Running Example
Two Relations R and S
T(R) = 60000
T(S) = 2000
Block size = 4000 bytes
S(R) = 100 bytes
S(S) = 500 bytes
Main Memory = 126 Buffers

Example **1a: Join Selection for Large Relations**

Two Relations R and S:

- T(R)=80,000, T(S)=3,000
- Block size = 4000 bytes, S(R)=120, S(S)=600 bytes
- Main Memory = 150 Buffers

Given the above details, what join algorithm would you suggest? Calculate the total number of I/O operations required.

Example **1b: Join Selection for Medium Relations**

Two Relations R and S:

- T(R)=30,000, T(S)=1,500, Block size = 4000 bytes, S(R)=80 bytes
- S(S)=400 bytes, Main Memory = 100 Buffers

Given the above details, what join algorithm would you suggest? Calculate the total number of I/O operations required.

Example **1c: Join Selection for Small Relations**

Two Relations R and S:

- T(R)=5,000, T(S)=500, Block size = 4000 bytes, S(R)=100 bytes, S(S)=500 bytes
- Main Memory = 50 Buffers

Given the above details, what join algorithm would you suggest? Calculate the total number of I/O operations required.

Explanation of Concepts in the Questions

- **Tuple Count T(R) and T(S)**: The number of tuples in each relation.
- **Block Size**: The size of each block in bytes.
- **Tuple Size (S(R) and S(S))**: The size of each tuple in bytes.
- **Main Memory Buffers**: The number of buffers available in main memory for processing.

These questions test understanding of:

- **Join Algorithm Selection**: Choosing the most efficient join algorithm (e.g., nested loop join, sort-merge join, hash join) based on the given parameters.
- **I/O Operations**: Calculating the total number of disk I/O operations required for the join.

Memory Requirement for Hash Join

Let's assume in our Hash join example from the last slide, **the number of main memory buffers available are only 6**. The rest of the specifications are the same. Now only 5 buffers are available for bucketizing.

Number of phases or passes : $\min(B(R), B(S)) \leq M^n$. ie $250 \leq 6^n$. This gives n = 4, ie we will need four passes to perform this join. The first three will be bucketizing and the last will be the join.

1. Phase 1 : Bucketizing
   a. R : Divide the 1500 blocks into 5 buckets, reading one block at a time, we get 300 blocks per bucket.
   b. S : Divide 250 blocks into 5 buckets, we get 50 blocks per bucket.
   c. Cost = 2 (B(R) + B(S)) = 3500
2. Phase 2 : re-Bucketizing
   a. R: Further divide each of the 5 buckets into another 5, size of new buckets = 300/5 = 60 blocks per new bucket.
   b. S : Further divide each of the 5 buckets into another 5, size of new buckets = 50/5 = 10 blocks per new bucket.
   c. Cost = 2 (B(R) + B(S)) = 3500
3. Phase 3 : re-re-Bucketizing
   a. R: new bucket size = 60/5 = 12 blocks / bucket.
   b. S: new bucket size = 10/5 = 2 blocks / bucket. **Finally we have a bucket small enough to fit in main memory with a block to spare!**
   c. Cost = 2 (B(R) + B(S)) = 3500
4. Phase 4 : Join
   a. O**nly Ss buckets can be kept in the memory** while the corresponding R Bucket is read one block at a time and joined with the whole of the S bucket.
   b. Cost = B(R) + B(S) = 1750

Total cost = 3x2x(B(R)+B(S)) + (B(R) + B(S)) = 3x3500+1750 = 12,250 IOs.

