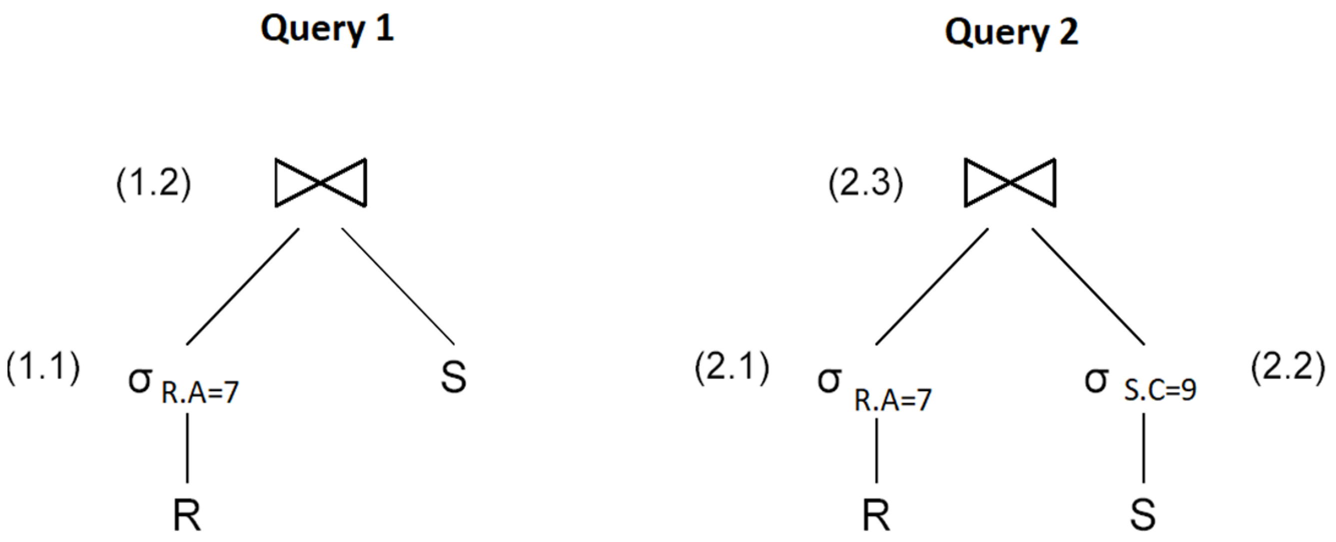
**Question 1a:**

Consider the following queries on relations R(A,B) and S(B,C):

- **Query 1:** SELECT * FROM R, S WHERE R.B = S.B AND R.A = 5
- **Query 2:** SELECT * FROM R, S WHERE R.B = S.B AND R.A = 5 AND S.C = 10

The following two plans are being considered:




Query 1 Query 2

(1.2) (2.3)

(1.1) (2.1) (2.2)

R.A=5 S R.A=5 S.C=10

R R S

Relation R has **50,000 tuples** with **20 tuples per disk block**, V(R,A)=5, and V(R,B)=10.
Similarly, S has **25,000 tuples** with **25 tuples per disk block**, V(S,B)=4, and V(S,C)=15. Assume values are distributed over possible V(Relation,Attribute) values (not over possible domain values).

In this exercise, we make the following additional assumptions:

- The join is implemented as a **hash-join**;
- The intermediate result produced by operation (1.1) is **not written to disk**;
- Hash buckets are stored on disk;
- The final result is kept in **main-memory**;
- There is enough memory to execute the hash join algorithm.

