also define next sibling of \( i \)-th child of \( v \) as the \( (i+1) \)-th child of \( v \). You are given the value \( n \) along with the tree \( T \) as two arrays \( fc \) (first child) and \( ns \) (next sibling) of size \( n \) each. For each \( u \in V \), the element \( fc(u) \) is the first child of \( u \), and 0 if \( u \) does not have children. Likewise, for each \( u \in V \), the element \( ns(u) \) is the next sibling of \( u \), and 0 if \( u \) does not have a next sibling. The following figure gives an example for \( n = 12 \) and root node = 7, with left-to-right child ordering. We want to compute an array \( p \) of size \( n \) such that \( p(u) \) stores the parent of \( u \) for all \( u \in V \) (we will have \( p(r) = 0 \) for the root node \( r \)). Propose an \( O(\log n) \) depth algorithm for an ‘exclusive’ read exclusive write
PRAM model assuming exclusive reads have the same depth as concurrent reads for simplicity. Also specify what is the work done by your algorithm. You can use extra space, if required.

Solution  The algorithm can proceed in two steps:

- First, each parent copies its number to its first child in parent array.
- Then, the first child copies the parent number to its siblings in a branched fashion.

For example, assume node $v$ has $k$ children $u_1, u_2, ..., u_k$. The first step will set $p(u_1) = v$. Then $u_1$ copies $v$ to $p(u_2)$, followed by $u_1, u_2$ copying $v$ to $p(u_3)$ and $p(u_4)$ respectively, followed by $u_1, u_2, u_3, u_4$ copying $v$ to $p(u_5), p(u_6), p(u_7), p(u_8)$, and so on. It is clear that all siblings of $u_1$ eventually get the parent number $v$ in $O(\log k)$ parallel steps. Since $k \leq n-1$, the depth of the algorithm will be bounded by $O(\log n)$. Here’s the pseudocode for same:

```
1 Parallel For u in 1,2,...,n:
   2   Initialize p(u) = 0.
   3   If fc(u) != 0, set p(fc(u)) = u.
   4   Make a copy S(u) = ns(u).
   5   While S(u) != 0, repeat: /* Sequential */
   6      If p(u) != 0, copy p(u) to p(S(u)).
   7      Set S(u) = S(S(u)).
```

The work done by this algorithm is $O(n \log n)$ (since each node has $\leq n-1$ children). Note that if the busy wait based upon the condition $p(u) \neq 0$ can be avoided, the work done can be derived to be $O(n)$.

4. (8 points) Parallel Gaussian Elimination: Let $A$ be an invertible $n \times n$ matrix with real-valued entries and we want to compute $B = A^{-1}$. One popular algorithm for this computation involves Gaussian elimination wherein we start with $B = I_n$ and apply elementary row operations (exchanging two rows, multiplying a row by a non-zero value, subtracting a multiple of a row from another) to convert $A$ to the $n \times n$ identity matrix $I_n$. As we do this, we apply the same sequence of elementary row operations to $B$ such that when $A$ is reduced to $I_n$, $B$ changes from $I_n$ to $A^{-1}$. This is how the row-reduction algorithm looks like for doing operations on only $A$:

```
1 For i = 1,2,...,n, do:
   2      Find one k in {i, i+1, ..., n} such that A[k, i] != 0
```
- If $k \neq i$, swap row $A[i, :]$ with row $A[k, :]$
- Divide row $A[i, :]$ by $A[i, i]$
- For $k$ in $\{1,2,\ldots,n\}$ with $k \neq i$ and $A[k, i] \neq 0$:
  Subtract $A[k, i]$ times row $A[i, :]$ from row $A[k, :]$

(a) Design an $O(n \log n)$ depth parallel algorithm and write its pseudocode using the above matrix-inversion method. You can use extra space for your algorithm. Remember to mention the steps for initializing and computing entries in $B$ as well when you use the above pseudocode for scaffolding.

(b) If it is given that $A$ is a lower triangular invertible $n \times n$ matrix with real-valued entries, explain how we can develop a $O(\log^2 n)$ depth algorithm to invert $A$. Also deduce the work done by your algorithm. Can we gain anything on work complexity by using Strassen’s algorithm? (Hint: Break each of $A$ and $A^{-1}$ into four $\frac{n}{2} \times \frac{n}{2}$ blocks, and use the fact that $A^{-1}$ is again lower triangular.)

