External Sorting

Why external sort?

- Sort is very common operation
  - B+ tree loading
  - ORDER BY
    - precursor to sort-merge join
    - aggregation and ordered-analytics
- Can’t guarantee that the table (or intermediate/final result) would fit in memory for sorting.
- Adapt technique to disk resident data

Step 1: create sorted runs

- assume B blocks/pages of memory are available

1. $i = 0$
2. **repeat**
    1. read B blocks of table (or the remaining, if smaller
    2. sort the in-memory blocks on the sort key
    3. write the sorted data into run R_i
3. **until** the end of input

Sorting Initial Runs

```
[618,587,491,623][517,565,914,197][519,566,592,554][408,254,450,756][768,619,561,539]
[926,340,858,788][583,112,571,  3][895,492,867,253][160,931,415, 51][580,911,845,306]
[144,574,414,416][583,337,701,977][767,270,199,143][597, 69, 70,944][160, 39,614,785]
[116,298,114,721][406,185,324,368][239,544,720,678][767,718,763,704][527,171,957, 52]
[778,689,264,366][585,238,380,244][890,855,937, 12][644,313,528,773][770,385,695,426]
[377,460,343,355][120,464,891,790][ 27,329,358,403][228,175, 81,597][ 79,621,209,404]
```

- Sorted Runs
  - Page can contain 4 numbers, memory available for 5 pages

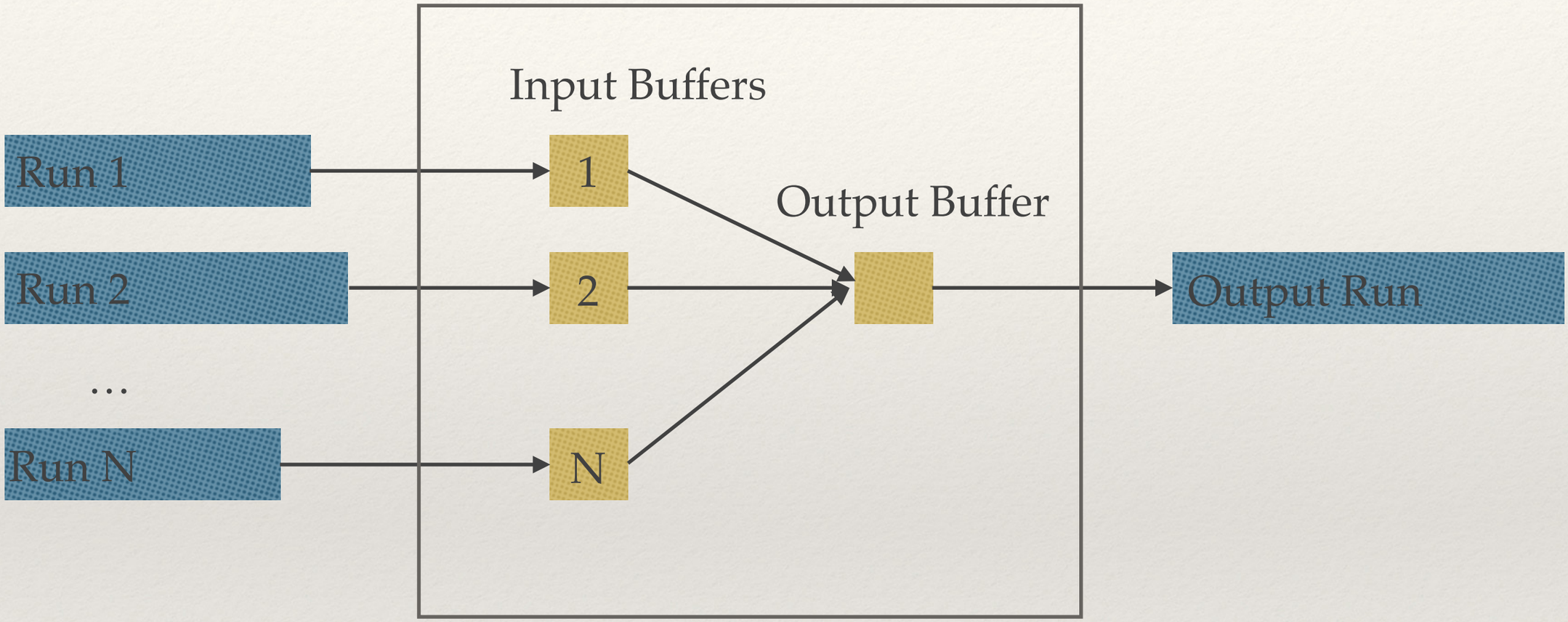
```
R1:[197,254,408,450][491,517,519,539][554,561,565,566][587,592,618,619][623,756,768,914]
R2:[  3, 51,112,160][253,306,340,415][492,571,580,583][788,845,858,867][895,911,926,931]
R3:[ 39, 69, 70,143][144,160,199,270][337,414,416,574][583,597,614,701][767,785,944,977]
R4:[ 52,114,116,171][185,239,298,324][368,406,527,544][678,704,718,720][721,763,767,957]
R5:[ 12,238,244,264][313,366,380,385][426,528,585,644][689,695,770,773][778,855,890,937]
R6:[ 27, 79, 81,120][175,209,228,329][343,355,358,377][403,404,460,464][597,621,790,891]
```

