Perfect 🙌 You've set up a **running example** with:

- $T(R) = 60,000$ tuples
- $T(S) = 2,000$ tuples
- Block size = 4,000 bytes
- Size of R tuple = 100 bytes → 40 tuples per block → $B(R) = 1500$ blocks
- Size of S tuple = 500 bytes → 8 tuples per block → $B(S) = 250$ blocks
- Main memory = 126 buffers

Now let's carefully **explain each improvement step** in the nested-loop join using these numbers.

🟦 Running Example Recap

Share

- $T(R) = 60,000$ tuples → $B(R) = 1500$ blocks
- $T(S) = 2000$ tuples → $B(S) = 250$ blocks
- Memory = 126 buffers → 125 for outer relation, 1 for inner relation

---

🟩 Cost Comparison Table

Method	Assumption	Strategy	Cost Formula	Cost with R=60,000, S=2000
Naïve Nested Loop	Non-contiguous	For each tuple of R, scan all of S	$T(R) \times T(S)$	$60,000 \times 2000 = 120,000,000$ I/Os ❌
Iteration Join (Case 1)	Non-contiguous	Chunk R (5000 tuples at a time), scan all of S each time	$T(R) + \frac{T(R)}{5000} \times T(S)$	$60,000 + 12 \times 2000 = 84,000$ I/Os ✅
Iteration Join (Case 2)	Contiguous	Chunk R (125 blocks at a time), scan all of S each time	$B(R) + \frac{B(R)}{125} \times B(S)$	$1500 + 12 \times 250 = 4500$ I/Os ✅✅
Iteration Join (Case 3a)	Non-contiguous	Chunk S (1000 tuples at a time), scan all of R each time	$T(S) + \frac{T(S)}{1000} \times T(R)$	$2000 + 2 \times 60,000 = 122,000$ I/Os ❌
Iteration Join (Case 3b)	Contiguous	Chunk S (125 blocks at a time), scan all of R each time	$B(S) + \frac{B(S)}{125} \times B(R)$	$125 + 2 \times 1500 = 3250$ I/Os ✅✅✅

🔑 Insights

1. **Naïve nested loop** → prohibitively expensive (120M I/Os).
2. **Iteration join with R as outer** → improves a lot (down to 84K or 4.5K depending on contiguity).
3. **Iteration join with S as outer** → bad for non-contiguous (122K), **but best** when contiguous (3250 I/Os).
4. **Rule of Thumb**: Always choose the **smaller relation (S)** as outer loop if data is contiguous.

🔹 Case 1: R and S Not Contiguous (tuple-by-tuple I/O)

Here, records are not stored contiguously, so **1 I/O fetches only 1 tuple**.

1. 1 buffer = 40 R records
2. 125 buffers = $125 \times 40 = 5000$ R records at a time in memory
3. Use the last buffer to read **1 tuple of S** at a time
4. Join these 5000 R records with all S tuples (2000 tuples)

Since R has 60,000 tuples → divide into $60,000/5000 = 12$ chunks

Cost:

- Read all R once → $T(R) = 60,000$ I/Os
- For each of the 12 chunks, scan all of S (2000 tuples) → $12 \times 2000 = 24,000$ I/Os
- ✅ Total = **84,000 I/Os**

👉 Much better than naïve $T(R) \times T(S) = 120M$.

🔹 Case 2: R and S Contiguous (block-by-block I/O)

Now tuples are stored contiguously, so **1 I/O fetches an entire block**.

- $S(R) = 100$ bytes → 40 tuples/block → $B(R) = 1500$ blocks
- $S(S) = 500$ bytes → 8 tuples/block → $B(S) = 250$ blocks

1. Load 125 blocks of R into memory → $125 \times 40 = 5000$ records at once
2. Keep these in memory, then scan S block by block (250 blocks)
3. Repeat this 12 times (since $1500/125 = 12$)

Cost:

- Read R once → $B(R) = 1500$ I/Os
- For each of 12 chunks of R, scan all S → $12 \times 250 = 3000$ I/Os
- ✅ Total = **4500 I/Os**

👉 Huge reduction compared to 84,000, thanks to contiguous block access.

🔹 Case 3: Chunking S Instead of R

Now we flip the strategy: load chunks of S into memory, and scan R multiple times.