Solution

(a) Here is a parallel implementation:

```markdown
// Initialization */
Parallel For $i$, $j$ in $\{1,2,\ldots,n\}$
  Set $B[i, j] = 1$ if $i = j$ else $B[i, j] = 0$
/* Row Reduction Loop */
For $i = 1,2,\ldots,n$, do: /* Sequential */
  (a) Find one $k$ in $\{i, i+1, \ldots, n\}$ such that $A[k, i] \neq 0$
  (b) If $k \neq i$, Parallel For $j$ in $\{1, 2, \ldots, n\}$:
      Swap row $A[i, j]$ with row $A[k, j]$
      Swap row $B[i, j]$ with row $B[k, j]$
  (c) Divide row $A[i, j]$ by $A[i, i]$ for each $j$ in parallel
      Divide row $B[i, j]$ by $A[i, i]$ for each $j$ in parallel
  (d) Parallel For $k$, $j$ in $\{1,2,\ldots,n\}$
      If $k \neq i$ and $A[k, i] \neq 0$:
        Subtract $A[k, i]$ times $B[i, j]$ from $B[k, j]$
```

Depth: Initialization takes $O(1)$ depth if we give each matrix entry to different processor. In row reduction loop, step (a) can be implemented in parallel in $O(\log n)$ depth. For that, use an array $U[i-1, \ldots, n]$ initialized with (1) $U[i-1] = 0$ and (2) $U[k] = 0$ if $A[k, i] = 0$, otherwise $U[k] = 1$. Compute the all prefix sum of $U$ in an array $V$ and after that, identify $k$ in parallel as the index $k \in \{i, i+1, \ldots, n\}$ such that $V[k-1] = 0$ and $V[k] = 1$. All these sub-steps of step (a) can be done in $O(\log n)$ depth and steps (b)-(d) can be done in $O(1)$ depth. Since the outer row-reduction loop is sequential, the total depth is $O(n \log n)$.

Work: Initialization requires $O(n^2)$ effort. For each iteration in the row-reduction loop, step (a) can be implemented so as to perform only $O(n)$ work. Steps (b) and (c) do $O(n)$ work but step (d) does $O(n^2)$ work. Thus, the overall row-reduction process takes $O(n^3)$ work when counted for each row.
5. (10 points) Stochastic Gradient Descent

(a) In class we stated that gradient descent on \( L \)-smooth functions is guaranteed to decrease the function value at each iteration. Stochastic gradient descent, on the other hand, does not have the same guarantee. Provide an example where stochastic gradient descent does not produce a descent step. Specifically, find a function \( f(x) = \sum_{i=1}^{m} f_i(x) \), and an iterate \( x_0 \) such that for all step sizes, there exist \( i \) such that \( f(x_1) > f(x_0) \) (where \( x_1 := x_0 - \alpha \nabla f_i(x) \)).

Solution Consider \( f(x) = \frac{1}{2}(f_1(x) + f_2(x)) \) where \( f_1(x) = \frac{1}{2}(x-2)^2 \) and \( f_2(x) = \frac{1}{2}(x+1)^2 \). Suppose \( x_0 = 0 \) and we sample \( f_2 \) first, then \( x_1 = x_0 - \gamma (x_0 + 1) = -\gamma \). So regardless of how we choose \( \gamma > 0 \), \( f(x_1) > f(x_0) \), so this would not be a descent step.

(b) This exercise will guide you through the convergence proof of SGD. As a reminder, we are proving that if there exists a constant \( G \) such that \( \mathbb{E}[\|\nabla f_i(x)\|^2] \leq G^2 \) and \( f(x) \) is \( \mu \)-strongly convex. Then, with step-sizes \( \gamma_k = \frac{1}{\mu k} \), we have

\[
\mathbb{E}[\|x_k - x^*\|^2] \leq \frac{\max\{\|x_1 - x^*\|^2, \frac{G^2}{\mu^2}\}}{k}.
\]

- Using strong convexity, prove that

\[
\langle \nabla f(x_k), x_k - x^* \rangle = \langle \nabla f(x_k), x_k - x^* \rangle \geq \mu \|x_k - x^*\|^2
\]