Question 1a: Hash Join with Limited Memory

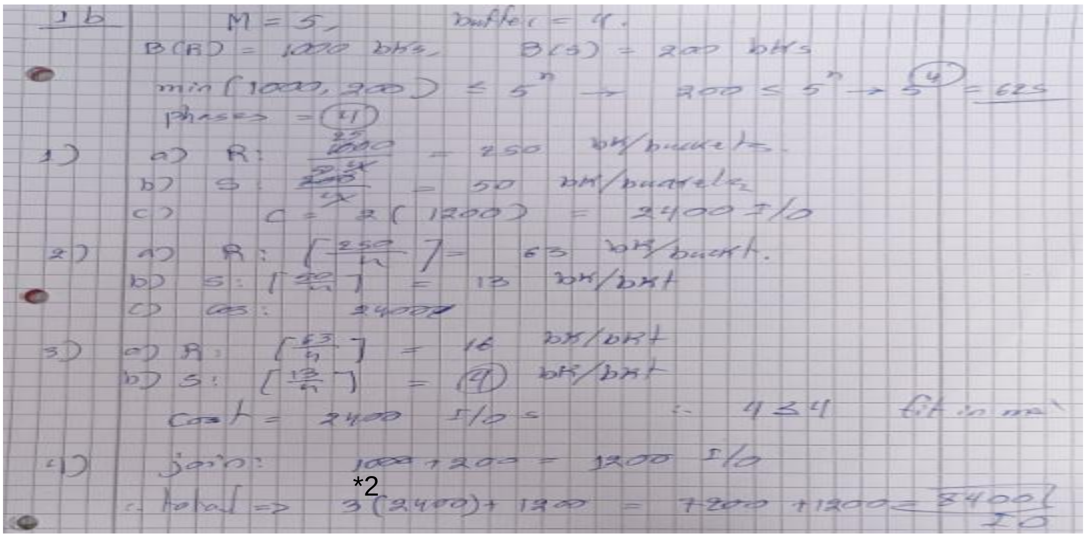
Let's assume in a Hash Join example, the number of main memory buffers available is only **8**. The rest of the specifications are as follows: B(R)=2000 blocks, B(S)=500 blocks, Number of buffers for bucketizing: **7**

a) Calculate the number of phases or passes required for the Hash Join.
b) Describe the bucketizing process for each phase and calculate the I/O cost for each phase.
c) What is the total I/O cost for the entire Hash Join operation?

Question 1b: Hash Join with Smaller Relations

Let's assume in a Hash Join example, the number of main memory buffers available is only **5**. The rest of the specifications are as follows: B(R)=1000 blocks, B(S)=200 blocks, Number of buffers for bucketizing: **4**

a) Calculate the number of phases or passes required for the Hash Join.
b) Describe the bucketizing process for each phase and calculate the I/O cost for each phase.
c) What is the total I/O cost for the entire Hash Join operation?



1b $M = 5$, buffer = 4.
$B(R) = 1000$ blks, $B(S) = 200$ blks
$\min(1000, 200) \leq 5^n \rightarrow 200 \leq 5^n \rightarrow 5^4 = 625$
phases = 4
1) a) R: $\frac{1000}{4} = 250$ blk/buckets.
b) S: $\frac{200}{4} = 50$ blk/buckets
c) $C = 2(1200) = 2400$ I/O
2) a) R: $\lceil \frac{250}{4} \rceil = 63$ blk/bucket.
b) S: $\lceil \frac{50}{4} \rceil = 13$ blk/bkt
c) cost: 2400
3) a) R: $\lceil \frac{63}{4} \rceil = 16$ blk/bkt
b) S: $\lceil \frac{13}{4} \rceil = 4$ blk/bkt
Cost = 2400 I/Os ∴ 4 ≤ 4 fit in mem
4) join: 1000 + 200 = 1200 I/O
*2
∴ total ⇒ 3(2400) + 1200 = 7200 + 1200 = 8400 IO

Question 1c: Hash Join with Larger Relations

Let’s assume in a Hash Join example, the number of main memory buffers available is only **10**. The rest of the specifications are as follows: $B(R)=3000$ blocks, $B(S)=1000$ blocks, Number of buffers for bucketizing: **9**

a) Calculate the number of phases or passes required for the Hash Join.
b) Describe the bucketizing process for each phase and calculate the I/O cost for each phase.
c) What is the total I/O cost for the entire Hash Join operation?

Explanation of Concepts in the Questions

1. **Hash Join**: A join algorithm that uses hashing to partition relations into smaller buckets that fit in memory.
2. **Phases or Passes**: The number of times data must be read/written to disk during bucketizing and joining.
3. **Bucketizing**: Dividing the relations into smaller buckets using a hash function.
4. **I/O Cost**: The total number of disk I/O operations required for the join, calculated as:
   - **Bucketizing Cost**: $2\times(B(R)+B(S))$ per phase.
   - **Join Cost**: $B(R)+B(S)$ for the final join phase.

