COURSE 6

External Sorting

External Sorting

- Sorting: a classic problem in computer science!
- Critical for Databases:
  - Data requested in sorted order
  - Sorting is first step in *bulk loading* B+ tree index.
  - Sorting useful for eliminating *duplicate copies* in a collection of records
  - *Sort-merge* join algorithm involves sorting.
- Conventional sort algorithms: e.g. *Quick Sort, Heap Sort, Selection Sort,...*
  - Very efficient but all data needs to fit completely into main memory.

External Sorting (cont)

- Problem: sort 1Tb of data with 1Gb of RAM.
- External sorting: performing sorting operations on amounts of data that are too large to fit into main memory
- External sorting cannot be done in one step
- In general, external sorting consists of two phases:
    1. Break the file into initial runs.
    2. Merge the runs together into a single sorted run.

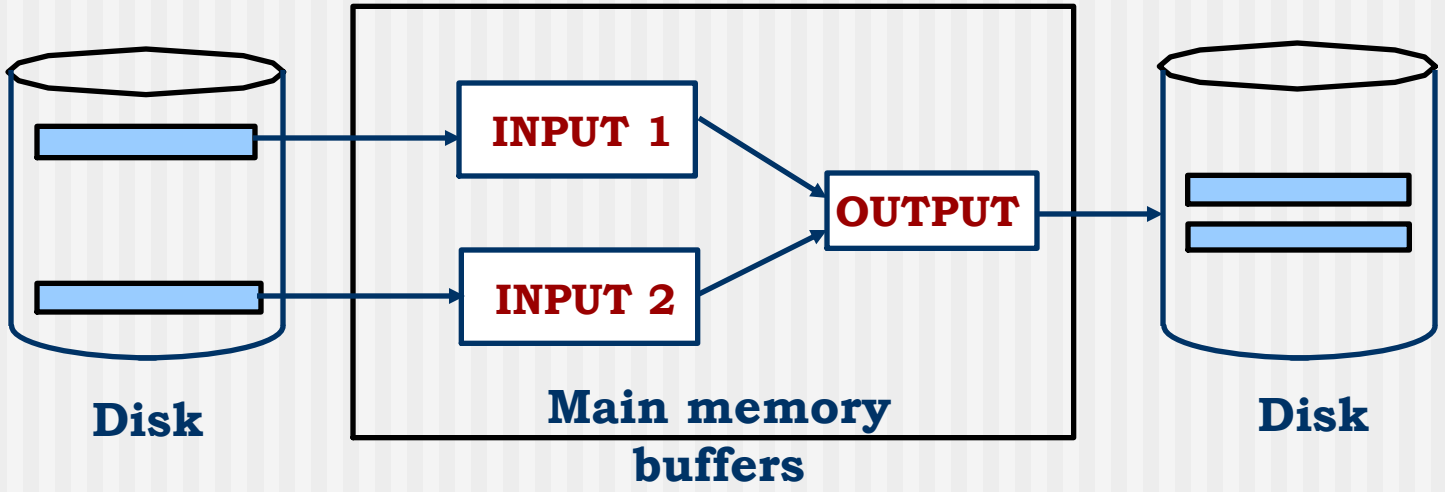
Run - a group of sorted records

General Principles

- A good external sorting algorithm will seek to do the following:
  - Make the initial runs as long as possible.
  - At all stages, overlap input, processing and output as much as possible.
  - Use as much working memory as possible. Applying more memory usually speeds processing.
  - If possible, use additional disk drives for more overlapping of processing with I/O, and allow for more sequential file processing.