Solution Because \( f \) is \( \mu \)-strongly convex, then we have that

\[
f(x^*) - f(x_k) \geq \langle \nabla f(x_k), x^* - x_k \rangle + \frac{\mu}{2} \|x_k - x^*\|^2
\]

\[
f(x_k) - f(x^*) \geq \langle \nabla f(x^*), x_k - x^* \rangle + \frac{\mu}{2} \|x_k - x^*\|^2
\]
Combining the inequalities:
\[ \langle \nabla f(x_k) - \nabla f(x_*), x_k - x_* \rangle = \langle \nabla f(x_k), x_k - x_* \rangle \geq \mu \| x_k - x_* \|^2 \]

- Apply the previous step, to express \( E[\|x_{k+1} - x_*\|^2] \) in terms of \( E[\|x_k - x_*\|^2] \), \( \gamma_k \), \( G \), and \( \mu \).

**Solution**

Expanding the norm,
\[
E(\|x_{k+1} - x_*\|^2) = E(\|x_k - \gamma_k g_k - x_*\|^2)
\]
\[
= E(\|x_k - x_*\|^2) - 2\gamma_k E(\langle g_k, x_k - x_* \rangle) + \gamma_k^2 E(\|g_k\|^2)
\]
\[
\leq E(\|x_k - x_*\|^2) - 2\gamma_k E(\langle \nabla f(x_k), x_k - x_* \rangle) + \gamma_k^2 G^2
\]
\[
\leq E(\|x_k - x_*\|^2) - 2\gamma_k \mu E(\|x_k - x_*\|^2) + \gamma_k^2 G^2
\]

- Prove the convergence of SGD using induction. It is clear that the base case, which states
\[ \|x_1 - x_*\|^2 \leq \max\{\|x_1 - x_*\|^2, \frac{G^2}{\mu^2}\} \]
is true. Now assume that statement holds until iteration \( k \), we need to show that it holds for \( k + 1 \). From the previous bullet point we have that
\[
E(\|x_{k+1} - x_*\|^2) \leq \left(1 - \frac{2}{k}\right) E(\|x_k - x_*\|^2) + \frac{1}{\mu^2 k^2} G^2
\]
\[
\leq \left(1 - \frac{2}{k}\right) \frac{\max\{\|x_k - x_*\|^2, \frac{G^2}{\mu^2}\}}{k} + \frac{\max\{\|x_k - x_*\|^2, \frac{G^2}{\mu^2}\}}{k^2}
\]
\[
\leq \frac{1}{k} \frac{\max\{\|x_k - x_*\|^2, \frac{G^2}{\mu^2}\}}{k + 1}
\]

6. (8 points) **HOGWILD!** Recall that in HOGWILD!, the objective function we want to minimize is:
\[ f(x) = \sum_{e \in E} f_e(x_e) \]

where we define the hyperedge \( e \) to be the subset of variables that \( f_e \) depends on. Figure 1 depicts such a graph. Then, if we denote the average degree of the conflict graph as \( \Delta_C \), convergence is still guaranteed if the core delay is less than \( \tau \leq \frac{n}{2\Delta_C} \) (i.e., no more than \( \tau \) samples are being processed while a core is processing one).
Graph Cuts: In graph cuts problems, we are given a sparse matrix $W$ which indexes similarity between node. We want to match each node to a list of $D$ classes i.e., we want assign a vector $x_i \in \{v \in \mathbb{R}^D | \sum_{j=1}^{D} v_j = 1, v_j \geq 0\}$ that solve the following optimization problem.

$$
\begin{align*}
\text{minimize} & \sum_{(u,v) \in E} w_{uv} \|x_u - x_v\|_1 \\
\text{subject to} & x_u \in \{v \in \mathbb{R}^D | \sum_{j=1}^{D} v_j = 1, v_j \geq 0\}.
\end{align*}
$$

Prove that in this case

$$\frac{\overline{\Delta_C}}{n} = O(\text{Avg. degree of graph in graph cut problem})$$

