Exercise 1: Assume (for simplicity in this exercise) that only one tuple fits in a block and memory holds at most 3 blocks. Show the runs created on each pass of the sort-merge algorithm, when applied to sort the following tuples on the first attribute:
(kangaroo, 17), (wallaby, 21), (emu, 1), (wombat, 13), (platypus, 3), (lion, 8), (warthog, 4), (zebra, 11), (meerkat, 6), (hyena, 9), (hornbill, 2), (baboon, 12).

Answer. We will refer to the tuples (kangaroo, 17) through (baboon, 12) using tuple numbers t1 through t12. We refer to the jth run used by the i th pass, as ri j . The initial sorted runs have three blocks each. They are:
r11 = {t3, t1, t2}
r12 = {t6, t5, t4}
r13 = {t9, t7, t8}
r14 = {t12, t11, t10}
Each pass merges two runs. Therefore the runs after the end of the first pass are:
r21 = {t3, t1, t6, t5, t2, t4}
r22 = {t12, t11, t10, t9, t7, t8}
At the end of the second pass, the tuples are completely sorted into one run:
r31 = {t12, t3, t11, t10, t1, t6, t9, t5, t2, t7, t4, t8}

Exercise 2: Consider the following SQL query for our bank database:

select T.branch-name
from branch T, branch S
where T.assets > S.assets and S.branch-city = "Brooklyn"

Write an efficient relational-algebra expression that is equivalent to this query. Justify your choice.

Answer.

$$\pi_{\text{T.branch name}}((\pi_{\text{branch name, assets}}(\rho_{T}(\text{branch})))\ \sigma_{\text{T.assets > S.assets}}\ (\pi_{\text{assets}}\ (\sigma_{(\text{branch city= 'Brooklyn'})}(\rho_{S}(\text{branch})))))$$

This expression performs the theta join on the smallest amount of data possible. It does this by restricting the right hand side operand of the join to only those branches in Brooklyn, and also eliminating the unneeded attributes from both the operands.

Exercise 3: Let relations r1(A, B,C) and r2(C, D, E) have the following properties: r1 has 20,000 tuples, r2 has 45,000 tuples, 25 tuples of r1 fit on one block, and 30 tuples of r2 fit on one block. Estimate the number of block transfers and seeks required, using each of the following join strategies for $r1 \bowtie r2$:
a. Nested-loop join. b. Block nested-loop join. c. Merge join. d. Hash join.

Answer.
r1 needs 800 blocks, and r2 needs 1500 blocks. Let us assume M pages of memory. If M > 800, the join can easily be done in 1500 + 800 disk accesses, using even plain nested-loop join. So we consider only the case where M ≤ 800 pages.
a. Nested-loop join: Using r1 as the outer relation we need 20000 ∗ 1500 + 800 = 30, 000, 800 disk accesses, if r2 is the outer relation we need 45000 ∗ 800 + 1500 = 36, 001, 500 disk accesses.
b. Block nested-loop join: If r1 is the outer relation, we need $\lceil 800 / (M-1) \rceil * 1500 + 800$ disk accesses, if r2 is the outer relation we need $\lceil 1500 / (M-1) \rceil * 800 + 1500$ disk accesses.

c. Merge-join: Assuming that r1 and r2 are not initially sorted on the join key.
$1500(2\lceil \log_{M-1}(1500/M)\rceil + 1)$ is the cost of sorting relation r2.
The total sorting cost inclusive of the output is Bs(total number of block transfers) = $1500(2\lceil \log_{M-1}(1500/M)\rceil + 1) + 800(2\lceil \log_{M-1}(800/M)\rceil + 1)$ disk accesses. Assuming all tuples with the same value for the join attributes fit in memory, the total cost is Bs + 1500 + 800 disk accesses.
In Merge join, if relations are not sorted, the total number cost = cost to sort the relations + cost to merge two relations (1500+800 disk access in the case). Refer to page 20 of the PDF uploaded on Backpack.

d. Hash-join: We assume no overflow occurs. Since r1 is smaller, we use it as the build relation and r2 as the probe relation. If M > 800/M, i.e. no need for recursive partitioning, then the cost is 3(1500+800) = 6900 disk accesses, else the cost is $2(1500 + 800)\lceil \log_{M-1}(800) - 1\rceil + 1500 + 800$ disk accesses.

Exercise 4: Let r and s be relations with no indices, and assume that the relations are not sorted. Assuming infinite memory, what is the lowest cost way (in terms of I/O operations) to compute $r \bowtie s$? What is the amount of memory required for this algorithm?

Answer. We can store the entire smaller relation in memory, read the larger relation block by block and perform nested loop join using the larger one as the outer relation. The number of I/O operations is equal to br + bs , and memory requirement is min(br, bs) + 2 pages.

Exercise 5: Suppose you need to sort a relation of 40 gigabytes, with 4 kilobytes block, using a memory size of 40 megabytes. Suppose the cost of a seek is 5 milliseconds, while the disk transfer rate is 40 megabytes per second. Find the number of block transfers and number of seeks with the number of buffers allocated are 2, i.e., b_b=2.

Answer. Given:
Relation size = 40 gigabytes
Block size = 4 kilobytes

Memory Size = 40 megabytes
Seek Time = 5 milliseconds
Disk Transfer Rate = 40 megabytes per second

b_r = number of relation blocks = relation size/block size = 40 gigabytes/4 kilobytes = 10,000,000 blocks
M = number of memory blocks = memory size/block size = 40 megabytes/4 kilobytes =10,000 blocks
Initial number of runs = b_r/M = 10,000,000/10,000 = 1000
Number of merge passes required = $\lceil \log_{M-1}(b_r/M) \rceil = \log_{9999}(1000) = 1$
B (number of block transfers) = $b_r * (2\lceil \log_{M-1}(b_r/M) \rceil + 1) = 10{,}000{,}000 * (2[1] + 1)$ = 30,000,000 transfers.

Find number of disk seeks
S = number of seeks = $2\lceil b_r/M \rceil + \lceil b_r/b_b \rceil (2 * \lceil \log_{M-1}(b_r/M) \rceil - 1) = 2[1000] + [10000000/2] (2*[1] - 1)$ = 5,002,000 seeks

Exercise 6: Suppose that a B+ tree index on (dept_name, building) is available on relation department. What should be the best way to handle the following selection?

$$\sigma_{(\text{building} < \text{“Watson”}) \Lambda (\text{budget} < 55000) \Lambda (\text{dept\_name} = \text{“music”})}(\text{department})$$

Answer. By using indexing, locate the first tuple having dept_name=music. Now retrieve successive tuples using pointers as long as building is less than Watson.

From the tuples retrieved, the ones not satisfying the condition (budget<55000) are rejected.