Query Processing

Introduction to Database Management

CS348 Fall 2022

Announcements (Tue., Nov 08)

- Project
  - Milestone 1 Reach your assigned TA for grading remark (cc Xi and Glaucia)
  - Milestone 2 due **Nov 17** (Thu)
  - Final demo in the week of **Nov 25th – Dec 1st (Week 13)**
    - Email your TA the choice of your demo (online/video) **by Nov 24**
    - Lose points if failing to do so
    - No lecture in that week
  - Final report is due Dec 1st (Thu)

- Assignment 3
  - Cover Lectures 11-15
  - Due Nov 24 (Thu)

Overview

- Many different ways of processing the same query
  - Scan? Sort? Hash? Use an index?
  - All have different performance characteristics and/or make different assumptions about data
- Best choice depends on the situation
  - Implement all alternatives
  - Let the query optimizer choose at run-time (next lecture)

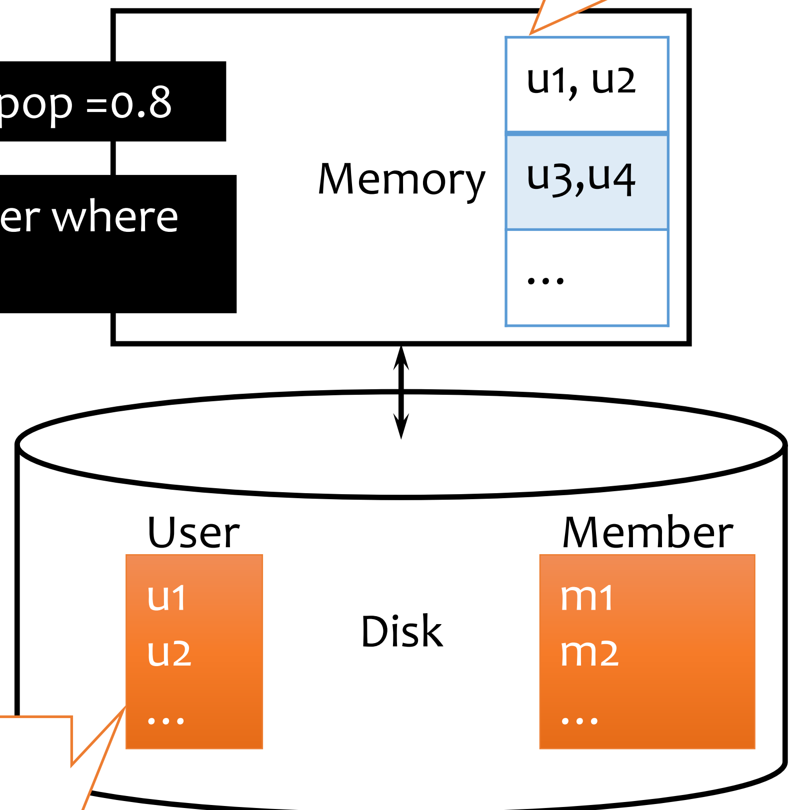
Outline

- Scan
- Index
- Sort (Optional)
- Hash (Optional)

```
select * from User where pop =0.8
```

```
select * from User, Member where
User.uid = Member.uid;
```

