1: Query Processing

Consider the natural join of relations Student(Name, Project) and Prof(Name, Project) on attribute Project, i.e., Student.Project = Prof.Project. The size of relation Student is 2,000 blocks and the size of relation Prof is 1,000 blocks. Assume these relations are not sorted by any of their attributes.

1. Assume that we have 201 blocks available in memory buffer, i.e., M=201. Check whether it is possible to join these relations using the sort-merge join algorithm and compute the cost of this join. Repeat the aforementioned steps for the optimized sort-merge algorithm.

   Solution:

   Because the size of each relation is less than or equal to M^2, we can use sort-merge join. The cost of this join will be 5 (B(Student) + B(Prof)) = 5 × (2000 + 1000) = 15000.

   Because B(Student) + B(Prof) = 3000 is less than or equal to M^2, we can also use optimized sort-merge join algorithm to join these relations. The cost will be 3 (B(Student) + B(Prof)) = 3 (2000+1000) = 9000.

2. Describe a situation where the aforementioned sort-merge join algorithms may need additional disk I/O accesses.

   Solution: If all students and professor work on one project, we have to join all tuples in these relations. In other words, we have to join the tuples in each run of Student with the tuples in every run of Prof. Hence, sort-merge algorithms need more I/O accesses for each run than the numbers given in the previous part of the problem. The cost will be close to a nested-loop join and almost equal to B(Student) × B(Prof).

2: Query Processing

Consider the following relational schema.

Student (StudentID, Name)
HealthInsurance (InsuranceID, DateStarted, StudentID)

Assume that number of blocks in relation Student is 10,000 and the number of blocks in relation HealthInsurance is 300. We do not have any indexes on these relations. We like to join these relations on StudentID.

1. If we can have 12,000 blocks in main memory, i.e., M = 12,000, explain what is the fastest join algorithm to join Student and HealthInsurance on StudentID and analyze its cost?

   Solution:

   Because we can fit both relations in main memory, we may use an internal memory join algorithms to join them. We may pick the internal memory version of nested-loop, sort-merge, or hash-join as they involve equal number of I/O accesses. The cost of join will be B (HealthInsurance) + B (Student) = 10300 disk I/O's. Note that we do not care about the cost of memory operations when analyzing the cost of query processing algorithms.

2. If we can fit 20 blocks in main memory,i.e., M= 20, what is the fastest join algorithm to join these relations? Analyze its cost.

   Solution:

   Hash-join and optimized sort-merge join are the fastest algorithms in the given setting and have equal costs. However, to apply optimized sort-merge join, we should have B (HealthInsurance) + B (Student) $\leq M^2$. As B(HealthInsurance) + B (Student) is not less than or equal to 400, we cannot use optimized sort-merge algorithm. Hash-join algorithm, however, requires smaller amount of main memory. Because the smaller relation is HealthInsurance, we need to have B(HealthInsurance) $\leq M^2$ or B(HealthInsurance) ≤ 400 to perform this join using hash-join algorithm. As we have B(HealthInsurance) ≤ 400, we can use hash-join algorithm for the join. The cost of hash-join for this join is 3 B(HealthInsurance) + 3 B (Student) = 30900 number of I/O access.

3: Query Processing

Assume that we like to sort a relation R. Answer the following questions.

1. If R is too large to fit in the main memory and does not have any index, explain which algorithm we can use to sort R and analyze it cost for the sort.

   Solution:

   We may use two-pass or general, multi-way merge-sort algorithm. You may find the costs in the lecture notes.

2. Assuming we can fit 200 blocks in main memory,i.e., M= 200, what should be the size of relation R (in blocks) to use two-pass multi-way merge-sort algorithm to sort it? What will be the maximum cost for this sort?

   Solution: A relation of B blocks can be sorted using two-pass multi-way merge-sort algorithm, as long as its number of blocks is no more than M^2. Hence, if the size of R is less than or equal to 40,000, we can use two-phase, multiway merge-sort algorithm to sort it. The cost of the sort is $3 \times$ B(R), which will be at most $3 \times 40{,}000 = 120{,}000$ number of I/O accesses. Note that because this sort is a standalone operation and is not used as the part of another operation, we do not consider the cost of writing output on disk in our cost analysis.