Case 3a: Non-Contiguous

- 125 buffers hold **1000 S tuples**
- Fetching 1000 S tuples costs **1000 I/Os** (since non-contiguous)
- For each chunk of S, read all of R (60,000 tuples = 60,000 I/Os)
- Need 2 chunks (2000/1000 = 2)

Cost:

$$T(S) + 2 \times T(R) = 2000 + 120,000 = 122,000\,\mathrm{I/Os}$$

❌ Worse than earlier 84,000

Case 3b: Contiguous

- 125 buffers hold **1000 S tuples**, but now contiguous = **125 I/Os**
- For each chunk of S, scan all of R = 1500 I/Os
- Need 2 chunks

Cost:

$$B(S) + 2 \times B(R) = 125 + 3000 = 3250\,\mathrm{I/Os}$$

✅ Better than 4500 in Case 2

🔑 Key Takeaways

1. **Iteration Join (BNLJ)** = Load as many blocks of one relation as possible into memory, then scan the other relation.
2. **Contiguity is critical**: fetching blocks is much cheaper than fetching tuples individually.
3. **Best strategy depends** on data layout:
   - If relations are **non-contiguous** → chunk **R** (outer relation) → 84,000 I/Os
   - If relations are **contiguous** → chunk **S** (inner relation) → 3250 I/Os

---

1) Given setup (derive all the pieces)

- Tuples per relation
  $T(R) = 60{,}000,\ T(S) = 2{,}000$. These are **tuple counts** (not blocks).
- Block size and tuple sizes
  Block size $= 4{,}000$ bytes, tuple sizes: $S(R) = 100$ bytes, $S(S) = 500$ bytes.
- Records per block
  $\dfrac{4{,}000}{100} = 40$ tuples of R per block; $\dfrac{4{,}000}{500} = 8$ tuples of S per block.
- Blocks per relation (contiguous storage case)
  $B(R) = \frac{T(R)}{40} = \frac{60{,}000}{40} = 1{,}500$ blocks,
  $B(S) = \frac{T(S)}{8} = \frac{2{,}000}{8} = 250$ blocks.
- Memory buffers: $M = 126$. We'll often "reserve" 1 buffer for the stream we're scanning and use the rest to hold a big chunk from the other relation.
- Contiguous vs non-contiguous storage model
  • **Contiguous**: 1 I/O fetches a **block** (many tuples).
  • **Non-contiguous (worst case)**: 1 I/O fetches **one tuple**.
  Also, index clustering vs non-clustering changes how many I/Os you need to pull a group of tuples.

2) Iteration (Nested-Loop) Join — simplest (non-contiguous) case

Assumptions (from your note's "Simplest Case"):

1. Relations are **not** contiguous ⇒ 1 I/O brings **one** record.
2. For **each** tuple of R, we scan **all** of S to find matches.

Cost derivation

- Read each tuple of R once: cost $= T(R) = 60{,}000$ I/Os.
- For each of the $T(R)$ tuples, scan all of S: cost $= T(R) \times T(S) = 60{,}000 \times 2{,}000$.
- Total:

$$\text{Cost} = T(R) + T(R) \cdot T(S) = T(R)\big(1 + T(S)\big).$$

  Plugging numbers: $1 + T(S) = 1 + 2{,}000 = 2{,}001$.
  $60{,}000 \times 2{,}001 = 60{,}000 \times 2{,}000 + 60{,}000 \times 1 = 120{,}000{,}000 + 60{,}000 = 120{,}060{,}000$ I/Os.

This model effectively uses only **two buffers** (one for the current R tuple, one for streaming S), which is why it's so slow but always possible regardless of storage layout.

3) Improvement 1/3 (non-contiguous, but use all memory)

Idea: keep **as many R tuples** in memory as possible (125 buffers), and use the last buffer to stream S.

- One buffer holds $\frac{4{,}000}{100} = 40$ R-tuples;
  125 buffers hold $125 \times 40 = 5{,}000$ R-tuples.
  So we process R in **12 chunks** of $5{,}000$ tuples because $\frac{60{,}000}{5{,}000} = 12$.
- For **each chunk**: we re-scan all of S (still non-contiguous ⇒ $T(S) = 2{,}000$ I/Os per scan). We need 12 such scans.
- We must also read **all tuples of** R once (non-contiguous ⇒ $T(R) = 60{,}000$ I/Os).

Total cost:

$$\text{Cost} = T(R) + 12 \cdot T(S) = 60{,}000 + 12 \cdot 2{,}000 = 60{,}000 + 24{,}000 = 84{,}000 \text{ I/Os.}$$

Huge drop from 120,060,000!