Two-Way Sort: Requires 3 Buffers

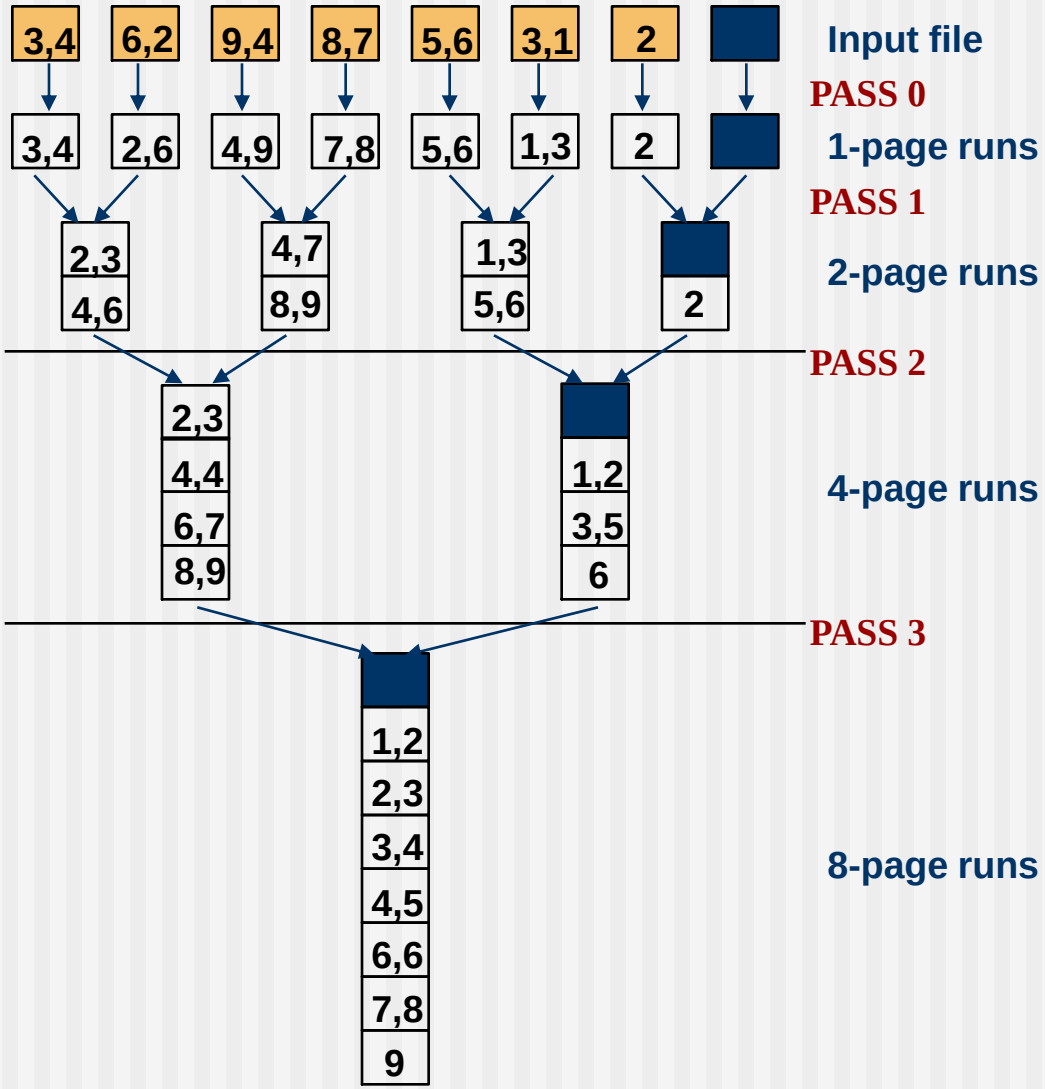
- Pass 0: Read a page → sort it → write it.
  - only one buffer page is used
- Pass 1, 2, …, etc.:
  - three buffer pages used.



