Q: Implement `UNION` operator. See tutorial slides posted on course website.

Q: Implement `INTERSECTION` operator.

Based on Sort-Merge Join. Use the basic 2-pass sort-merge join algorithm from the lecture slides. In the merge phase, call `GetMin` on all runs of R and S:

Pseudocode version:

```
countR <- 0
countS <- 0
lastTuple <- null
for tuple, relation <- GetMin(runs) do
      if tuple != lastTuple and lastTuple != null then
            if countR >= 1 and countS >= 1 then
                  Output(lastTuple)
            countR = 0
            countS = 0
      if relation == R then
            countR <- countR + 1
      else
            countS <- countS + 1
      lastTuple = tuple
```

Text version:
Use min-heap to get the minimum tuple from all runs of both R and S. For each tuple got, keep track of its source relation. Only output a distinct tuple when it appears at least once in each relation.

Based on Block Nested Loop Join. We can only use 1-Pass Block Nested Loop join algorithm, that loads the entire S relation (or the smaller one of R and S) in memory buffer. Because we need to have a complete view of S in order to deduplicate.

1. Load the entire S relation into memory buffer
2. Find all distinct tuples in S relation
3. Load each page of R into a buffer page. For each tuple in R in the buffer, if there is a matching tuple in S, output the tuple, and remove the matching tuple in S.

Based on Hash Join. Remember Hash Join is just partitioning using a hash function, and run Block Nested Loop Join on each pair of corresponding of partitions from R and S. Because of the hash function, each partition can only join with tuples from the corresponding partition. So here it is the same: after partitioning both R and S using a hash function, just run the `INTERSECTION` algorithm based on Block Nested Loop Join on each partition.