Number of memory blocks available: M



Number of rows for a table $|Users|$
Number of disk blocks for a table

$$B(Users) = \frac{|Users|}{\#\ of\ rows\ per\ block}$$

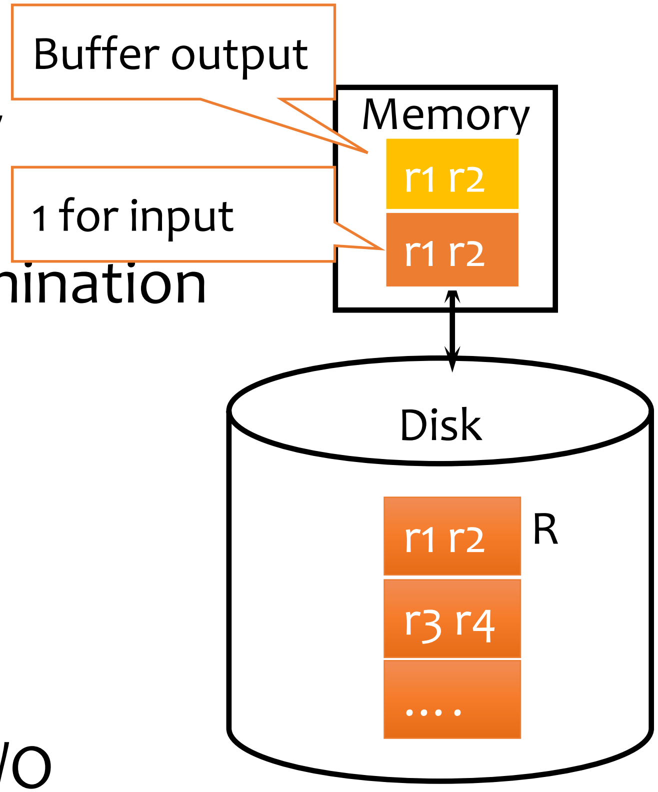
Notation

- Relations: R, S
- Tuples: r, s
- Number of tuples: $|R|$, $|S|$
- Number of disk blocks: $B(R)$, $B(S)$
- Number of memory blocks available: M
- Cost metric
    - Number of I/O's
    - Memory requirement

Scanning-based algorithms

Table scan

- Scan table *R* and process the query
  - Selection over *R*
  - Projection of *R* without duplicate elimination
- I/O's: $B(R)$
  - Trick for selection:
    - stop early if it is a lookup by key
- Memory requirement: 2 (blocks)
  - 1 for input, 1 for buffer output
  - Increase memory does not improve I/O
- Not counting the cost of writing the result out
  - Same for any algorithm!
  - Maybe not needed—results may be pipelined into another operator



Nested-loop join

$R \bowtie_p S$

- For each block of R, and for each r in the block:
  For each block of S, and for each s in the block:
  Output rs if p evaluates to true over r and s
  - R is called the outer table; S is called the inner table
  - I/O's: $B(R) + |R| \cdot B(S)$

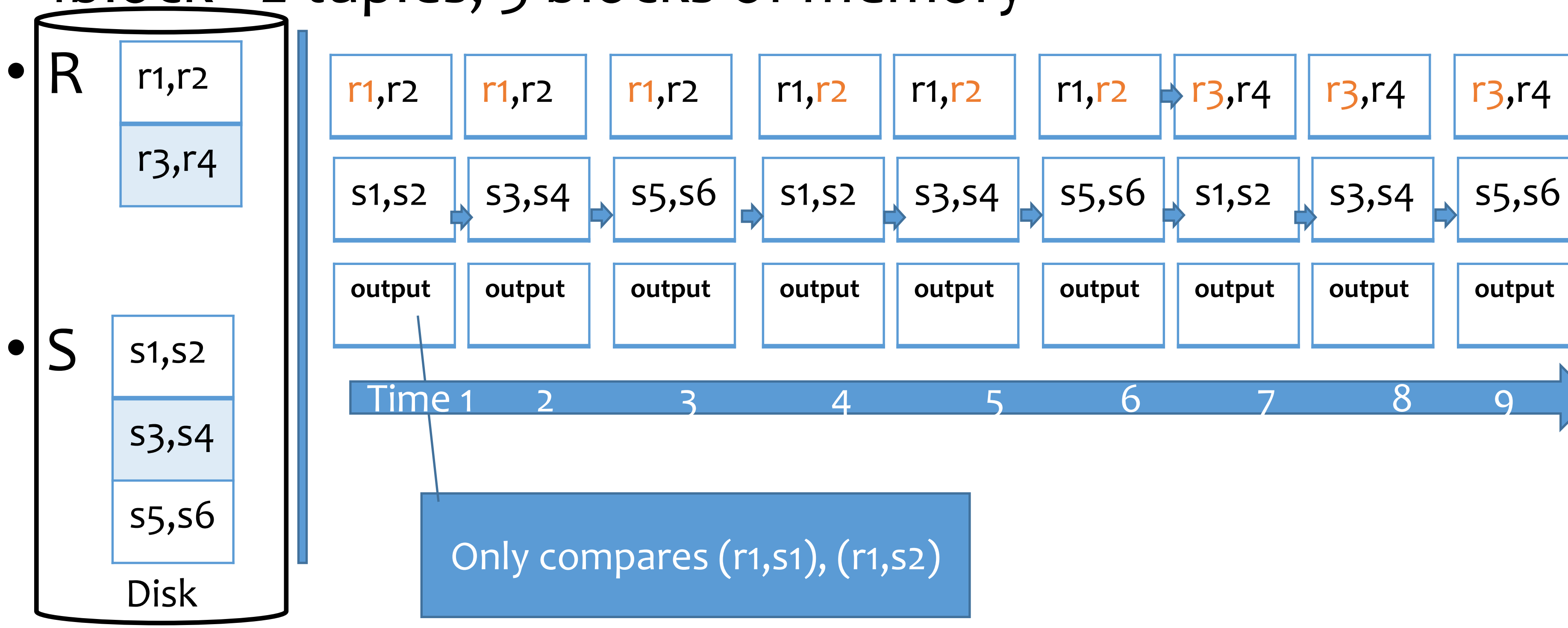
Blocks of R are moved into memory only once