Exercise 1

Relation R : B(R) = 10,000

Relation S : B(S) = 4000

Calculate the number of phases and the IOs for a Hash Join if R and S are contiguous for:

a) M = 100
b) M = 50
c) M = 30

Exercise 1a

Relation **R**: B(R)=12,000. Relation **S**: B(S)=5,000

Calculate the **number of phases** and the **I/O cost** for a **Hash Join** if **R and S are contiguous** for:
a) M=120, **b)** M=60, **c)** M=40

Exercise 1b

Relation **R**: B(R)=15,000 Relation **S**: B(S)=6,000

Compute the **number of phases** and the **I/O cost** for a **Hash Join** if **R and S are contiguous** for:
a) M=150, **b)** M=80, **c)** M=50

Exercise 1c

Relation **R**: B(R)=8,000 Relation **S**: B(S)=3,500

Determine the **number of phases** and the **I/O cost** for a **Hash Join** if **R and S are contiguous** for:
a) M=90 **b)** M=45 **c)** M=25

Exercise 2

Relation R : B(R) = 10,000

Relation S : B(S) = 4000

Calculate the number of phases and the IOs for a Merge Join if R and S are contiguous. Optimize whenever possible.

a) M = 101
b) M = 50
c) M = 30

Exercise 2a

Relation **R**: B(R)=12,000 Relation **S**: B(S)=5,000

Calculate the **number of phases** and the **I/O cost** for a **Merge Join** if **R and S are contiguous**. **Optimize whenever possible**.

a) M=120 **b)** M=60 **c)** M=40

Exercise 2b

Relation **R**: B(R)=15,000 Relation **S**: B(S)=6,000

Calculate the **number of phases** and the **I/O cost** for a **Merge Join** if **R and S are contiguous**. **Optimize whenever possible**.

a) M=150 **b)** M=80 **c)** M=50

Exercise 2c

Relation **R**: B(R)=8,000 Relation **S**: B(S)=3,500

Calculate the **number of phases** and the **I/O cost** for a **Merge Join** if **R and S are contiguous**. **Optimize whenever possible**.

a) M=90 **b)** M=45 **c)** M=25

Exercise 3

Relation R : B(R) = 10,000, T(R) = 100,000

Relation S : B(S) = 4000, T(S) = 32,000

M = 100. Calculate IOs:

a) for Hash Join, if the relations are not contiguous
b) for Hash Join, if the relations are sorted
c) for Merge Join, if the relations are sorted
d) And Minimum memory requirement for One Pass Join if the relations are Contiguous.

Exercise 3a

Relation **R**: B(R)=12,000, T(R)=120,000. Relation. **S**: B(S)=5,000, T(S)=40,000 Available **main memory blocks**: M=120

Calculate the I/O costs for:
a) Hash Join when the relations are **not contiguous**.
b) Hash Join when the relations are **sorted**.
c) Merge Join when the relations are **sorted**.
d) The **minimum memory requirement** for **One-Pass Join** when the relations are **contiguous**.

Exercise 3b

Relation **R**: B(R)=15,000, T(R)=150,000. Relation **S**: B(S)=6,000, T(S)=48,000. Available **main memory blocks**: M=150

Calculate the I/O costs for:
a) Hash Join if the relations are **not contiguous**.
b) Hash Join if the relations are **already sorted**.
c) Merge Join if both relations are **sorted**.
d) The **minimum memory requirement** for a **One-Pass Join**, assuming **contiguous storage**.

Exercise 3c

Relation **R**: B(R)=8,000, T(R)=80,000. Relation. **S**: B(S)=3,500, T(S)=28,000. Available **main memory blocks**: M=80

Compute the I/O costs for:
a) Hash Join if the relations are **not stored contiguously**.
b) Hash Join if both relations are **sorted**.
c) Merge Join assuming the relations are **sorted**.
d) The **minimum memory requirement** for a **One-Pass Join** when the relations are **contiguous**.

Joins: Iteration vs. Hash vs. Merge

Joins are a fundamental operation in relational databases, used to combine records from two or more tables based on a common key. There are multiple ways to implement joins efficiently, depending on the **size of the tables**, **availability of indexes**, and **amount of memory available**.

The three major types of join algorithms are:

1. **Nested Loop Join (Iteration-based Join)**
2. **Hash Join**
3. **Merge Join (Sort-Merge Join)**

---

1. Nested Loop Join (Iteration-based Join)

Concept:

This is the most basic way to perform a join. It involves iterating through one relation (outer loop) and, for each row, scanning the entire second relation (inner loop) to find matching tuples.

Types:

- **Brute Force Nested Loop:** No indexing, scanning every combination of tuples.
- **Indexed Nested Loop:** Uses an index on the inner relation to speed up lookups.