Step 2: merge the runs

- Assume (for now) than number of runs, N < B
- Allocate 1 page frame for each run + 1 for output

1. read 1 block of each of the N runs into memory
2. **repeat**
    1. choose the first row in sort order among all buffer pages
    2. write the row to the output (and go to next row in the page)
    3. if the buffer page from which the row is read is now done (empty) and it's not the end of file for that run
        1. fetch the next block into buffer page
3. **until** all buffer pages are empty (and all runs are done)

N-way Merge: Visually



- N < B, number of page frames (memory) available for the sort

Merge

```
R1:[197,254,408,450][491,517,519,539][554,561,565,566][587,592,618,619][623,756,768,914]
R2:[  3, 51,112,160][253,306,340,415][492,571,580,583][788,845,858,867][895,911,926,931]
R3:[ 39, 69, 70,143][144,160,199,270][337,414,416,574][583,597,614,701][767,785,944,977]
R4:[ 52,114,116,171][185,239,298,324][368,406,527,544][678,704,718,720][721,763,767,957]
R5:[ 12,238,244,264][313,366,380,385][426,528,585,644][689,695,770,773][778,855,890,937]
R6:[ 27, 79, 81,120][175,209,228,329][343,355,358,377][403,404,460,464][597,621,790,891]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]

Output Page:[                 ]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]

Output Page:[ 3,              ]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]

Output Page:[ 3, 39,          ]
```

Merge

```
R1:[197,254,408,450][491,517,519,539][554,561,565,566][587,592,618,619][623,756,768,914]
R2:[  3, 51,112,160][253,306,340,415][492,571,580,583][788,845,858,867][895,911,926,931]
R3:[ 39, 69, 70,143][144,160,199,270][337,414,416,574][583,597,614,701][767,785,944,977]
R4:[ 52,114,116,171][185,239,298,324][368,406,527,544][678,704,718,720][721,763,767,957]
R5:[ 12,238,244,264][313,366,380,385][426,528,585,644][689,695,770,773][778,855,890,937]
R6:[ 27, 79, 81,120][175,209,228,329][343,355,358,377][403,404,460,464][597,621,790,891]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]
```

```
Output Page:[ 3, 39, 51,   ]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]
```

```
Output Page:[ 3, 39, 51, 52]
```

- Page Full
- Write the output page

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]
```

```
Output Page:[ 69,          ]
```

Merge

```
R1:[197,254,408,450][491,517,519,539][554,561,565,566][587,592,618,619][623,756,768,914]
R2:[  3, 51,112,160][253,306,340,415][492,571,580,583][788,845,858,867][895,911,926,931]
R3:[ 39, 69, 70,143][144,160,199,270][337,414,416,574][583,597,614,701][767,785,944,977]
R4:[ 52,114,116,171][185,239,298,324][368,406,527,544][678,704,718,720][721,763,767,957]
R5:[ 12,238,244,264][313,366,380,385][426,528,585,644][689,695,770,773][778,855,890,937]
R6:[ 27, 79, 81,120][175,209,228,329][343,355,358,377][403,404,460,464][597,621,790,891]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]
```