Blocks of S are moved into memory with |R| number of times

  - Memory requirement: 3

Example for basic nested loop join

- 1block =2 tuples, 3 blocks of memory



- Number of I/O:

$$B(R) + |R| * S(R) = 2 \text{ blocks} + 4 * 3\text{blocks} = 14$$

Nested-loop join

$R \bowtie_p S$

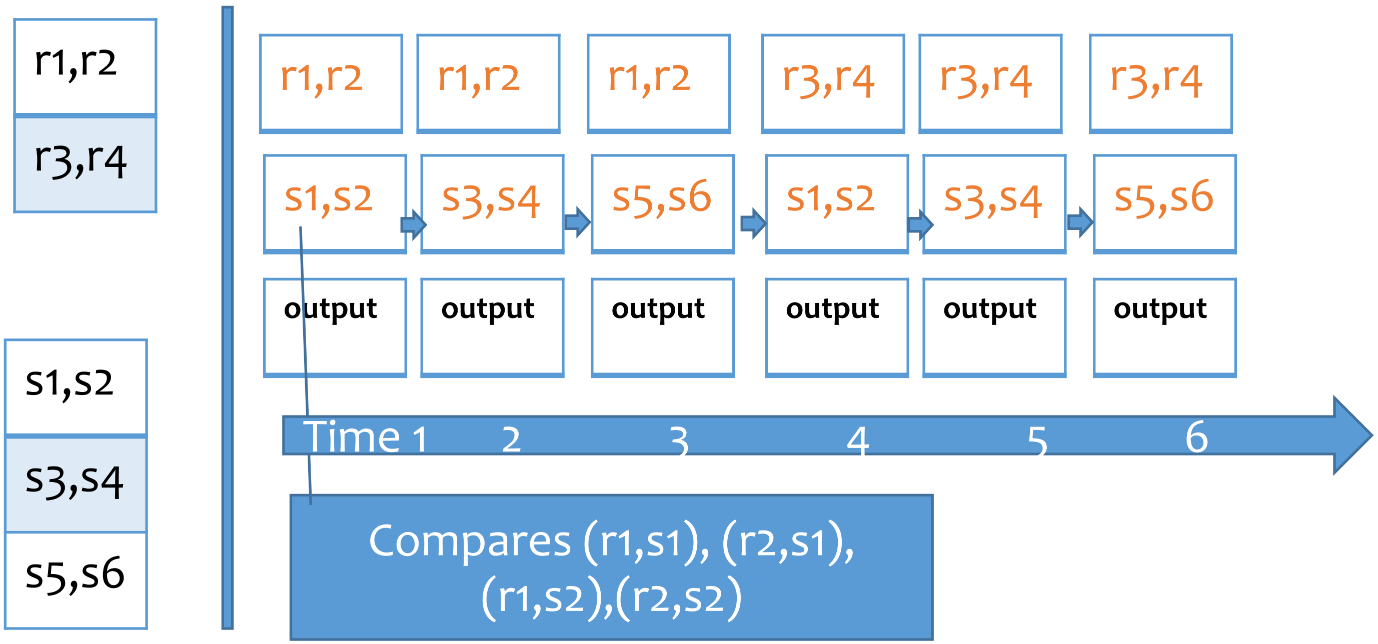
- For each block of R, and for each r in the block:
  For each block of S, and for each s in the block:
  Output rs if p evaluates to true over r and s
  - R is called the outer table; S is called the inner table
  - I/O's: $B(R) + |R| \cdot B(S)$
  - Memory requirement: 3

