Q2. Consider natural-joining tables R(a, b) and S(a,c). Suppose we have the following scenario.
i. R is a clustered relation with 10,000 blocks.
ii. S is a clustered relation with 20,000 blocks.
iii. 102 pages available in main memory for the join.
iv. 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 sizes of output 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?

M = 102
M-2 =100
B(R) = 10000 blocks
B(S) = 20000 blocks

a.[10 points] (Block-based) nested-loop join with R as the outer relation.
<u>Solution</u>
A: R ⋈ S
Pseudocode:

```
For each (102-2) blocks br of R do:
    For each block bs of S do:
        For each tuple r in br do:
            For each tuple s in bs do:
                If "r and s join" then output (r, s)
```

Number of block I/O's:
1. Read R once, cost B(R) = 10000 blocks
2. In main memory it can load 100 blocks at a time when outer loop is executed.
3.R as the outer relation, so it will has B(R)/(M-2) = 10000/100 = 100 time iterations in outer loop.
Each time,100 blocks are loaded into main menory
4. In each iteration, make one pass through R, scanning entire S to check every block in S and join tuples to output buffer.
So the cost is (B(R)/(M-2))*B(S) = 100*20000 = 2000000
Total Cost: B(R) + ((B(R)/(M-2))*B(S)) = 10000 + ((10000/100)*20000) = **2010000 block I/O's**

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

M = 102
M-2 =100

B(R) = 10000 blocks
B(S) = 20000 blocks

Pseudocode:

```
 For each (102-2) blocks bs of S do:
       For each block br of R do:
             For each tuple s in bs do:
                For each tuple r in br do:
                   If "s and r join" then output (s, r)
```

Number of block I/O's:
1. S as the outer relation, make a pass through S, cost B(S) = 20000 blocks.
2. In outer loop, it will execute B(S)/(M-2) = 20000/100 = 200 times. Each time, 200 blocks are loaded into main memory.
3. In inner loop, each time scanning R in input buffer to check every block in R and join tuples to output buffer.
The cost is (B(S)/(M-2))*B(R) = (20000/100)*10000 = 200*10000 = 2000000
4. Total Cost: B(S) + ((B(S)/(M-2))*B(R)) = 20000 + ((20000/100)*10000) = **2020000 block I/O's**

c. [20 points] Sort-merge join (assume only 100 pages are used for sorting and 101 pages 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.
<u>Solution</u>

M=101 blocks
B(R) = 10000 blocks
B(S) = 20000 blocks

-> 100 pages are used for sorting
so there are:
    B(R)/100 = 10000/100 = 100 runs (each size 100) of R
    B(S)/100 = 20000/100 = 200 runs (each size 100) of S
The cost (sort and write) is 2B(R)+2B(S) = 2*10000 + 2*20000 = 20000 + 40000 = 60000 blocks.
Send them back to disk.

-> merging - 1 = 101-1 = 100,
   while B(R)+ B(S) = 10000 + 20000 = 30000
   results into B(R)+ B(S) >= (M-1)M ==> 30000 >= (101-1)*100 ==> 30000 >= 10000.
   So, we cannot merge M-1 runs from R and S directly
   We need to merge:

R into 100/100 = 1 run, Cost = 4B(R)
S into 200/100 = 2 runs Cost = 4B(S)

-> Join by merging 1 run with 2 runs, Cost = B(R) + B(S)

-> Total = 5B(R) + 5B(S) = 5(B(R) + B(S)) = 5*(30000) = **150000**

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.

<u>Solution</u>

B(R) = 10000 blocks
B(S) = 20000 blocks

-->
Hash R into M-1(100) buckets, 100 blocks/bucket(R1)
Hash S into M-1(100) buckets, 200 blocks/bucket(S1)
Cost : 2*B(R) + 2*B(S)

--> Extra hash is required as sum of bucket R and S is more than 100
Hash R into 100 buckets, 1 block/bucket(R11)
Hash S into 100 buckets, 2 blocks/bucket(S11)
Cost : 2*B(R) + 2*B(S)

-> Join by merging R with S, Cost = B(R) + B(S)

-> Total = 5B(R) + 5B(S) = 5(B(R) + B(S)) = 5*(30000) = **150000**

Q. Which algorithm is most efficient in terms of block's I/O?

The Sort-merge join and Partitioned-hash join both are the most efficient in terms of block's I/O.