Time Complexity:

- **Without Index:** $O(N \times M)$ (where N and M are the sizes of the two relations)
- **With Index:** $O(N \times \log M)$ (if an index exists on the inner relation)

Example:

Consider two tables:

sql Kopieren Bearbeiten

```sql
SELECT *
FROM Employees E, Departments D
WHERE E.DepartmentID = D.DepartmentID;
```

- **Brute Force:** For each employee, scan all departments.
- **With Index:** If `DepartmentID` in `Departments` is indexed, lookup is much faster.

I/O Cost:

For relations **R** (outer) and **S** (inner), assuming:

- **B(R) = 10,000** blocks
- **B(S) = 4,000** blocks
- **Memory = M = 100** blocks

Brute Force Nested Loop I/O Cost:

$$B(R) \times B(S) = 10,000 \times 4,000 = 40,000,000 \text{ I/Os}$$

Indexed Nested Loop I/O Cost (if index exists on S):

$$B(R) + (T(R) \times \text{Index Cost})$$

Pros & Cons:

✅ Simple to implement
✅ Works without sorting or hashing
❌ Very slow for large datasets
❌ Not efficient unless indexes exist

---

2. Hash Join

Concept:

Uses **hashing** to divide records into buckets and then joins corresponding buckets.

Phases:

1. **Partition Phase:**
   - Use a hash function $h(x)$ to distribute both relations into **buckets** stored in memory/disk.
2. **Join Phase:**
   - Match records in the same bucket.

Time Complexity:

$$O(N + M)$$

(Better than Nested Loop!)

Example:

sql Kopieren Bearbeiten

```sql
SELECT *
FROM Orders O, Customers C
WHERE O.CustomerID = C.CustomerID;
```

- A **hash function** is applied to `CustomerID` to partition both `Orders` and `Customers` into buckets.
- Matching is only done **within buckets**, reducing comparisons.

I/O Cost:

For **R (10,000 blocks)** and **S (4,000 blocks)**:

- Partitioning: $B(R) + B(S) = 10,000 + 4,000 = 14,000$
- Probing: $B(R) + B(S) = 14,000$

Total I/Os = 28,000

Pros & Cons:

✅ **Faster than Nested Loop**, especially for large datasets
✅ **Does not require sorting**
❌ Needs **more memory**
❌ Performance drops if hash function creates skewed buckets

---

3. Merge Join (Sort-Merge Join)

Concept:

Sorts both relations on the join key and then **merges them in a single pass**.

Phases:

1. **Sorting Phase:**
   - Sort both relations on the join key.
2. **Merge Phase:**
   - Scan both sorted relations and match tuples.

Time Complexity:

$$O(N \log N + M \log M)$$

(because of sorting)

$$+O(N + M)$$

(for merging)

Example:

sql Kopieren Bearbeiten

```sql
SELECT *
FROM Students S, Enrollments E
WHERE S.StudentID = E.StudentID;
```

- Both **Students** and **Enrollments** are **sorted** on `StudentID`.
- They are **merged efficiently** by scanning both tables **in order**.

I/O Cost:

For **R (10,000 blocks)** and **S (4,000 blocks)**:

- Sorting: $2B(R)\log_M B(R) + 2B(S)\log_M B(S)$
- Merging: $B(R) + B(S) = 14,000$

Total I/Os ≈ 30,000 (depends on sorting efficiency)

Pros & Cons:

✅ **Best when relations are already sorted**
✅ **Efficient for large datasets**
✅ **Works well for range queries**
❌ **Expensive if sorting is required**
❌ **Needs enough memory to hold sorted runs**

Comparison Table

Join Type	Best Use Case	Time Complexity	Memory Required	I/O Cost
Nested Loop	Small tables or indexed joins	$O(N \times M)$	Low	Very high
Hash Join	Large tables, equality joins	$O(N + M)$	Medium to High	Medium
Merge Join	Already sorted data, range joins	$O(N \log N + M \log M)$	Medium	Medium

---

When to Use Each Join Type?

- **Use Nested Loop** if relations are small or indexes exist.
- **Use Hash Join** for **large, unsorted** tables when equality conditions are used.
- **Use Merge Join** when both relations **are sorted** (or sorting is acceptable).

Conclusion

Joins are a **critical operation in query processing**, and choosing the right algorithm is crucial for performance.

- **Nested Loop** is simple but inefficient.
- **Hash Join** is fast for large, unsorted data.
- **Merge Join** is great for **sorted relations**.