Improvement: block-based nested-loop join

- For each block of R, for each block of S:
  For each r in the R block, for each s in the S block: …
  - I/O's: $B(R) + B(R) \cdot B(S)$
  - Memory requirement: same as before

Example for block-based nested loop join

- 1block =2 tuples, 3 blocks of memory
- R
- S



- Number of I/O:

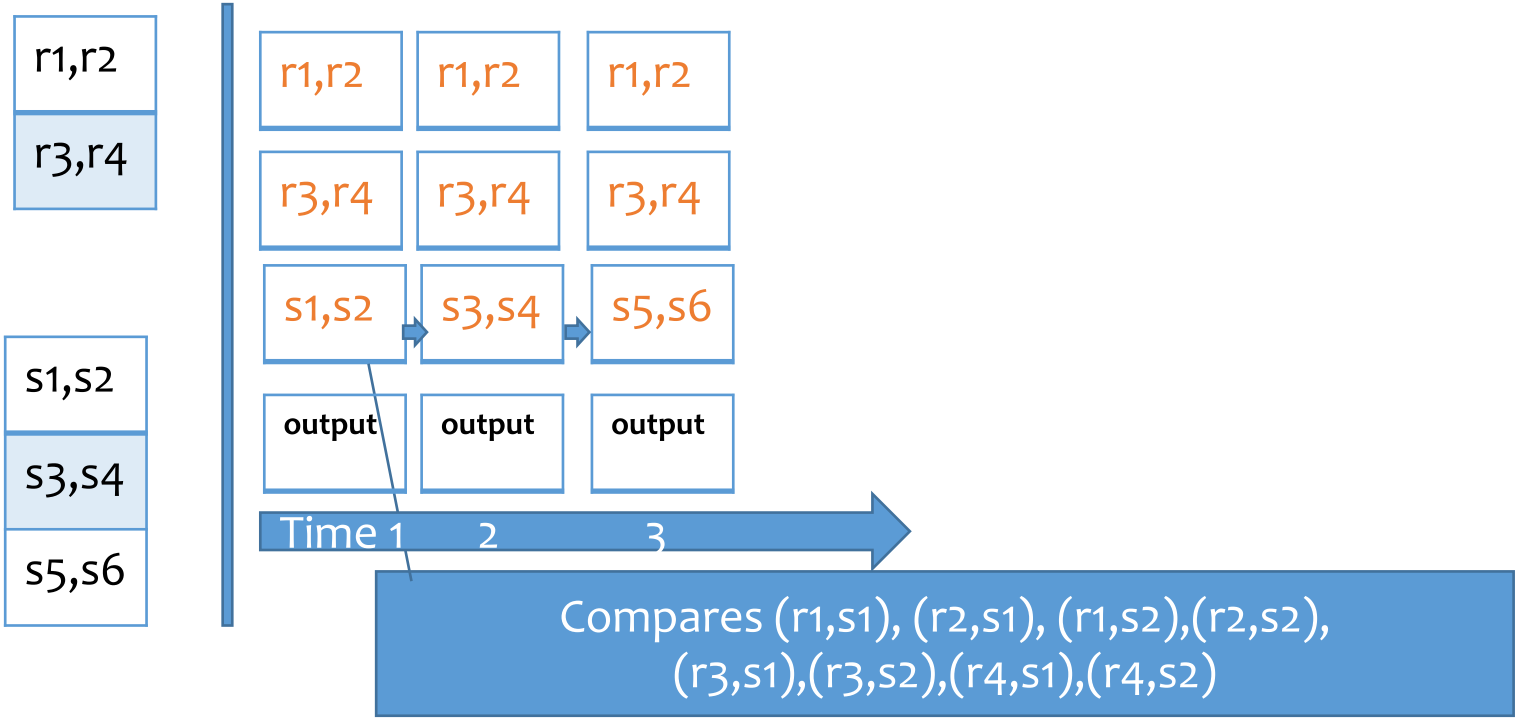
  B(R) + B(R)* B(S) = 2 blocks + 2 * 3blocks = 8

