Sort–Merge Algorithm for External Sorting

The algorithm has **two main phases**:

1. Sorting Phase (Create Sorted Runs)

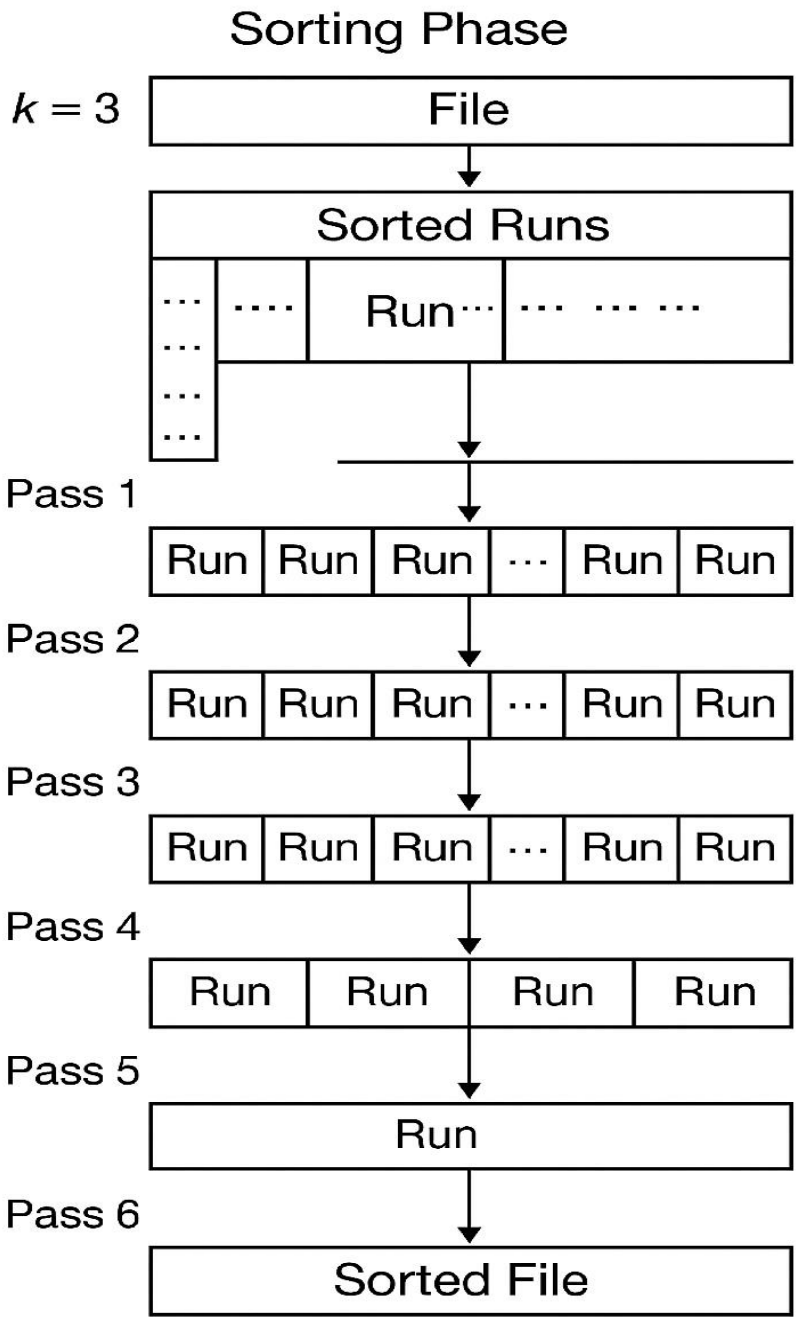
- Divide the file into chunks that fit into memory buffers.
- Load each chunk (say k blocks), sort it using an internal algorithm (like quicksort), and write it back to disk as a **sorted subfile (run)**.
- Continue until the whole file is processed.

After this step, you have multiple **sorted runs** on disk.

2. Merging Phase (Combine Runs)

- Take k-1 subfiles at a time into memory (keeping 1 buffer for output).
- Merge them into a larger sorted subfile, writing back to disk block by block.
- Repeat in multiple **passes**, reducing the number of subfiles each time, until only **1 sorted file remains**.

Example


Sorting Phase
k = 3
File
Sorted Runs
Run
Pass 1
Run Run Run ··· Run Run
Pass 2
Run Run Run ··· Run Run
Pass 3
Run Run Run ··· Run Run
Pass 4
Run Run Run Run
Pass 5
Run
Pass 6
Sorted File

Suppose:

- File has **10,000 records**.
- Each block stores **100 records**.
- Buffer can hold **3 blocks (k = 3)**.

Sorting Phase

- Total blocks in file = 10,000 ÷ 100 = **100 blocks**.
- Buffer can hold 3 blocks → sort 3 blocks at a time.
- Number of runs = 100 ÷ 3 ≈ **34 runs** (each run sorted internally).

So now we have 34 **sorted runs** stored on disk.

Merging Phase

- Use k–1 = 2 runs per merge pass.
- Pass 1: Merge runs pair by pair → 34 runs → **17 runs**.
- Pass 2: Merge again → 17 runs → **9 runs**.
- Pass 3: Merge again → 9 runs → **5 runs**.
- Pass 4: Merge again → 5 runs → **3 runs**.
- Pass 5: Merge again → 3 runs → **2 runs**.

- Pass 6: Merge again → **final single sorted file**.

(a) Sort–Merge Join (R ⋈ S on A = B)

Steps:

1. Sort relation **R** on attribute A and relation **S** on attribute B.
2. Use two pointers (i for R, j for S).
3. If R[i].A < S[j].B → move i.
   If R[i].A > S[j].B → move j.
   If R[i].A = S[j].B → output the join pair and continue.

Example:

R:

A | Name

1 | Alex

3 | Ben

5 | Cathy

S:

B | Dept

1 | HR

3 | IT

5 | Finance

👉 Output (R ⋈ S):

(1, Alex, HR)

(3, Ben, IT)

(5, Cathy, Finance)

📌 (b) Sort–Merge Projection (π attributes (R))

Steps:

1. From relation **R**, keep only required attributes into T’.
2. If projection key includes a primary key → no duplicates possible.
3. Otherwise → sort T’ and remove duplicates.

Example:
R:

(ID, Name, Dept)

1, Alex, HR

2, Ben, IT

3, Cathy, HR

Projection πDept(R):

HR, IT, HR

→ After removing duplicates:

HR, IT

📌 (c) Sort–Merge Union (R ∪ S)

Steps:

1. Sort both R and S.
2. Scan both lists with two pointers.
3. If R[i] < S[j] → output R[i].
   If R[i] > S[j] → output S[j].
   If equal → output one (skip duplicate).

Example:
R = {1, 3, 5}
S = {2, 3, 6}

Output (R ∪ S) = {1, 2, 3, 5, 6}

📌 (d) Sort–Merge Intersection (R ∩ S)

Steps:

1. Sort both R and S.

2. Scan both with two pointers.
3. If equal, output the tuple.
4. Otherwise advance the pointer of the smaller value.

Example:

R = {1, 3, 5}

S = {2, 3, 5, 7}

Output (R ∩ S) = {3, 5}

📌 (e) Sort–Merge Set Difference (R − S)

Steps:

1. Sort both R and S.
2. Scan both.
3. If R[i] < S[j] → output R[i].
4. If R[i] = S[j] → skip (remove common).
5. If R[i] > S[j] → move j.

Example:

R = {1, 3, 5, 7}

S = {3, 6, 7}

Output (R − S) = {1, 5}