---

4) Improvement 2/3 (contiguous R and S, use all memory)

Now assume **contiguous** files. One I/O fetches a **block**. Keep **125 blocks of** R in memory (that's $125 \times 40 = 5{,}000$ R-tuples again). Then stream S **by blocks** (8 S-tuples per block). We repeat this for every 125-block chunk of R. Since $B(R) = 1{,}500$, we have $\frac{1{,}500}{125} = 12$ rounds.

- Reading R once (by blocks): $B(R) = 1{,}500$ I/Os.
- For each round, scan all of S by blocks: $B(S) = 250$ I/Os per round × 12 rounds.

- Total:

$$\text{Cost} = B(R) + 12 \cdot B(S) = 1{,}500 + 12 \cdot 250 = 1{,}500 + 3{,}000 = 4{,}500 \text{ I/Os.}$$

---

5) Improvement 3/3 (flip the roles: chunk S and rescan R)

We now **chunk** S to fit memory and, for **each chunk of** S, we scan all of R. The note analyzes two storage models:

a) Non-contiguous case

- Treat S in **two** chunks of $1{,}000$ tuples each (so we'll scan R twice). Reading all of S still costs $T(S) = 2{,}000$ I/Os in total. Reading R twice costs $2 \cdot T(R) = 120{,}000$ I/Os.
- Total:

$$\text{Cost} = T(S) + 2 \cdot T(R) = 2{,}000 + 120{,}000 = 122{,}000 \text{ I/Os } (= 2 \cdot (1{,}000 + 60{,}000)).$$

This is **worse** than Improvement 1 (84,000).

> Intuition: when files are non-contiguous (1 I/O per tuple), repeatedly scanning the *larger* relation R is painful.

b) Contiguous case

- With contiguous S: 125 buffers can hold 125 blocks of S at once; each block holds 8 tuples ⇒ **1,000** S-tuples per chunk. That's **2 chunks** to cover all $2{,}000$ tuples of S.
- Reading one S chunk costs 125 I/Os (not 1,000), .d each time we rescan R by blocks.

- Total:

$$\text{Cost} = B(S) + 2 \cdot B(R) = 250 + 2 \cdot 1{,}500 = 250 + 3{,}000 = 3{,}250 \text{ I/Os.}$$

This is **better** than Improvement 2's 4,500 I/Os.

> Intuition: when files are contiguous (many tuples per I/O), it can be cheaper to hold large chunks of the **smaller** relation in memory and rescan the **larger** relation a few times—because rescanning by **blocks** is relatively cheap.

Here are **two more practice questions** in the exact same style as your running example:

🔹 Question 1

Running Example

- Two Relations R and S
- $T(R) = 80,000$
- $T(S) = 10,000$
- Block size = 4000 bytes
- $S(R) = 200$ bytes
- $S(S) = 100$ bytes
- Main Memory = 101 Buffers

Tasks:

1. Compute $B(R)$ and $B(S)$.
2. If relations are **not contiguous**, estimate the I/O cost of a block nested-loop join with R as outer relation.
3. If relations are **contiguous**, repeat the calculation.
4. Now try chunking S instead of R. Which strategy is better?

🔹 Question 2

Running Example

- Two Relations R and S
- $T(R) = 50,000$
- $T(S) = 5,000$
- Block size = 4096 bytes
- $S(R) = 128$ bytes
- $S(S) = 256$ bytes
- Main Memory = 50 Buffers

Tasks:

1. Compute the number of blocks $B(R)$ and $B(S)$.
2. Using **block nested-loop join**, calculate the cost if R is chosen as outer relation.
3. Repeat the calculation if S is chosen as outer relation.
4. Compare both cases and identify which choice minimizes I/Os.

Question 1

Exercise 3

Relation R: $B(R) = 12,000$

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) $M = 200$

b) $M = 100$

c) $M = 40$

Question 2

Exercise 4

Relation R: $B(R) = 20,000$

Relation S: $B(S) = 8,000$

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) $M = 75$

c) $M = 25$

Exercise 3a

Relation R : $B(R) = 12,000, T(R) = 120,000$

Relation S : $B(S) = 6,000, T(S) = 48,000$

$M = 150.$

Calculate I/Os:

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 the **minimum memory requirement** for a **One-Pass Join** if the relations are contiguous.

Exercise 3b

Relation R : $B(R) = 20,000, T(R) = 200,000$

Relation S : $B(S) = 8,000, T(S) = 64,000$

$M = 200.$

Calculate I/Os:

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 the **minimum memory requirement** for a **One-Pass Join** if the relations are contiguous.