More improvements

- Stop early if the key of the inner table is being matched
- Make use of available memory
  - Stuff memory with as much of R as possible, stream S by, and join every S tuple with all R tuples in memory
  - I/O's: $B(R) + \left\lceil \frac{B(R)}{M-2} \right\rceil \cdot B(S)$
    - Or, roughly: $B(R) \cdot B(S)/M$
  - Memory requirement: M (as much as possible)
- Which table would you pick as the outer? (exercise)

Example for block-based nested loop join

- 1block =2 tuples, 4 blocks of memory
- R
- S



- Number of I/O:
B(R) + B(R)/(M-2)* S(R) = 2 blocks + 1 * 3blocks = 5

Case study:

- System requirements:
  - Each disk/memory block can hold up to 10 rows (from any table);
  - All tables are stored compactly on disk (10 rows per block);
  - 8 memory blocks are available for query processing: M=8
- Database:
  - User(uid, age, pop), Member(gid,uid,date), Group(gid, gname)
  - |User|=1000 rows, |Group|=100 rows, |Member|=50000 rows
  - #of blocks: B(User)=1000/10=100; B(Group)=100/10=10; B(Member)=50000/10=5k
- Q1: select * from User where pop =0.8
  - I/O cost using table scan? $B(User) = 100$ (slide 7)
- Q2: select * from User, Member where User.uid = Member.uid;
  - I/O cost using blocked-based nested loop join (slide 12)

$$B(User) + \left\lceil \frac{B(User)}{M-2} \right\rceil \cdot B(Member) = 100 + \left\lceil \frac{100}{8-2} \right\rceil \cdot 5000$$

Outline

- Scan
  - Selection, duplicate-preserving projection, nested-loop join
- Index
- Sort (Optional)
- Hash (Optional)

Index-based algorithms

http://i1.trekearth.com/photos/28820/p2270994.jpg

Selection using index

- Equality predicate: $\sigma_{A=v}(R)$
  - Use an ISAM, B^+-tree, or hash index on $R(A)$
- Range predicate: $\sigma_{A>v}(R)$
  - Use an ordered index (e.g., ISAM or B^+-tree) on $R(A)$
  - Hash index is not applicable

- Indexes other than those on $R(A)$ may be useful
  - Example: B^+-tree index on $R(A, B)$
  - How about B^+-tree index on $R(B, A)$?

Index versus table scan

Situations where index clearly wins:

- Index-only queries which do not require retrieving actual tuples
  - Example: $\pi_A\left(\sigma_{A>v}(R)\right)$
- Primary index clustered according to search key
  - One lookup leads to all result tuples in their entirety

Index versus table scan (cont'd)

BUT(!):

- Consider $\sigma_{A>v}(R)$ and a secondary, non-clustered index on $R(A)$
  - Need to follow pointers to get the actual result tuples
  - Say that 20% of R satisfies $A > v$
    - Could happen even for equality predicates
  - I/O's for scan-based selection: $B(R)$
  - I/O's for index-based selection: lookup + 20% $|R|$
  - Table scan wins if a block contains more than 5 tuples!
    - B(R) = |R|/5 < 20%|R|+lookup