Solution Since a function $f_{u,v}$ conflicts with all other functions involving $u$ and $v$, the total number of conflicts is

$$
\sum_{(u,v) \in E} \text{deg}(u) + \text{deg}(v) - 2 = -2|E| + \sum_{(u,v) \in E} \text{deg}(u) + \sum_{(u,v) \in E} \text{deg}(v)
$$

$$
= -2|E| + \sum_{u \in V} \sum_{v \in N(u)} \text{deg}(u)
$$

$$
= -2|E| + \sum_{u \in V} \text{deg}^2(u)
$$

$$
< \sum_{u \in V} \text{deg}^2(u)
$$

$$
\leq \left( \sum_{u \in V} \text{deg}(u) \right)^2
$$

$$
= (2m)^2
$$

So $\overline{\Delta_C} \leq 4m$ and so $\frac{\overline{\Delta_C}}{n} = O(\text{Avg. deg.}).$
Programming Warm Up (30 points)

For the next few problems, we expect that you can learn SQL on your own and answer the below questions. This will act as a warm up for second half of the course. Some good guides for learning SQL:

(a) https://www.w3schools.com/sql/sql_intro.asp
(b) https://www.youtube.com/watch?v=9Pzj7Aj25lw

7. (4 points) In this set of problems, we review how to select data from relational databases.

(a.) Write a SQL statement to find the total purchase amount of all orders. Sample table: orders.

<table>
<thead>
<tr>
<th>ord_no</th>
<th>purch_amt</th>
<th>ord_date</th>
<th>customer_id</th>
<th>salesman_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>70001</td>
<td>150.5</td>
<td>2012-10-05</td>
<td>3005</td>
<td>5002</td>
</tr>
<tr>
<td>70009</td>
<td>270.65</td>
<td>2012-09-10</td>
<td>3001</td>
<td>5005</td>
</tr>
<tr>
<td>70002</td>
<td>65.26</td>
<td>2012-10-05</td>
<td>3002</td>
<td>5001</td>
</tr>
<tr>
<td>70004</td>
<td>110.5</td>
<td>2012-08-17</td>
<td>3009</td>
<td>5003</td>
</tr>
<tr>
<td>70007</td>
<td>948.5</td>
<td>2012-09-10</td>
<td>3005</td>
<td>5002</td>
</tr>
</tbody>
</table>

Solution

```
SELECT SUM (purch_amt)
FROM orders;
```

(b.) Write a SQL statement which selects the highest grade for each of the cities of the customers. Sample table: customer.

<table>
<thead>
<tr>
<th>customer_id</th>
<th>cust_name</th>
<th>city</th>
<th>grade</th>
<th>salesman_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>3002</td>
<td>Nick Rimando</td>
<td>New York</td>
<td>100</td>
<td>5001</td>
</tr>
<tr>
<td>3005</td>
<td>Graham Zusi</td>
<td>California</td>
<td>200</td>
<td>5002</td>
</tr>
<tr>
<td>3001</td>
<td>Brad Guzan</td>
<td>London</td>
<td></td>
<td>5005</td>
</tr>
<tr>
<td>3004</td>
<td>Fabian Johns</td>
<td>Paris</td>
<td>300</td>
<td>5006</td>
</tr>
<tr>
<td>3007</td>
<td>Brad Davis</td>
<td>New York</td>
<td>200</td>
<td>5001</td>
</tr>
</tbody>
</table>

Solution

```
SELECT MAX (grade)
FROM customer
GROUP BY city;
```

(c.) Write a SQL statement to find the highest purchase amount on a date “2012-08-17” for each salesman with their ID. Sample table: orders, used in (a).
SELECT salesman_id, MAX(purch_amt)
FROM orders
WHERE ord_date = "2012-08-17"
GROUP BY salesman_id;

8. (8 points) In this problem, we review how to merge two tables together.

(a.) Write a SQL statement to know which salesman are working for which customer. Use the sample table customer, used in previous problem, and also salesman.

<table>
<thead>
<tr>
<th>salesman_id</th>
<th>name</th>
<th>city</th>
<th>commission</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>James Hoog</td>
<td>New York</td>
<td>0.15</td>
</tr>
<tr>
<td>5002</td>
<td>Nail Knite</td>
<td>Paris</td>
<td>0.13</td>
</tr>
<tr>
<td>5005</td>
<td>Pit Alex</td>
<td>London</td>
<td>0.11</td>
</tr>
<tr>
<td>5006</td>
<td>Mc Lyon</td>
<td>Paris</td>
<td>0.14</td>
</tr>
<tr>
<td>5003</td>
<td>Lauson Hen</td>
<td></td>
<td>0.12</td>
</tr>
</tbody>
</table>

Solution

```
SELECT c.cust_name, c.city, s.name, s.commission
FROM customer c
INNER JOIN salesman s
ON c.salesman_id = s.salesman_id;
```  

(b.) Write a query to display all salesmen and customers located in London.

