



PA152: Efficient Use of DB

7. Sorting Algorithms

Vlastislav Dohnal

Applications of Sorting

- On result presentation
 - `SELECT ... ORDER BY`
- Doing joins
 - `SELECT ... r JOIN s`
- Filtering duplicates
 - `SELECT DISTINCT ...`

Assumptions

- Main Memory
 - Limited capacity – M blocks
- Data stored on disk
 - Input (relation) is read from a disk
- Output kept in memory
 - Output is usually processed by next operations
- Costs of sorting
 - Number of disk accesses

Sorting in Memory

■ Many Algorithms

- BubbleSort – $O(n^2)$
- QuickSort – $\Theta(n \log n)$
- MergeSort – $O(n \log n)$
- InsertSort – $O(n^2)$
- HeapSort – $O(n \log n)$
- RadixSort – $O(kn)$
- CountingSort – $O(k+n)$
- ...

Examples

■ Counting Sort

- Small cardinality of domain (values)
- E.g., need to sort 100 grades (A-F)
 - Create an array for all grades
 - Count the number of occurrences of each grade
 - Write the grades into correct (sorted) position

■ Radix Sort

- Recursive sorting by bytes (bits)
- Apply the CountingSort byte by byte
 - First round – get counts
 - Second round – locate and store the values to correct places

Sorting in Memory: Facts

- Data in main memory
- Sorting in-place
- Use little additional memory ($\log n$)

Small Main Memory

■ Data compression

- Process only key values and pointers to records; not whole records
- OK, but on output whole records must be read → random accesses

■ Memory virtualization

- Typically, slow → too many I/Os

■ Algorithm modification

- Combine more algorithms (ideas)
- MergeSort and QuickSort often used

MergeSort – in memory

i.e. for small relations!!!

■ “Divide and conquer” principle

□ Split to halves

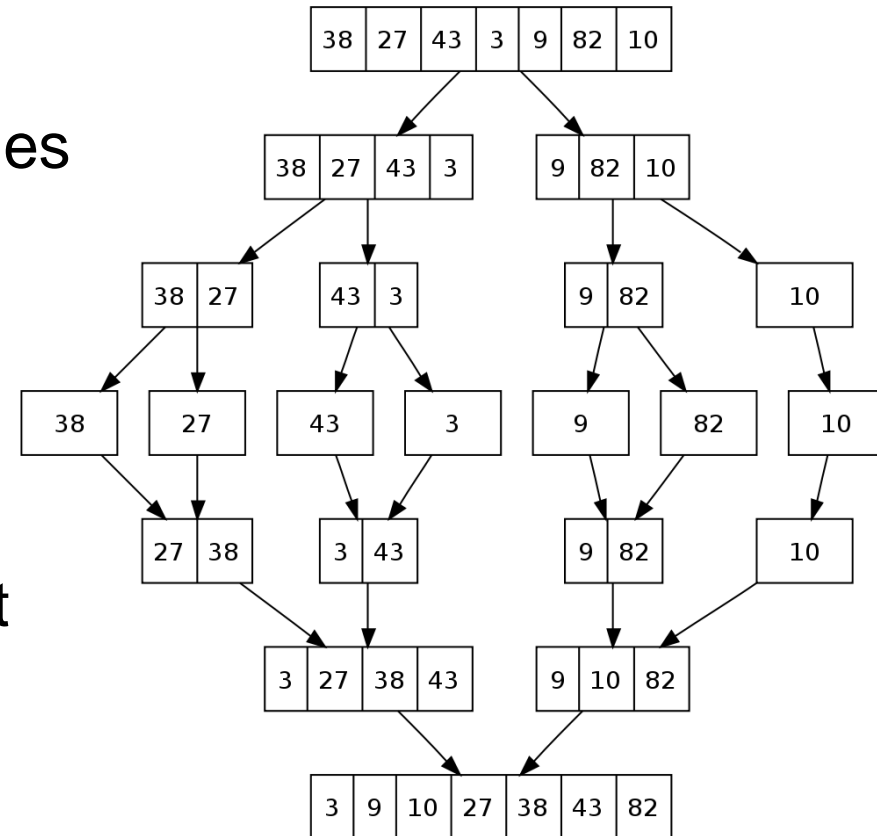
■ Until individual key values

□ Merge partly-sorted partitions

■ By linear scan of both parts

■ The smallest value sent to output

■ $O(n \log n)$



Source: Wikipedia.org, MergeSort Algorithm

MergeSort – disk-based variant

- Two-Phase Multiway MergeSort
- Procedure
 1. Create runs of size of available RAM
 2. Sort each run
 1. Read the run from a disk
 2. Sort in mem
 3. Write out to the disk
 3. Read all sorted runs at once and merge them

Two-phase MergeSort

■ Example

- Relation of 100 mil. records, 100 bytes each
- Blok of 8 kB, i.e., 80 records
 - Relation stored in 1 250 000 blocks (9.5 GB)
- Memory buffer for sorting
 - 6 400 blocks (50 MB)

■ Phase 1

- $\lceil 1\,250\,000 / 6\,400 \rceil$ runs = 196 runs
 - The last run contains 2,000 blocks only
- I/O: 1 250 000 reads + 1 250 000 writes

Two-phase MergeSort

■ Phase 2

- In-memory merging of two runs is slow!
 - i.e., $\log_2(\# \text{ of runs})$ reading and writing the relation
 - For 196 runs – $8\times$ reading and writing the whole file
- Multi-way merging
 - Read all runs block by block
 - Do merging into an output block

Two-phase MergeSort

■ Phase 2

□ Repeat

- Find the smallest value out of all runs
- Write it to output block
 - If full → flush it to disk
- An empty block of a run → read the next block of it

□ Resulting in 1x reading and 1x writing of relation

- i.e., 1 250 000 blocks read + 1 250 000 blocks written

■ In total

□ $4 \cdot B(R)$ I/Os, i.e., $O(n)$

- where $2 \cdot B(R)$ sequentially, $2 \cdot B(R)$ randomly

Two-phase MergeSort – Limitations

■ Parameters

- M – size of main memory buffer in blocks
- $B(R)$ – size of relation R in blocks

■ Limitations

- Max. run length: M
- Max. number of runs: $M-1$
- Max. relation size: $M \cdot (M-1)$

■ Running example: 100B record, 50MB buffer, 8KB block

- Max. 81 913 600 blocks(625GB)
- Max. 6 553 088 000 records

- If not enough, three-phase sort can be applied...