```
Output Page:[ 69, 70,        ]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]
```

```
Output Page:[ 69, 70,112,    ]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]
```

```
Output Page:[ 69, 70,112,114]
```

- Page Full
- Write the output page

Merge

```
R1:[197,254,408,450][491,517,519,539][554,561,565,566][587,592,618,619][623,756,768,914]
R2:[  3, 51,112,160][253,306,340,415][492,571,580,583][788,845,858,867][895,911,926,931]
R3:[ 39, 69, 70,143][144,160,199,270][337,414,416,574][583,597,614,701][767,785,944,977]
R4:[ 52,114,116,171][185,239,298,324][368,406,527,544][678,704,718,720][721,763,767,957]
R5:[ 12,238,244,264][313,366,380,385][426,528,585,644][689,695,770,773][778,855,890,937]
R6:[ 27, 79, 81,120][175,209,228,329][343,355,358,377][403,404,460,464][597,621,790,891]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]      Output Page:[116,           ]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]      Output Page:[116,143,       ]
R3:[ 39, 69, 70,143]
R4:[ 52,114,116,171]
```

```
Input Pages:
R1:[197,254,408,450]
R2:[  3, 51,112,160]      Output Page:[116,143,144,   ]
R3:[144,160,199,270]
R4:[ 52,114,116,171]
```

- Fetch new R3 page

Output Runs & Merge

- Run 1

```
[  3, 39, 51, 52][ 69, 70,112,114][116,143,144,160][160,171,185,197][199,239,253,254]
[270,298,306,324][337,340,368,406][408,414,415,416][450,491,492,517][519,527,539,544]
[554,561,565,566][571,574,580,583][583,587,592,597][614,618,619,623][678,701,704,718]
[720,721,756,763][767,767,768,785][788,845,858,867][895,911,914,926][931,944,957,977]
```

- Run 2

```
[ 12, 27, 79, 81][120,175,209,228][238,244,264,313][329,343,355,358][366,377,380,385]
[403,404,426,460][464,528,585,597][621,644,689,695][770,773,778,790][855,890,891,937]
```

```
Input Pages:
R1:[  3, 39, 51, 52]
R2:[ 12, 27, 79, 81]          Output Page:[                ]


Input Pages:
R1:[  3, 39, 51, 52]          Output Page:[ 3,             ]
R2:[ 12, 27, 79, 81]

Input Pages:
R1:[  3, 39, 51, 52]          Output Page:[ 3, 12,         ]
R2:[ 12, 27, 79, 81]
```

Output Runs Merge

- Run 1

```
[  3, 39, 51, 52][ 69, 70,112,114][116,143,144,160][160,171,185,197][199,239,253,254]
[270,298,306,324][337,340,368,406][408,414,415,416][450,491,492,517][519,527,539,544]
[554,561,565,566][571,574,580,583][583,587,592,597][614,618,619,623][678,701,704,718]
[720,721,756,763][767,767,768,785][788,845,858,867][895,911,914,926][931,944,957,977]
```

- Run 2

```
[ 12, 27, 79, 81][120,175,209,228][238,244,264,313][329,343,355,358][366,377,380,385]
[403,404,426,460][464,528,585,597][621,644,689,695][770,773,778,790][855,890,891,937]
```

```
Input Pages:
R1:[  3, 39, 51, 52]
R2:[ 12, 27, 79, 81]        Output Page:[  3, 12, 27    ]


Input Pages:
R1:[  3, 39, 51, 52]        Output Page:[  3, 12, 27, 39]
R2:[ 12, 27, 79, 81]


Input Pages:
R1:[  3, 39, 51, 52]        Output Page:[ 51,           ]
R2:[ 12, 27, 79, 81]
```

- Page Full
- Write the output page

Output Runs

- Run 1