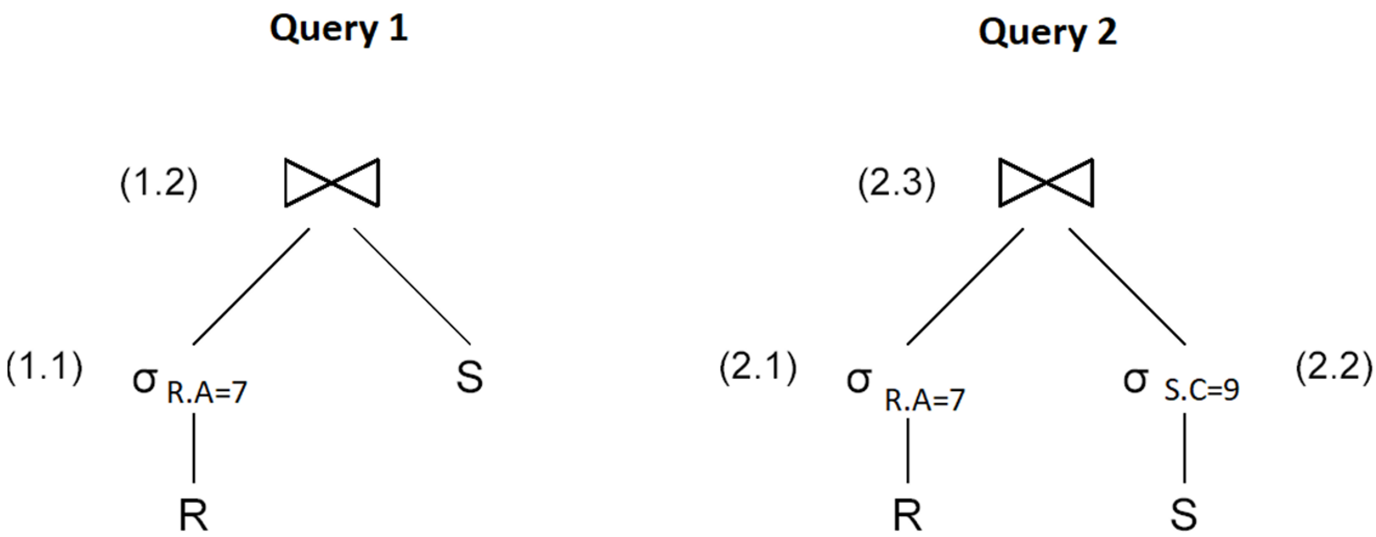
**a)** How many disk I/O's does **Query 1** require?
**b)** Also assume that the result of neither (2.1) nor (2.2) is stored to disk. How many disk I/O's does **Query 2** require?

**Question 1b:**

Consider the following queries on relations R(A,B) and S(B,C):

- **Query 1:** SELECT * FROM R, S WHERE R.B = S.B AND R.A = 3
- **Query 2:** SELECT * FROM R, S WHERE R.B = S.B AND R.A = 3 AND S.C = 7

The following two plans are being considered:




Query 1 Query 2

(1.2) (2.3)

(1.1) (2.1) (2.2)

R.A=3 S R.A=3 S.C=7

R R S

Relation R has **40,000 tuples** with **15 tuples per disk block**, V(R,A)=4, and V(R,B)=8.
Similarly, *SS* has **20,000 tuples** with **20 tuples per disk block**, V(S,B)=3, and V(S,C)=10. Assume values are distributed over possible V(Relation,Attribute) values (not over possible domain values).

In this exercise, we make the following additional assumptions:

- The join is implemented as a **hash-join**;
- The intermediate result produced by operation (1.1) is **not written to disk**;
- Hash buckets are stored on disk;
- The final result is kept in **main-memory**;
- There is enough memory to execute the hash join algorithm.

**a)** How many disk I/O's does **Query 1** require?
**b)** Also assume that the result of neither (2.1) nor (2.2) is stored to disk. How many disk I/O's does **Query 2** require?

---

**Question 1c:**

Consider the following queries on relations R(A,B) and S(B,C):

- **Query 1:** SELECT * FROM R, S WHERE R.B = S.B AND R.A = 9
- **Query 2:** SELECT * FROM R, S WHERE R.B = S.B AND R.A = 9 AND S.C = 12

The following two plans are being considered:

Query 1 Query 2

(1.2) (2.3)

(1.1) (2.1) (2.2)

R.A=9 R.A=9 S.C=12

R R S

Relation R*R* has **70,000 tuples** with **25 tuples per disk block**, V(R,A)=7, and V(R,B)=14.
Similarly, S*S* has **35,000 tuples** with **35 tuples per disk block**, V(S,B)=6, and V(S,C)=25. Assume values are distributed over possible V(Relation,Attribute) values (not over possible domain values).

In this exercise, we make the following additional assumptions:

- The join is implemented as a **hash-join**;
- The intermediate result produced by operation (1.1) is **not written to disk**;
- Hash buckets are stored on disk;
- The final result is kept in **main-memory**;
- There is enough memory to execute the hash join algorithm.

**a)** How many disk I/O's does **Query 1** require?
**b)** Also assume that the result of neither (2.1) nor (2.2) is stored to disk. How many disk I/O's does **Query 2** require?