Solution

```
SELECT salesman_id, name
FROM salesman
WHERE city = 'London'
UNION
SELECT customer_id, customer_name
FROM customer
WHERE city = 'London';
```  

(c.) Write a SQL statement to make a cartesian product between salesman and customer i.e. each salesman will appear for all customer and vice versa for those salesmen who belongs to a city and the customers who must have a grade.

Solution

```
SELECT * FROM salesman CROSS JOIN customer WHERE salesman.city IS NOT NULL;
```  

(d.) Write a SQL statement to make a report with customer name, city, order number, order date, and order amount in ascending order according to the order date to find that either any of the existing customers have placed no order or placed one or more orders. Use customer and orders tables.
Solution

```sql
SELECT c.cust_name, c.city, order.ord_no, order.ord_date, order.purch_amt
FROM customer c
LEFT OUTER JOIN order
ON c.customer_id = order.customer_id
ORDER BY order.ord_date;
```

9. (5 points) In this problem, we consider *aggregation* of data.

(a.) Write a SQL statement to find the highest purchase amount ordered by the each customer on a particular date with their ID, order date and highest purchase amount. Sample table: orders, used in problem 7 part (a).

Solution

```sql
SELECT customer_id, ord_date, MAX(purch_amt)
FROM orders
GROUP BY customer_id, ord_date;
```

(b.) Write a SQL query to display the average price of each company’s products, along with their code. Sample table: item_mast.

<table>
<thead>
<tr>
<th>PROD_ID</th>
<th>PROD_NAME</th>
<th>PROD_PRICE</th>
<th>PROD_COMPANY</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Mother Board</td>
<td>3200</td>
<td>15</td>
</tr>
<tr>
<td>102</td>
<td>Key Board</td>
<td>450</td>
<td>16</td>
</tr>
<tr>
<td>103</td>
<td>ZIP drive</td>
<td>250</td>
<td>14</td>
</tr>
<tr>
<td>104</td>
<td>Speaker</td>
<td>550</td>
<td>16</td>
</tr>
<tr>
<td>105</td>
<td>Monitor</td>
<td>5000</td>
<td>11</td>
</tr>
<tr>
<td>106</td>
<td>DVD drive</td>
<td>900</td>
<td>12</td>
</tr>
</tbody>
</table>

Solution

```sql
SELECT AVG(prod_price), prod_company
FROM item_mast
GROUP BY prod_company
```

10. (4 points) Joins with multiple keys The point of this question is to explore how SQL handles cases where a join is performed on tables containing duplicate rows. Consider the following table item_mast.

<table>
<thead>
<tr>
<th>PROD_ID</th>
<th>PRODUCT</th>
<th>PROD_PRICE</th>
<th>PROD_COMPANY</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Mother Board</td>
<td>3200</td>
<td>1</td>
</tr>
<tr>
<td>102</td>
<td>Key Board</td>
<td>450</td>
<td>15</td>
</tr>
<tr>
<td>103</td>
<td>ZIP drive</td>
<td>250</td>
<td>16</td>
</tr>
<tr>
<td>104</td>
<td>Speaker</td>
<td>550</td>
<td>14</td>
</tr>
<tr>
<td>105</td>
<td>Monitor</td>
<td>5000</td>
<td>11</td>
</tr>
<tr>
<td>106</td>
<td>DVD drive</td>
<td>900</td>
<td>12</td>
</tr>
</tbody>
</table>
and a corresponding table of customer purchases, \( \text{purchases} \).

<table>
<thead>
<tr>
<th>PROD_ID</th>
<th>CUSTOMER</th>
<th>PRODUCT</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>James Hoog</td>
<td>Mother Board</td>
<td>New York</td>
</tr>
<tr>
<td>101</td>
<td>James Hoog</td>
<td>ZIP drive</td>
<td>Los Angeles</td>
</tr>
<tr>
<td>103</td>
<td>Mc Lyon</td>
<td>ZIP drive</td>
<td>Pittsburgh</td>
</tr>
</tbody>
</table>

Notice that in \( \text{item mast} \), the same product can appear multiple times (listed under different manufacturers). Also, in database purchases the same customer can appear multiple times. If we join carefully using select columns, we can identify observations uniquely in the resulting output table. However, suppose we join the two tables only on \( \text{item mast: product, product price and purchases: customer, product} \).