```
[  3, 39, 51, 52][ 69, 70,112,114][116,143,144,160][160,171,185,197][199,239,253,254]
[270,298,306,324][337,340,368,406][408,414,415,416][450,491,492,517][519,527,539,544]
[554,561,565,566][571,574,580,583][583,587,592,597][614,618,619,623][678,701,704,718]
[720,721,756,763][767,767,768,785][788,845,858,867][895,911,914,926][931,944,957,977]
```

- Run 2

```
[ 12, 27, 79, 81][120,175,209,228][238,244,264,313][329,343,355,358][366,377,380,385]
[403,404,426,460][464,528,585,597][621,644,689,695][770,773,778,790][855,890,891,937]
```

```
Input Pages:
R1:[  3, 39, 51, 52]
R2:[ 12, 27, 79, 81]          Output Page:[ 51, 52,       ]
```

```
Input Pages:
R1:[ 69, 70,112,114]
R2:[ 12, 27, 79, 81]          Output Page:[ 51, 52, 69,   ]
```

- Fetch new R1 page

```
Input Pages:
R1:[  3, 39, 51, 52]
R2:[ 12, 27, 79, 81]          Output Page:[ 51, 52, 69, 70]
```

- Page Full
- Write the output page

Final Merged Output

```
[  3, 12, 27, 39][ 51, 52, 69, 70][ 79, 81,112,114][116,120,143,144][160,160,171,175]
[185,197,199,209][228,238,239,244][253,254,264,270][298,306,313,324][329,337,340,343]
[355,358,366,368][377,380,385,403][404,406,408,414][415,416,426,450][460,464,491,492]
[517,519,527,528][539,544,554,561][565,566,571,574][580,583,583,585][587,592,597,597]
[614,618,619,621][623,644,678,689][695,701,704,718][720,721,756,763][767,767,768,770]
[773,778,785,788][790,845,855,858][867,890,891,895][911,914,926,931][937,944,957,977]
```

How many passes over data?

- A Merge Pass:
  - Merge the first B-1 runs to get a single run
    - Repeat for each set of B-1 till last set of runs is done
    - (Number of runs reduced by a factor of B-1)
- Repeat the merge pass till number of runs <= B-1 which would be the final pass

I/O Cost

- N is the number of blocks in data, B is available buffer
  - N = 30, B = 5
- # of initial runs of ⌈N/B⌉ requiring 2 I/O's per block
  - # = 30/5 = 6
- Each pass decreases number of runs by factor of B-1
  - From 6 we went to ⌈6/4⌉ = 2 runs
- # merge passes = $\lceil \log_{(B-1)}(\lceil N/B \rceil) \rceil$ requiring 2 I/O's per block
  - $\lceil \log_4(\lceil 30/5 \rceil) \rceil$ = 2 passes
- not counting final output, we get
  - N * (2 * $\lceil \log_{(B-1)}(\lceil N/B \rceil) \rceil$ + 1) I/O's = 30 * (2 * 2 + 1) = 30*5 = 150 pages of I/O

1001 page frames

- In one pass
  - read a 1001 page table, sort, write
- In two passes
  - read 1001000 pages table => 1000 runs of 1001 pages
  - merge 1000 runs using 1000 input and 1 output buffers to get result run
- In 3 passes
  - read 1001000000 pages table => 1000000 runs of 1001 pages
  - merge 1000 runs using 1000 input and 1 output buffers to get 1000 runs of 1001000 pages each
  - merge 1000 runs using 1000 input and 1 output buffer to get result run

B+ Tree and Sorting

- Data is already sorted on the primary index (i.e. index of the B+ tree), simply scan the leaf nodes
- Secondary Index B+ trees are not suited for sorting entire records as rid's are in "random" order

Internal Sort Algorithm

- In-memory (internal) sort algorithm can be important
  - Quicksort and Heapsort
  - Can optimize further for better memory / L2 cache performance

Summary

- DBMS's optimize sorting as it is a critical operation
- Specialized sorting options, e.g. duplicate elimination
- 1-3 passes in practice for all but largest tables
- B buffer pages can sort approximately
  - B pages in 1 pass (no runs, just read/sort in memory)
  - B^2 pages in 2 passes
  - B^3 pages in 3 passes
- B+ Tree's are already sorted on the index (key) value