---

**Key Features of These Questions:**

1. **Query Plans:**
   - Each query involves a join between two relations R*R* and S*S*, with additional filtering conditions.
2. **Relation Statistics:**
   - Each relation has a specified number of tuples, tuples per block, and distinct values for attributes.
3. **Assumptions:**
   - The join is implemented as a hash-join.
   - Intermediate results are not written to disk.
   - Hash buckets are stored on disk.
   - The final result is kept in main-memory.

4. **Disk I/O Calculations:**
    - The goal is to compute the number of disk I/O's required for each query under the given assumptions.

---

**2a)** Suppose **B(R) = 30000**, **B(S) = 70000**, and the number of main memory blocks is **M = 150**. We want to perform the **natural join of R and S using a hash join algorithm**.

a) How many partitions are needed to ensure that each partition of R fits in memory?
b) Describe the process of building and probing the hash table and compute the number of I/O operations required.
c) How many main memory blocks would be required to perform a one-pass hash join?

---

**2b)** Suppose **B(R) = 10000**, **B(S) = 25000**, and we have **M = 200** main memory blocks available. We wish to join R and S using a **sort-merge join algorithm**, assuming both relations are initially unsorted.

a) How many sorting passes are required for each relation?
b) Describe how the sorted relations are merged and the number of I/O's required.
c) What is the minimum number of main memory blocks needed for an efficient sort-merge join?

---

**2c)** Suppose **B(R) = 15000**, **B(S) = 40000**, and the available **main memory blocks are M = 120**. We want to perform a **nested-loop join of R and S**.

a) How many block accesses are required for a **tuple nested-loop join**?
b) How many block accesses are required for a **block nested-loop join**?
c) What is the minimum number of memory blocks needed to perform a **single-pass block nested-loop join**?

---

**Example 1: External Merge Sort for Join Preparation**

Suppose we have two relations, **R and S**, with the following properties: B(R)=15000B, B(S)=40000, The number of available **main memory blocks** is M=80.

- Both relations are **clustered but unsorted**, and we want to sort them before applying a **merge join**.

**Questions:**
**a)** How many passes do we need to sort R and S separately using external merge sort?
**b)** Describe the execution of the merge join and calculate the I/O cost.
**c)** How many main memory blocks would be needed to perform a one-pass join?

---

**Example 2: Sort-Merge Join with Different Memory Constraints**

Consider two relations:

- B(R)=30000, B(S)=70000, and **M = 120**.

- Both relations are **not sorted** and must be sorted before using a **merge join**.
- Assume **R and S are clustered** on disk.

**Questions:**
**a)** Compute the number of passes needed to sort **R and S** using **external merge sort**.
**b)** How many total I/O operations are required for the full sort-merge join process?
**c)** If we increase the available memory to M=250M = 250M=250, how would this affect the sorting and join efficiency?

---

**Example 3: Comparing Sort-Merge and Block Nested Loop Joins**

We have two relations: B(R)=50000 and B(S)=90000, The number of **main memory blocks** available is M=60.

- The relations are **not sorted**, and we want to perform a join.

**Questions:**
**a)** How many passes are required if we first sort both relations and use a **merge join**?
**b)** Compare the total **I/O cost** of using a **sort-merge join** versus a **block nested loop join** under these constraints.
**c)** What is the minimum number of memory blocks needed for a **one-pass join**, and is it feasible in this case?

---

## Question 1: Sort-Merge Join with Clustered Relations

Suppose $B(R)=15000$, $B(S)=30000$, and the number of main memory blocks is $M=201$. We want to perform the natural join of $R$ and $S$ using a **sort-merge join algorithm**. Both relations are clustered but neither is sorted.
a) How many passes are required to sort $R$ and $S$?
b) Describe how the join is executed and calculate the total number of I/O operations required.
c) How many main memory blocks would be needed to perform a **one-pass join**?

---

## Question 2: Hash Join with Unclustered Relations

Suppose $B(R)=25000$, $B(S)=40000$, and the number of main memory blocks is $M=151$. We want to perform the natural join of $R$ and $S$ using a **hash join algorithm**. Both relations are unclustered.
a) How many passes are required for the hash join?
b) Describe how the join is executed and calculate the total number of I/O operations required.
c) How many main memory blocks would be needed to perform a **one-pass join**?

---

## Question 3: Block-Nested Loop Join with Sorted Relations

Suppose $B(R)=10000$, $B(S)=50000$, and the number of main memory blocks is $M=51$. We want to perform the natural join of $R$ and $S$ using a **block-nested loop join algorithm**. Relation $R$ is sorted, but $S$ is not sorted.
a) How many passes are required for the block-nested loop join?

b) Describe how the join is executed and calculate the total number of I/O operations required.
c) How many main memory blocks would be needed to perform a **one-pass join**?

