Question 1: Nested-Loop-Join and Hash-Join (1 P.)

We want to join the relations `Invoice` and `Item`. There are 200 000 invoices and 500 000 items. Each tuple of each relation fits in exactly one page. The join buffer has 152 pages and the result tuples of the join also fit into one page (We perform the projection during the join). For this join `Item` is the inner relation and `Invoice` the outer.

a) Instead of counting all page accesses, we now would like to estimate the amount of sequential page accesses (e.g.: Loading the first 152 pages of a table into the buffer counts as 1 sequential page access).

   In the Block-Nested-Loop join from the lecture as many tuples of the outer relation as possible are stored in the buffer. Now we will fill the buffer with 50% inner tuples and 50% outer tuples. Calculate the amount of sequential page accesses, that are required to calculate the join.

b) We now want to perform a hash join, with the hash function mod k. Determine k, such that the buffer is utilized optimally, and calculate the amount of sequential page accesses for the join. You can ignore the write operations after partitioning and assume that all relevant attributes are uniformly distributed natural numbers ≥ 1

Question 2: Join Implementation (1 P.)

This question requires you to implement code in any programming language you want. The code has to compile and return the correct result. **Submit the code as a separate file (or archive, if you have more than one source file).** If you use a different language than Java, please provide instructions on how to compile and run your code. In OLAT, we provide a Java template with most of the boilerplate code already in place.

Please delete all indices that were created on the tables `lineitem` and `orders`.

a) Implement the following query[1]:

```
SELECT l_orderkey, l_shipdate, o_orderdate
FROM lineitem JOIN orders ON l_orderkey = o_orderkey
```

   First you should load the required values from the TPC-H database into lists. You then execute the nested loop join and the hash-based nested join once. The index should be a suitable `HashMap`.

b) Measure the execution time of your implementations (including the index creation time) and compare it to the time Postgres requires for the same query. How do you explain the differences?

c) Which join algorithm did Postgres choose? Make Postgres perform a merge join and measure the execution time.

[1] In the example code we only load tuples where `orderkey < 50000`, you may try bigger values if you want.

Question 3: External Sort (1 P.)

a) A file consisting of 300 000 blocks is to be sorted. The sorting should take not more than 4 merge phases. Is this possible using standard external sort with an in-memory buffer of 30 pages, and if it is possible, what is the minimum buffer size required?

b) We want to use external sort with blocked I/O. Derive a formula that identifies the minimum number of I/O operations (i.e., number of chunks read) required for sorting a file of N (disk) blocks/pages depending on b. Give a sample result of this reasoning for $N = 250000$ and $n_B = 20$, e.g., by computing cost for reasonable values of b (no closed form optimization required). To keep things simple, consider only read operations.

Question 4: V-Optimal Histograms (1 P.)

a) Given the following table of values and frequencies. Calculate the v-optimal histogram for $B = 2$ cells. Use the algorithm that was discussed in the lecture.

Please write down P, PP and a table with all calculated $SSE^*(i, k)$ values and note when and how it was updated. As final result, the histogram bounds should be provided.

Value	Frequency
1	12
2	9
3	0
4	4
5	3

b) Formally show that

$$SSE([i, j]) = \sum_{i \leq k \leq j} (F[k]^2) - (j - i + 1) * AVG([i, j])^2$$