Query Processing

Given a query,

```
SELECT B.manufacturer
FROM Beers B, Sells S
WHERE B.name = S.beer
  AND S.price < 20
```

We execute it in two levels:
① Logical Plan : Relational Alg, many variations exist
② Physical Plan : Implementation of Logical Plan
↳ Can be extended

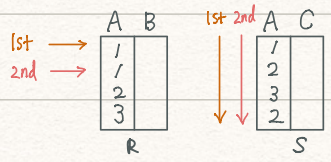
Implementations of JOIN Operations

↳ Dominant cost is I/O access ① Logical Level: $R \bowtie S$
② Physical Level:

Internal Memory Join

1. Nested Loop Join

↳ If both R and S can fit into the main memory

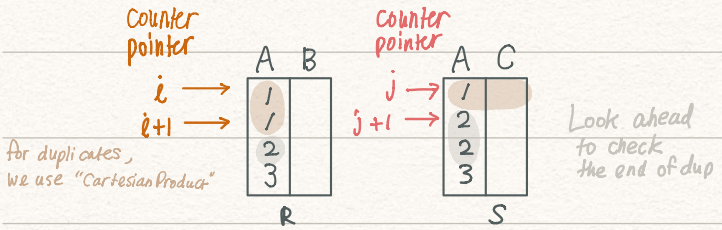


Basically, for i in R(A)
  for j in S(A)
Nested for Loop

↳ Cost: $B(R) + B(S)$

2. Sort-Merge Join

↳ If both R and S can fit into the main memory



Since R(A) and S(A) are sorted, we can search and match.

↳ Cost: $B(R) + B(S)$

3. Hash Join

Hash table based on the join attribute
Then we can just scan the bigger relation

R ⋈ S Algorithm:

```
Look up h(v) of v in the hash table
Fetch the value
JOIN!   scan of R or S & Probe the hash table
```

↳ Cost: $B(R) + B(S)$

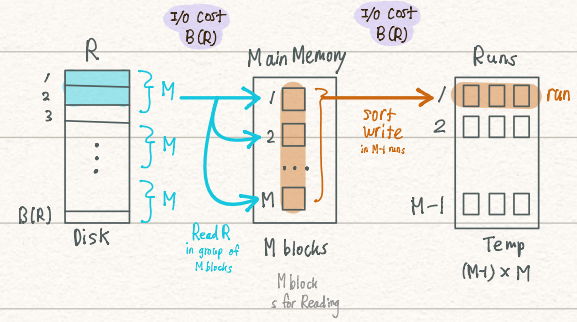
cf) Sorting

External Memory Sort

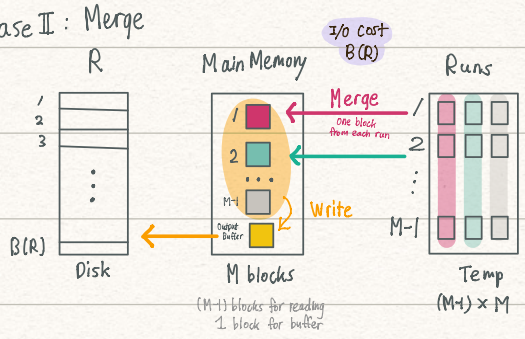
Two Pass (Phase) Multiway Merge Sort

↳ At least one relation doesn't fit into main memory.

• Phase I: Read & Sort



• Phase II: Merge



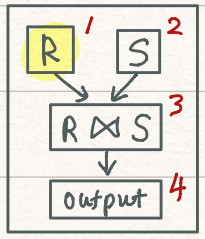
↳ Requirement : $B(R) \leq M(M-1)$
or $B(R) \leq M^2$

↳ cost: $2B(R)$ in Phase I + $B(R)$ in Phase II → $3B(R)$

External Memory Join

1. Nested Loop Join

Memory Block



↳ cost: $B(R) + |R| B(S)$

Improved!

Read blockwise → $B(R) + B(R) \cdot B(S)$

↳ Requirement: 4

Improved w/ bigger M

If M = 200 (large), You can fill 198 blocks w/ R and join w/ S

$$B(R) + \left[\frac{B(R)}{M-2}\right] B(S) \approx \frac{B(R)B(S)}{M}$$

↳ Requirement: M

↳ M-2 because 2 blocks for output

3. Partitioned Hash Join

Hash S into M buckets, R into M buckets → Join corresponding buckets

② send all buckets to disk

↳ Cost: $3B(R) + 3B(S)$

↳ Requirement: the smaller bucket fits in memory

$$\frac{\min(B(R), B(S))}{M-1} \leq M$$

$$\min(B(R), B(S)) \leq M^2$$

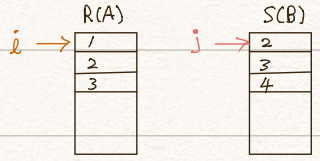
✱ What if a partition is larger than memory,
→ Recursive partioning

2. Sort Merge Join

Sort R and S using Two pass Multiway SM

Join Attribute

- Designate a pointer to the 1st tuple



If r.A > s.B
  s: Next tuple in S(B)
Else if r.A < s.B
  R: Next tuple in R(A)
Else
  Return matching tuples
  R,S: Next tuples in S(B),R(A)