---

**3a)** We want to sort the tuples of a relation **S** on a given key. The following information is known:

- **S contains 200,000 tuples**, i.e., **T(S) = 200000**.
- The **size of a block** on disk is **4096 bytes**.
- The **size of each tuple in S is 512 bytes**.
- Relation S is **clustered**, meaning each disk block is fully occupied with S tuples.
- The **size of the sort key is 40 bytes**.
- A **record pointer** is **8 bytes**.

Answer the following questions:
a) If we use a **two-pass sorting algorithm**, what is the minimum amount of **main memory** (in number of blocks) required?
b) What is the **I/O cost** of the two-pass sorting algorithm, including the cost of writing the sorted file to disk?
c) If instead of sorting entire tuples, we sort only the **<key, recordPointer>** pairs and merge them later using record pointers to retrieve tuples, what is the **I/O cost** in this case?

---

**3b)** Consider a relation **T** that needs to be sorted on a particular key. Given the following information:

- **T contains 150,000 tuples**, i.e., **T(T) = 150000**.
- The **disk block size is 4096 bytes**.
- Each tuple in T has a size of **256 bytes**.
- Relation T is **clustered**.
- The **sort key is 48 bytes**.
- A **record pointer** is **8 bytes**.

Answer the following questions:
a) How many **blocks** are required to store relation **T**?
b) What is the **cost of two-pass sorting** for this relation in terms of **disk I/Os**?
c) If we use a **three-pass sorting algorithm** instead of two-pass sorting, what is the **minimum main memory required** in terms of blocks?

---

**3c)** Suppose we need to sort a **relation Q** based on a given key. The relation has the following properties:

- **Q contains 250,000 tuples**, i.e., **T(Q) = 250000**.
- The **block size on disk** is **4096 bytes**.
- The **size of each tuple** in Q is **320 bytes**.
- The relation is **clustered**.
- The **sort key is 36 bytes**.
- A **record pointer** is **8 bytes**.

Answer the following:
a) How many **disk blocks** are required to store **Q**?
b) If a **two-pass sorting algorithm** is used, what is the **minimum memory required** (in terms of blocks)?

c) What is the **disk I/O cost** if we sort only **<key, recordPointer>** pairs first, then retrieve tuples during the merge phase?

### Question 1: Merge Join with Limited Memory

Consider two relations **A** and **B** with the following properties:

- **B(A) = 15,000 blocks**, **B(B) = 40,000 blocks**, and the number of **available main memory blocks M = 50**.
- Both relations are **clustered** but **unsorted**.

**Questions:**

a) How many passes are required to sort **A** and **B** before performing a merge join?
b) Describe the steps involved in performing the **merge join** and determine the **total I/O cost**.
c) How much **main memory** would be needed to perform the join in **one pass** instead of multiple passes?

**Concept Explanation:**

- **Merge join** requires that both relations be **sorted first** if they are not already sorted.
- **External merge sort** requires **log-based passes** over the data to sort it.
- **Sorting cost** is **2B × (number of passes)** since each block is read and written multiple times.
- Once sorted, the **merge join** can be performed in **one pass**, requiring only **B(A) + B(B) I/Os**.
- If we had **enough memory to fit one entire relation**, the join could be completed in **one pass**.

### Question 2: Hash Join with Limited Partitions

Consider relations **X(P, Q)** and **Y(Q, R)**:

- **B(X) = 25,000 blocks**, **B(Y) = 60,000 blocks**.
- **Available main memory blocks M = 200**.
- We need to perform a **hash join** between X and Y.

**Questions:**

a) How many **partitions** will be created during the **partitioning phase** of the hash join?
b) What is the **total I/O cost** of performing the **hash join**?
c) If **M = 5,000**, would hash join still be the best option, or would another join method be preferable?

**Concept Explanation:**

- **Hash join** consists of two phases:
  1. **Partitioning**: Each relation is divided into **M-1 partitions**, ensuring that each partition fits in memory.
  2. **Probing**: Each partition of **X** is matched with its corresponding partition in **Y**.
- **Partitioning cost**: Each relation is **read and written once**.
- **Probing cost**: Each partition is **read again** and joined.
- **Total I/O cost** for hash join:

$$3(B(X) + B(Y))$$

- If **M is large enough** to fit one relation **entirely in memory**, a **one-pass join** is possible.