Draw a sample table describing what the output looks like, and explain the result.

**Solution**  Suppose we execute the following command:

```sql
SELECT item mast.PRODUCT, item mast.PROD PRICE, purchases.CUSTOMER
FROM item mast
INNER JOIN purchases
ON item mast.PRODUCT=purchases.PRODUCT;
```

then we get the following table as output:

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>PROD_PRICE</th>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mother Board</td>
<td>3200</td>
<td>James Hoog</td>
</tr>
<tr>
<td>Mother Board</td>
<td>2900</td>
<td>James Hoog</td>
</tr>
<tr>
<td>ZIP drive</td>
<td>250</td>
<td>James Hoog</td>
</tr>
<tr>
<td>ZIP drive</td>
<td>250</td>
<td>Mc Lyon</td>
</tr>
</tbody>
</table>

We can see that in the table item mast, there are 2 products named ‘Mother Board’ from different companies, and they both are shown in this output table. They have the same product ID 101, and since the table purchases doesn’t mention the company name, it is correct to show both of these in the output table. The product ‘ZIP drive’ has ID 103 in the table item mast, and 2 entries with ID 101 and 103 in the table purchases. When we join the tables, only the entry with ID 103 should be output. But we see that both the entries have been output. So, this is incorrect. To give the correct output, we need to join on both the prod id and product columns.
SELECT item mast.PRODUCT, item mast.PROD PRICE, purchases.CUSTOMER
FROM item mast
INNER JOIN purchases
ON item mast.PRODUCT=purchases.PRODUCT
AND item mast.PROD ID=purchases.PROD ID;

then we get the following table as output:

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>PROD_PRICE</th>
<th>CUSTOMER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mother Board</td>
<td>3200</td>
<td>James Hoog</td>
</tr>
<tr>
<td>Mother Board</td>
<td>2900</td>
<td>James Hoog</td>
</tr>
<tr>
<td>ZIP drive</td>
<td>250</td>
<td>Mc Lyon</td>
</tr>
</tbody>
</table>

11. **(10 points)** Implement logistic regression using tensorflow. Use the following code to generate train and test data. Note that we have set seed (using "random_state=42"). Use cross-entropy loss and gradient descent optimizer with a learning rate of 0.01. Use batch size of 100, and run for 500 steps. Report the accuracy on test set.

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Generate data
X_data, y_data = make_classification(n_samples=200, n_features=2,
n_redundant=0, random_state=42)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
test_size=0.2, random_state=42)

# Plot training data
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
plt.show()
```

**Solution**

```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np

# Generate data
X_data, y_data = make_classification(n_samples=200, n_features=2,
n_redundant=0, random_state=42)
```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data,
test_size=0.2, random_state=42)

# Plot training data
# plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
# plt.show()

y_train = y_train.reshape((-1,1))
y_test = y_test.reshape((-1,1))

# Define parameters
learning_rate = 0.01
batch_size = 100
num_steps=500
n_samples=X_train.shape[0]

# Define placeholders for input
X = tf.placeholder(tf.float32, shape=[None, 2])
y = tf.placeholder(tf.float32, shape=[None, 1])

# Define variables to be learned
W = tf.get_variable("weights", [2,1], initializer =
  tf.random_normal_initializer())
b = tf.get_variable("bias", [1,], initializer =
  tf.constant_initializer(0.0))
y_pred = tf.matmul(X,W)+b
loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=
y_pred, labels=y))

# Define optimizer
opt = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

with tf.Session() as sess:
  # Initialize Variables in graph
  sess.run(tf.global_variables_initializer())
  for _ in range(num_steps):
    # Select random minibatch
    indices = np.random.choice(n_samples, batch_size)
    X_batch, y_batch = X_train[indices, :], y_train[indices]
    # Do gradient descent step
    _, loss_val = sess.run([opt, loss], feed_dict={X: X_batch,
y: y_batch})
    print(loss_val)
  # Test model
  correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))
  # Calculate accuracy
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
  print("Accuracy:", accuracy.eval({X: X_test, y: y_test}))