Index nested-loop join

$R \bowtie_{R.A=S.B} S$

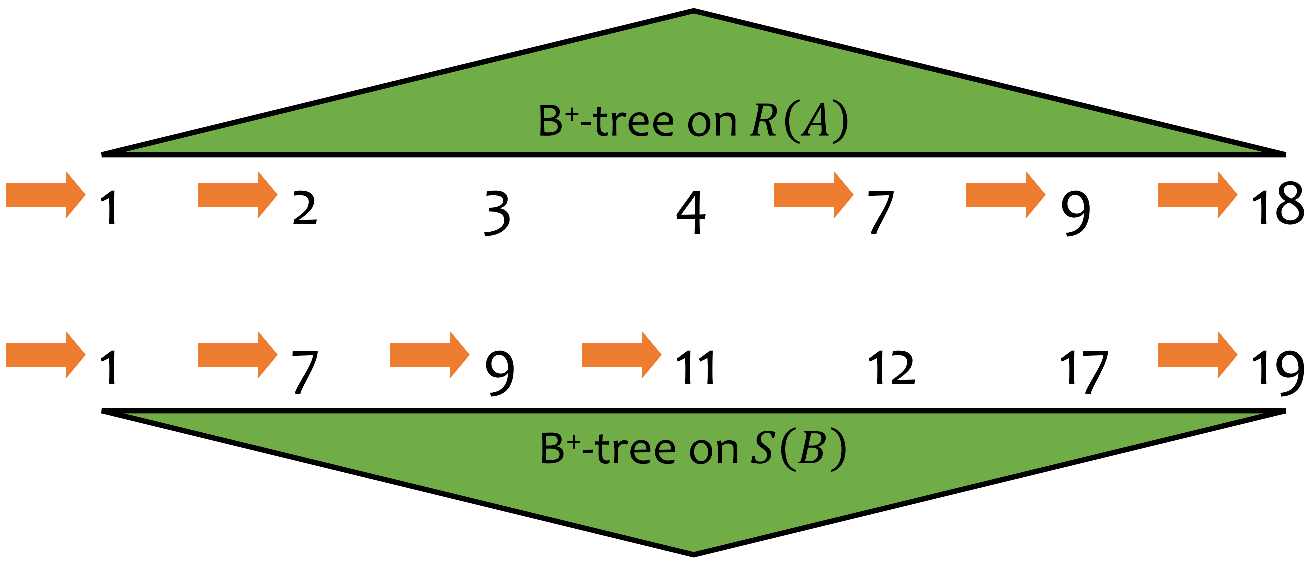
- Idea: use a value of $R.A$ to probe the index on $S(B)$
- For each block of R, and for each r in the block:
  Use the index on $S(B)$ to retrieve s with $s.B = r.A$
  Output rs

- I/O's: $B(R) + |R| \cdot (\text{index lookup})$
  - Typically, the cost of an index lookup is 2-4 I/O's (depending on the index tree height if B+ tree)
  - Beats other join methods if $|R|$ is not too big
  - Better pick R to be the smaller relation
- Memory requirement: 3 (extra memory can be used to cache index, e.g. root of B+ tree).

Zig-zag join using ordered indexes (Optional)

$R \bowtie_{R.A=S.B} S$

- Idea: use the ordering provided by the indexes on $R(A)$ and $S(B)$ to eliminate the sorting step of sort-merge join
- Use the larger key to probe the other index
  - Possibly skipping many keys that don't match



Outline

- Scan
  - Selection, duplicate-preserving projection, nested-loop join
- Index
  - Selection, index nested-loop join, zig-zag join
- Sort (Optional)
- Hash (Optional)

Optional (won't be tested)

Another view of techniques

- Selection
  - Scan without index (linear search): $O(B(R))$
  - Scan with index – selection condition must be on search-key of index
    - B+ index: $O(\log(B(R))$
    - Hash index: $O(1)$
- Projection
  - Without duplicate elimination: $O(B(R))$
  - With duplicate elimination
    - Sorting-based: $O\left(B(R) \cdot \log_M B(R)\right)$
    - Hash-based: $O(B(R) + t)$where t is the result of the hashing phase
- Join
  - Block-based nested loop join (scan table): $O(B(R) \cdot \frac{B(S)}{M})$
  - Index nested loop join $O(B(R) + |R| \cdot (\text{index lookup}))$
  - Sort-merge join $O\left(B(R) \cdot \log_M B(R) + B(S) \cdot \log_M B(S)\right)$
  - Hash join $O\left(B(R) \cdot \log_M B(R) + B(S) \cdot \log_M B(S)\right)$