## Question 3: Comparing Nested-Loop Join and Block Nested-Loop Join

Consider two relations **M(S, T)** and **N(T, U)**:

- **B(M) = 35,000 blocks, B(N) = 80,000 blocks.**
- **Available main memory blocks M = 400.**

**Questions:**

a) What is the **I/O cost** if we use a **simple nested-loop join**?

b) How does using a **block nested-loop join** improve performance, and what is its I/O cost?

c) If **M were increased to 10,000**, would block nested-loop join still be the best choice?

**Concept Explanation:**

- **Simple Nested-Loop Join (NLJ):**
  - For each block of **M**, scan **N** entirely.
  - **I/O Cost**:

$$B(M) \times B(N) + B(M)$$

- **Block Nested-Loop Join (BNLJ):**
  - Load **M-1** blocks of **M** into memory at once.
  - Scan **N** using fewer passes.
  - **I/O Cost**:

$$\left(\frac{B(M)}{M-1}\right) \times B(N) + B(M)$$

    - **Larger M reduces cost**, making **BNLJ more efficient** than simple nested-loop join.

---

Here are three similar questions focusing on **external sorting, I/O optimization, and efficient sorting techniques**:

---

### Question 3a: Sorting a Relation with a Limited Memory Budget

We need to sort the tuples of a relation **S** on a given key. The following details are provided:

- **Relation S contains 200,000 tuples**, i.e., **T(S) = 200,000**.
- The **size of a block on disk** is **8,000 bytes**.
- The **size of each tuple** is **500 bytes**.
- Relation **S is clustered**.
- The **sort key size** is **40 bytes**, and a **record pointer** is **8 bytes**.

**Questions:**

a) What is the **minimum number of main memory blocks** required to sort the relation using a **two-pass sorting algorithm**?
b) What is the **total disk I/O cost** for sorting **S** using the two-pass algorithm, including writing the final sorted relation back to disk?
c) If we modify the sorting algorithm to sort **only** the <key, recordPointer> pairs instead of the full tuples (using a technique similar to an index sort), how does this affect the **I/O cost**?

**Concept Explanation:**

- The **two-pass sorting algorithm** requires enough memory to handle **B(S) / M** sorted runs in one merge pass.
- **Sorting only key-pointer pairs** reduces **I/O costs** since fewer data blocks are read/written.
- The **total I/O cost** consists of reading, writing, and merging the sorted runs.

---

**Question 3b: External Merge Sort with Large Relation**

We need to **sort relation T** using **external merge sort**. Given:

- **Relation T has 500,000 tuples**, i.e., **T(T) = 500,000**.
- **Block size** is **4,096 bytes**.
- Each **tuple** is **512 bytes**.
- **T is not clustered**, meaning tuples may not be packed efficiently in blocks.
- **The available main memory** has **500 blocks**.

**Questions:**

a) How many **passes** are required for sorting **T** using **external merge sort**?
b) What is the **total I/O cost**, including the merge phase?
c) If we store the **sorted relation as an index** instead of full tuples, how much **storage** is saved, and what is the **new I/O cost**?

**Concept Explanation:**

- The **number of passes** in external merge sort depends on the **initial sorted runs** and how many runs we can merge at once.
- The **total I/O cost** is computed by summing all **read/write operations** in each phase.
- Using **index-based sorting** (sorting keys instead of tuples) reduces **storage and I/O costs**.

---

**Question 3c: Optimizing External Sorting with Indexed Sorting**

We consider a scenario where we optimize sorting by only sorting **keys and pointers**.

- **Relation U has 1,000,000 tuples**, i.e., **T(U) = 1,000,000**.
- **Block size** = **8,192 bytes**.
- Each **tuple** is **1,024 bytes**.
- **Sort key** = **64 bytes**, and each **record pointer** is **8 bytes**.

- **The relation is clustered**.
- **Available main memory** has **1,000 blocks**.

**Questions:**

a) How many **sorted runs** are produced in the **initial pass** of external sorting?
b) If we only sort **<key, recordPointer>** pairs, what is the **new disk I/O cost**?
c) How does **sorting only index entries** instead of full tuples improve performance in DBMS query processing?

**Concept Explanation:**

- Sorting **only key-pointer pairs** significantly reduces the **amount of data written to disk**.
- The **merge process** then retrieves **tuples from disk only when needed**, reducing **random I/O operations**.
- This technique is often used in **B+ tree index sorting** for efficient **query lookups**.