Two-Way External Merge Sort

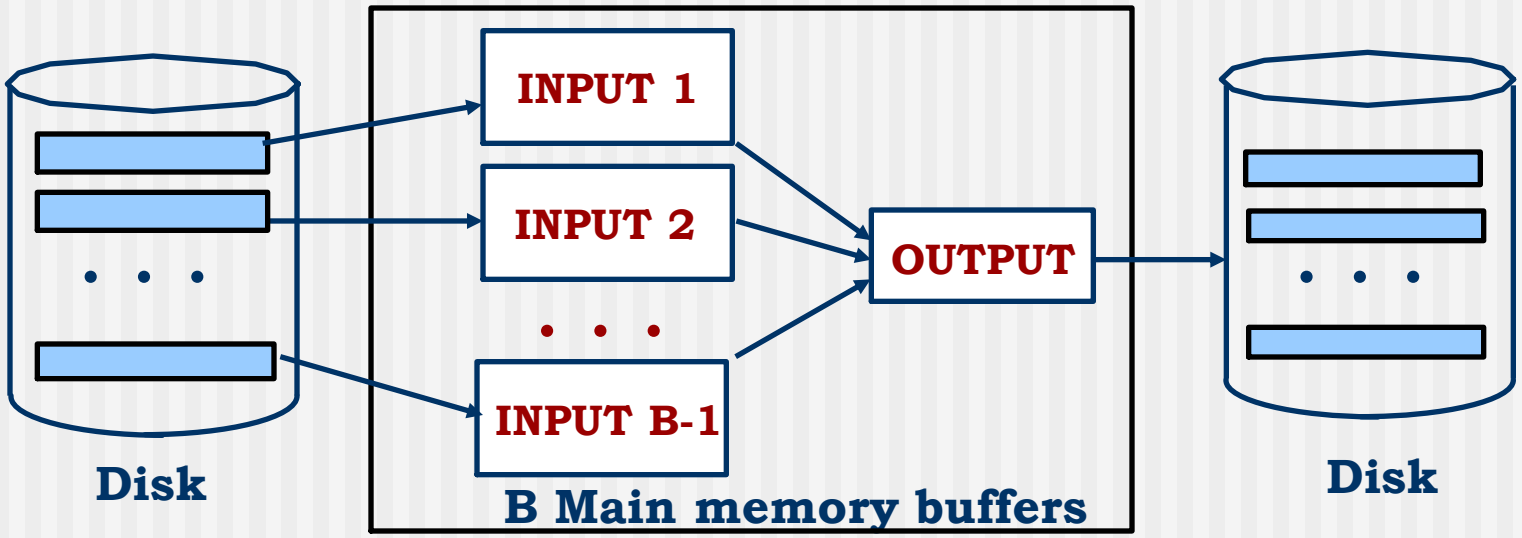
- Each pass we read + write each page in file.
- N pages in the file => the number of passes
$= \lceil \log_2 N \rceil + 1$
- So total cost is:
$2N(\lceil \log_2 N \rceil + 1)$

- *Idea:* ***Divide and conquer:*** sort sub-files and merge



General External Merge Sort

- To sort a file with *N* pages using *B* buffer pages:
  - Pass 0: use *B* buffer pages. Produce ⌈N/B⌉ sorted runs of *B* pages each.
  - Pass 1, 2, …, etc.: merge *B-1* runs.



Cost of External Merge Sort

- Number of passes: $1+\lceil \log_{B-1} \lceil N/B \rceil \rceil$
- Cost = 2N * (number of passes)
- E.g., with 5 buffer pages, to sort 108 page file:
  - Pass 0: $\lceil 108/5 \rceil = 22$ sorted runs of 5 pages each (last run is only 3 pages)
  - Pass 1: $\lceil 22/4 \rceil = 6$ sorted runs of 20 pages each (last run is only 8 pages)
  - Pass 2: 2 sorted runs, 80 pages and 28 pages
  - Pass 3: Sorted file of 108 pages

Number of Passes of External Sort

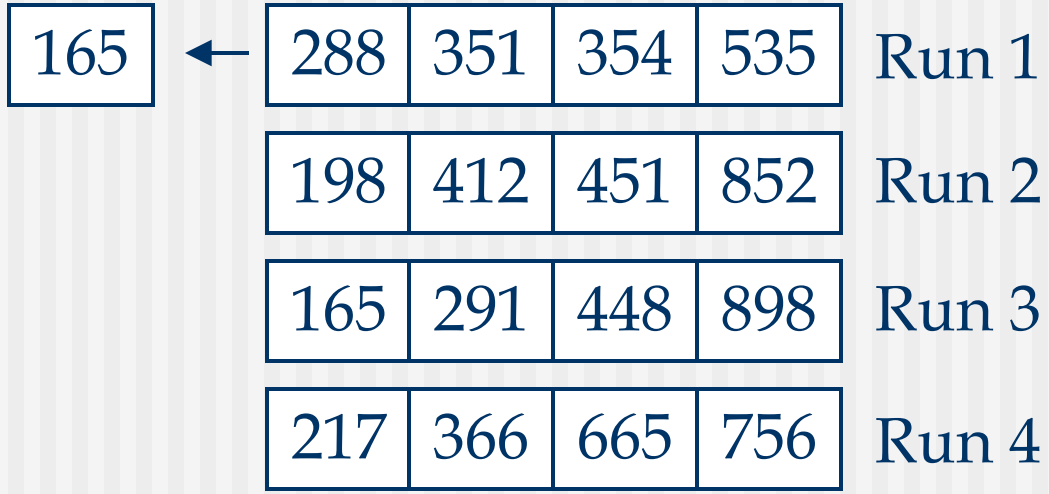
N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4

Sophisticated External Merge Sort

- **Optimizations:** Algorithms like *Polyphase merge, cascade merge*
  - Reducing the number of intermediate steps by implementing *n*-way-merging with great values of *n*.
  - Saving time by doing a perfect spreading of the runs on the storage media.
  - Maximizing speed by increasing the number of drives for storage disposals for minimal access time.
- **Disadvantages:** Additional costs and expenditure

