1. Scans

(a) What are the different kinds of a scan?
(b) Sketch the principles of the scan strategies?
(c) When do you use which scan strategy?
(d) What is a scan semantic and what is it used for?

2. Sorting

(a) Why are external sorting algorithms needed? Sketch their functionality.
(b) Welche Kosten verursachen externe Sortieralgorithmen? Ist externes Sortieren mit logarithmischen Aufwand möglich?
(c) How much recursion steps does the sort-merge technique need for sorting 100,000,000 tuples (10 tuples per block) with a buffer size of mem = 3, 11, 101?

3. Join algorithms

Explain the following join algorithms:

(a) Nested-Loop Join
(b) Sort-Merge Join
(c) Hash Join

Use these strategies to join the Tables 1 and 2 on Person.PID=Residence.PID. For hashing, use the hash function \( h(x) = x \mod 3 \).

4. Given the algorithm 1 from the lecture to compute a join, extend the code of the merge-join so that also duplicates in R1 are handled for the join.

Good Luck!
<table>
<thead>
<tr>
<th>PID</th>
<th>Given Name</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Mareicke</td>
<td>Müller</td>
</tr>
<tr>
<td>2</td>
<td>Tom</td>
<td>Meier</td>
</tr>
<tr>
<td>3</td>
<td>Frank</td>
<td>Schmitt</td>
</tr>
<tr>
<td>5</td>
<td>Stefan</td>
<td>Schulz</td>
</tr>
</tbody>
</table>

Tabelle 1: Person

<table>
<thead>
<tr>
<th>PID</th>
<th>City</th>
<th>Street</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Hannover</td>
<td>Lindenstraße</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Magdeburg</td>
<td>Schillerstraße</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Hannover</td>
<td>Breiter Weg</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>Hamburg</td>
<td>Hafenstraße</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>Berlin</td>
<td>Gartenstraße</td>
<td>1</td>
</tr>
</tbody>
</table>

Tabelle 2: Residence

Algorithm 1 Merge-join algorithm from the lecture

Require: Relation R1 and R2
Ensure: Join of R1 and R2
1: R1ScanID := open-index-scan(R1IndexID, min(X), max(X));
2: R1TID := next-TID(R1ScanID);
3: R1Buffer := fetch-tuple(R1ID,R1TID);
4: R2ScanID := open-index-scan(R2YIndexID, min(Y), max(Y));
5: R2TID := next-TID(R2ScanID);
6: R2Buffer := fetch-tuple(R2ID,R2TID);
7: while not end-of-scan(R1ScanID) and not end-of-scan(R2ScanID) do
8:    if R1Buffer.X < R2Buffer.X then
9:       R1TID := next-TID(R1ScanID)
10:      R1Buffer := fetch-tuple(R1ID,R1TID);
11:    else if R1Buffer.X > R2Buffer.X then
12:       R2TID := next-TID(R2ScanID);
13:      R2Buffer := fetch-tuple(R2ID,R2TID);
14:    else
15:       insert into RES (R1Buffer.A1, ..., R1Buffer.An, R1.Buffer.X, R2Buffer.B1, ...., R1Buffer.Bm);
16:    end if
17: end while
18: close-scan (R1ScanID);
19: close-scan (R2ScanID);