Task 1: Given the following data set:

8, 12, 9, 5, 1, 3, 10, 14, 4, 6, 7, 11, 2, 13, 15, 16

Sort the data set using external sorting (Partition Size: 2). Why is external sorting suitable for sorting large data sets in databases?

Task 2: 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?

Task 3: Explain the following join algorithms:

(a) Nested-Loop Join
(b) Block Nested-Loop Join
(c) Sort-Merge Join
(d) 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 and a block size of 2 for the block nested loop join. Which of these algorithms could adjusted also be used for grouping?

<table>
<thead>
<tr>
<th>PID</th>
<th>First Name</th>
<th>Last 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>

<table>
<thead>
<tr>
<th>PID</th>
<th>Town</th>
<th>Street</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Hannover</td>
<td>Lindenstr</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>Magdeburg</td>
<td>Schillerstr</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>Hafenstr</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>Berlin</td>
<td>Gartenstr</td>
<td>1</td>
</tr>
</tbody>
</table>

Tabelle 1: Person

Tabelle 2: Wohnort
Task 4 Given the algorithm 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.

Algorithm 1 Merge-join algorithm from the lecture

Require: Relation R1 and R2
Ensure: Join of R1 and R2
1: R1ScanID := open-index-scan(R1XIndexID, 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: R1TID := next-TID(R1ScanID);
17: R1Buffer := fetch-tuple(R1ID,R1TID);
18: R2TID := next-TID(R2ScanID);
19: R2Buffer := fetch-tuple(R2ID,R2TID);
20: end if
21: end while
22: close-scan (R1ScanID);
23: close-scan (R2ScanID);

Task 5 Which problems exist for classical hash-joins in multi-core systems? How can the Radix join solve these problems?

Good Luck!