[60 points] Consider natural-joining tables R(a, b) and S(a,c). Suppose we have the following scenario.

- R is a clustered relation with 10,000 blocks.
- S is a clustered relation with 20,000 blocks.
- 102 pages available in main memory for the join.
- Assume the output of join is given to the next operator in the query execution plan (instead of writing to the disk) and thus the cost of writing the output is ignored.

Describe the steps for each of the following join algorithms. For sorting and hashing-based algorithms, also indicate the output sizes from each step. What is the total number of block I/O's needed for each algorithm? Which algorithm is most efficient in terms of block's I/O?

a. [10 points] (Block-based) nested-loop join with R as the outer relation.

Steps:
1. Load 100 blocks of R (considering 2 pages are used for other purposes) into memory.
2. For each loaded set of R blocks, scan each block of S sequentially.
3. For each tuple in the R blocks, compare with each tuple in the current block of S.
4. If the tuples match on the joining attribute, output the joined tuple to the next operator in the query plan.

Number of Block I/Os:

a. Read R one time: $B(R) = 10000$

$\frac{B(R)}{100}$ loops to read S: $\frac{10000}{100} \times B(S) = 2000000$

Total: $B(R) + \frac{B(R)}{100} \times B(S) = 10000 + 2000000 = 2010000$ block I/Os

2010000 Block I/Os

b. [10 points] (Block-based) nested-loop join with S as the outer relation.

Steps:
1. Load 100 blocks of S into memory.
2. For each loaded set of S blocks, scan each block of R sequentially.
3. For each tuple in the S blocks, compare with each tuple in the current block of R.
4. If the tuples match the joining attribute, output the joined tuple.

Number of Block I/Os:

b. Read R one time: $B(R) = 20000$

$\frac{B(R)}{100}$ loops to read S: $\frac{10000}{100} \times B(S) = 2000000$

Total: $B(R) + \frac{B(R)}{100} \times B(S) = 20000 + 2000000 = 2020000$ block I/Os

2020000 Block I/Os

c. [20 points] Sort-merge join (assume only 100 pages are used for sorting and 101 for merging). Note that if join cannot be done by using only a single merging pass, runs from one or both relations need to be further merged, in order to reduce the number of runs. Select the relation with a larger number of runs for further merging first if both have too many runs.

Steps:
1. Sort R using 100 pages of memory, creating sorted runs.
2. Sort S using 100 pages of memory, creating sorted runs.
3. Merge the sorted runs from R and S in a single pass, using 101 pages for merging.

Number of Block I/Os:

C

Sort R: $2 \times B(R) = 2 \times 10000 = 20000$

merging 100 runs from R $\Rightarrow$ $2B(R) = 20000$

Sort S: $2 \times B(S) = 2 \times 20000 = 40000$

merging 200 runs from S $\Rightarrow$ $2B(S) = 40000$

Merge R and S: $B(R) + B(S)$

$= 5B(R) + 5B(S) = 150{,}000$ block I/Os

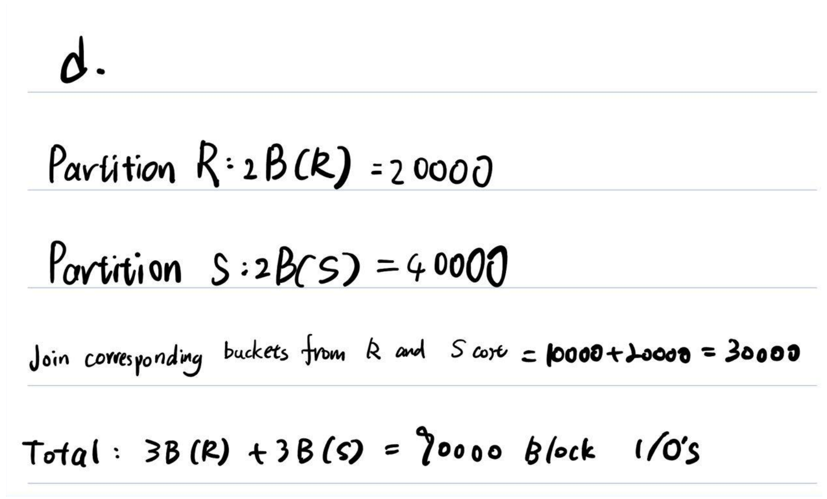
150000 Block I/Os

d. [20 points] Partitioned-hash join (assume 101 pages used in partitioning of relations and no hash table is used to lookup in joining tuples). Note if buckets are still too large to join within memory, you should further partition them.

Steps:

1. Partition R into 101 buckets using a hash function, which fits into memory.
2. Partition S into 101 buckets using the same hash function.
3. Join each bucket of R with the corresponding bucket of S in memory.

Number of Block I/Os:



d.

Partition R: 2B(R) = 20000

Partition S: 2B(S) = 40000

Join corresponding buckets from R and S cost = 10000+20000 = 30000

Total: 3B(R) + 3B(S) = 90000 block I/O's

90000 Block I/Os

Most Efficient Algorithm in Terms of Block I/O:
The Partitioned-hash join is the most I/O efficient with 30,000 block I/Os, significantly less than the Sort-merge join with 90,000 block I/Os, and far more efficient than the Block-based Nested-loop joins with over 2 million block I/Os each.