If # of matching tuples is too large to fit in M,
↳ Use "Nested Loop Join"

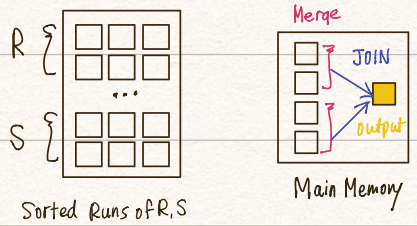
↳ Cost = $Cost_{Sort} + 2B(R) + 2B(S)$

Two Pass Sorting: $3B(R) + 3B(S)$
$+ 1B(R) + 1B(S)$ ⇝ Reading R,S for Merge
Merge: $B(R) + B(S)$

$5B(R) + 5B(S)$

↳ Requirement: $B(R) \leq M^2$, $B(S) \leq M^2$

✱ Optimization: Combine JOIN w/ Merge



↳ $|run(R) + run(S)| < M-1$

The # of runs for R & S should fit in M-1

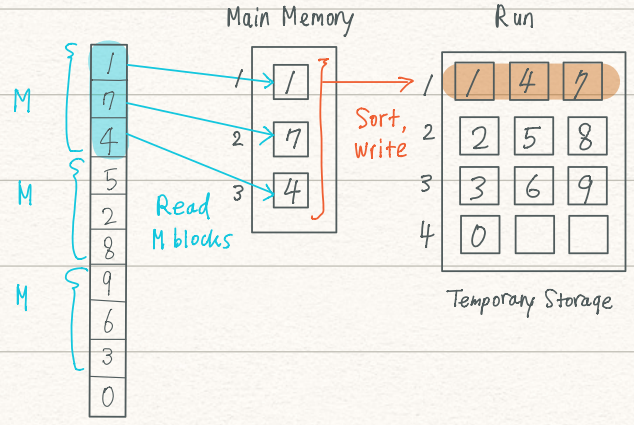
↳ cost: $3B(R) + 3B(S)$

↳ Requirement: $B(R) + B(S) \leq M^2$

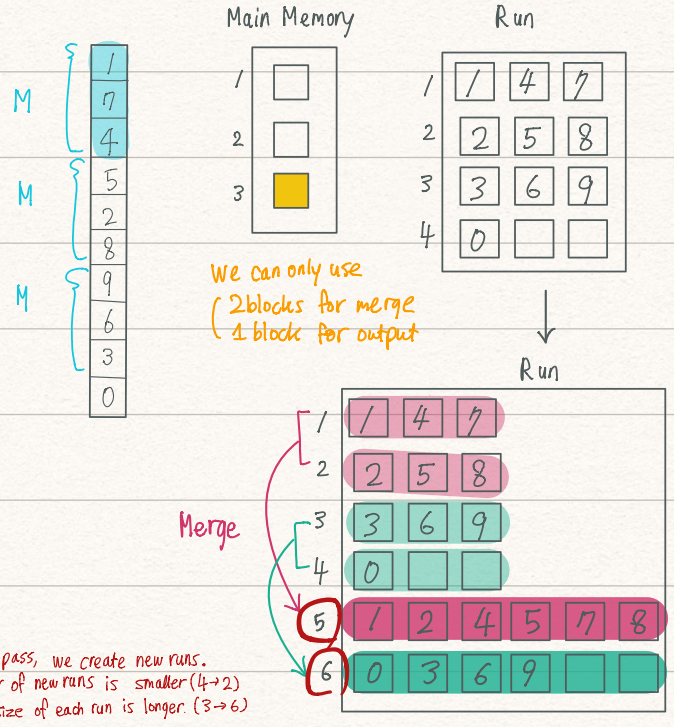
General Multipass Multiway Merge Sort

ex> M : 3 blocks, each block holding one number

- Phase I (Pass 0) : Read & Sort



- Phase II (Pass 1) : Merge



- Phase III (Pass 2) : Merge

Can handle more than $B(R) > M^2$

The key to apply Multipass Multiway Merge Sort is

① For each pass, merge as much as you can w/ given M

② Create new runs (Merge & Sort) w/ each run # runs ↓, |runs| ↑

③ Finish ② when everything is sorted.

④ Return the one sorted relation.

Generalization of MMMS :

- Pass 0 : Read a relation, R, in group of M blocks & Sort them
  - ↳ $\lceil \frac{B(R)}{M} \rceil$ runs in level 0.
- Pass k : Merge (M-1) runs in level(k-1) & write them in level k
  - why? (M-1) blocks for input, 1 block for output
  - ↳ In level k, the number of runs = $\frac{\text{The number of runs in level (k-1)}}{(M-1)}$
- Final Pass : results in one sorted run.

Analysis of MMMS :

- # of Passes : $\lceil \log_{M-1} \lceil \frac{B(R)}{M} \rceil \rceil + 1$ (pass 0 for reading R)
- Cost
  - In each pass, we read and write R
    - ↳ # of passes · 2B(R)
  - In the final pass, we simply write, so disregard
  - ∴ cost of MMMS = $O(B(R) \cdot \log_M B(R))$
- Memory Requirement : M