1) For each of the following join algorithms, circle whether it supports theta (non-equi) join.

1. (Yes / No) Block Nested Loop Join  
2. (Yes / No) Index Nested Loop Join  
3. (Yes / No) Sort Merge Join  
4. (Yes / No) Grace Hash Join

2) Suppose we have the following two tables.
   • customers (cid, name, age, ...)  
     o cid is the primary key;  
     o 4,000 records (total), 300 pages;  
     o An unclustered B+ tree on 'cid';  
   • orders (date, cid, price, ...)  
     o 25,000 records (total), 500 pages;  
     o 'cid' is a foreign key to 'customers.cid';  
     o An unclustered B+ tree on 'cid';  
     o The 'cid' values are uniformly distributed.

We want to join the tables on their 'cid' attributes, i.e., customers.cid = orders.cid, assume data is not skewed.

1. Consider the case where we want to use Simple Hash Join. Assume we have B = 153 in-memory buffer pages. We treat the smaller table as the outer table to build the in-memory hash table. Calculate the number of disk I/Os, excluding the final writes.

2. Consider the case where we want to use Simple Hash Join. Assume we have B = 78 in-memory buffer pages. We treat the smaller table as the outer table to build the in-memory hash table. Calculate the number of disk I/Os, excluding the final writes.

3. Consider the case where we want to use Grace Hash Join. Assume we have B = 16 in-memory buffer pages. We treat the smaller table as the outer table to build the in-memory hash table. Calculate the number of disk I/Os in the partitioning/building phase and the number of disk I/Os in the probing phase, excluding the final writes.
1. For each of the following join algorithms, circle whether it supports theta (non-equi) join.

1. (Yes / No) Block Nested Loop Join  2. (Yes / No) Index Nested Loop Join
3. (Yes / No) Sort Merge Join  4. (Yes / No) Grace Hash Join

2.1
We use \( B - 3 = 150 \) pages for in-memory hash table

Smaller relation is customers \( C \) and we can load half of relation \( C \) and build a hash table in memory. That means we will scan relation orders \( O \) and probe the in-memory hash table. Reading \( C \) and \( O \) + writing and reading half of \( C \) and half of \( O \)

\[
(|C| + |O|) + 2 \times (|0.5C| + |0.5O|) \\
= (300 + 500) + 2 \times (300 + 500) = 2400
\]

2.2
We use \( B - 3 = 75 \) pages for in-memory hash table

Smaller relation is customers \( C \) and we can load 75 pages of relation \( S \) each iteration and build a hash table in memory. That means we will scan relation orders \( O \) and probe the in-memory hash table for each iteration. Assuming data is not skewed, we can approximate the proportion of pages written to disk for \( S \) and apply the same for \( R \). If data is skewed, we can't guarantee that.

Reading \( C \) and \( O \) +
writing and reading of 225 pages of \( C \) and 375 pages of \( O \) +
writing and reading of 150 pages of \( C \) and 250 pages of \( O \) +
writing and reading of 75 pages of \( C \) and 125 pages of \( O \)

\[
(|C| + |O|) + 2 \times (0.75C| + 0.75O|) + 2 \times (0.5C| + 0.5O|) + 2 \times (0.25C| + 0.25O|) \\
(300 + 500) + 2 \times (225 + 375) + 2 \times (150 + 250) + 2 \times (75 + 125) = 3200
\]

2.3
partitioning/building phase:

We use \( B - 1 = 15 \) pages for hash buckets.

needs two passes as \( B-1 = 15 \) and \( 15 \times 15 < 300 < 500 \) for customers:
pass 0:
15 partitions with 20 pages each. Each page is read and written once, total 300 reads and 300 writes

pass 1:
each partition is read back and repartition to 15 sub-partitions, each sub-partition will be ceiling (20 / 15) = 2 pages. Total 300 page reads and 15 * 15 * 2 = 450 writes

for orders:

pass 0:
15 partitions with ceiling (500/15) = 34 pages each. Each page is read and written once. Total 500 page reads and 15 * 34 = 510 writes

pass 1:
each partition is read back and repartition to 15 sub-partitions, each sub-partition will be ceiling (34/15) = 3 pages. Total 510 reads and 15 * 15 * 3 = 675 writes.

Probing phase:
Read costumers’s and orders’s corresponding partition back. Total 450 + 675 reads.

Total: (300 + 500) + (300 + 510) * 2 + (450 + 675) * 2 = 4670