Advanced Topics in Databases
Exercise 7

Task 1 Given the following dataset:

\[ C = [12, 13, 18, 8, 2, 3, 2, 3, 1, 1, 1, 4, 6, 7, 8, 9] \]

(a) Create a Column Imprint for the given data set. Assume 8-bit Imprints, 32-bit integer as column values and 16 byte cache lines. Sample the data range on every second value in the data space.

(b) Execute the following queries on the created Imprint (you can express the result as bitmap or position list):
   i. \( C < 5 \),
   ii. \( C = 8 \),
   iii. \( 5 < C < 10 \).

Task 2 How well can Column Imprints and BitWeaving support the following use cases?

(a) Equality predicates (e.g., \( C = 4 \))
(b) Between predicates (e.g., \( 2 < C < 7 \))
(c) Multi-dimensional queries (e.g., \( C_1 < 5 \) AND \( C_2 < 300 \))
(d) Analytical workloads with frequent insertions (e.g., daily sales).
(e) Storing and querying strings (char or varchar)

Task 3 Given the following dataset:

<table>
<thead>
<tr>
<th>TID</th>
<th>( C_1 )</th>
<th>( C_2 )</th>
<th>( C_3 )</th>
<th>( C_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>T4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

(a) Build a tree-based Elf with the following column Order: \( C_3, C_4, C_2, C_1 \).
(b) Linearize the build elf.
(c) Execute the following queries:
   i. \( C_3 = 0 \)
   ii. \( C_2 < 1 \) AND \( 0 < C_3 < 2 \)
   iii. \( C_1 = 1 \)
Task 4 State and explain the hazards discussed in lecture. What is forwarding (sometimes also called bypassing) and what hazard can be mitigated with it?

Task 5 What hazards can be found in the following assembler code snippets? To execute the code snippets in a RISC architecture without reordering, how much pipeline stalls would have to be included? What are possible points where a reordering makes sense?

```
ADDI F2, F0, #5
ADD F1, F0, F0
ADDI F4, F1, #5
L1: SUBI F3, F3, #1
    ADDI F2, F2, #2
    ADDI F1, F1, #1
    BNE F2, F1, L1
DADDU F2, F3, F4
    BEQZ F2, L1
    LW F1, 0(F2)
    DADDU R1, R2, R3
    BEQZ R4, L1
    DSUBU R1, R5, R6
L1: OR R7, R1, R8
```

Task 6 Loop unrolling can be used to reduce data hazards in tight loops. Considering the following database tasks:

- Selections
- Compression
- Hashing
- Sorting
- Grouping
- Aggregation
- Insert/Update/Delete

Where could those tight loops occur?

Good Luck!