Selection Tree

- **Problem**: selecting the smallest element is very time-consuming



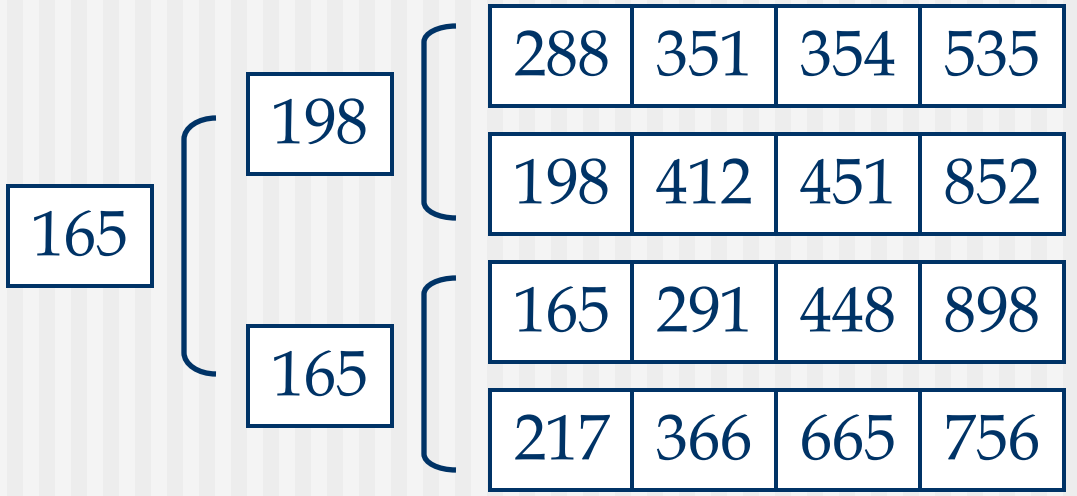
It requires $(N / P) - 1$ comparisons when using a non- advanced algorithm.

first element is compared subsequently with all remaining *P-1* elements

- **Solution:** Building a *selection tree* saves lots of comparisons and speeds up the selection process (just $log_2 P$ comparisons are necessary)

Selection Tree (cont)

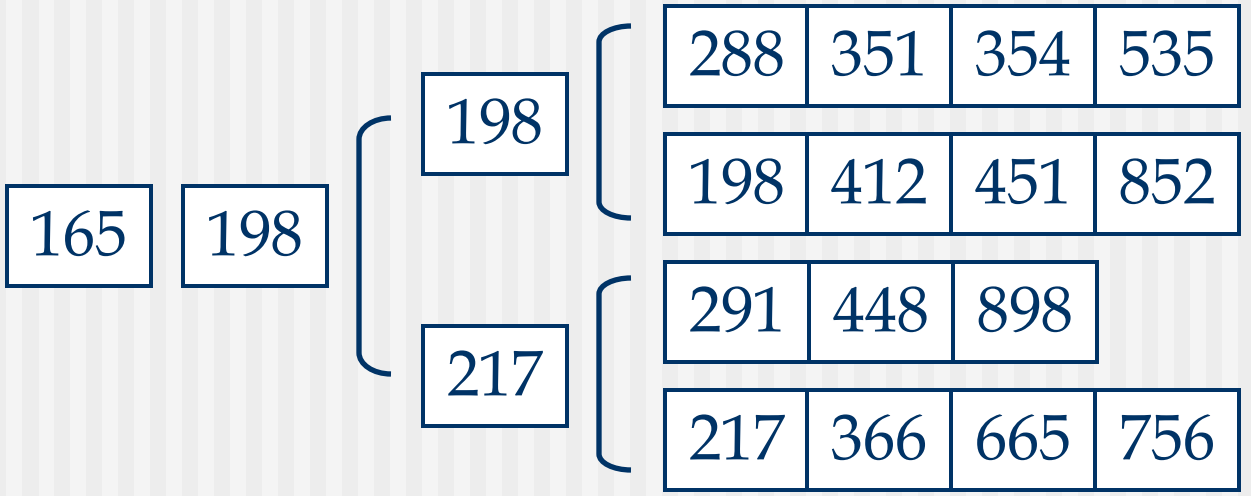
Start: Building a selection tree



Always the smallest element is taken out of the top of the tree
New elements are pulled forward in the current branch
Repeats until all branches of the selection tree are empty

Selection Tree (cont)

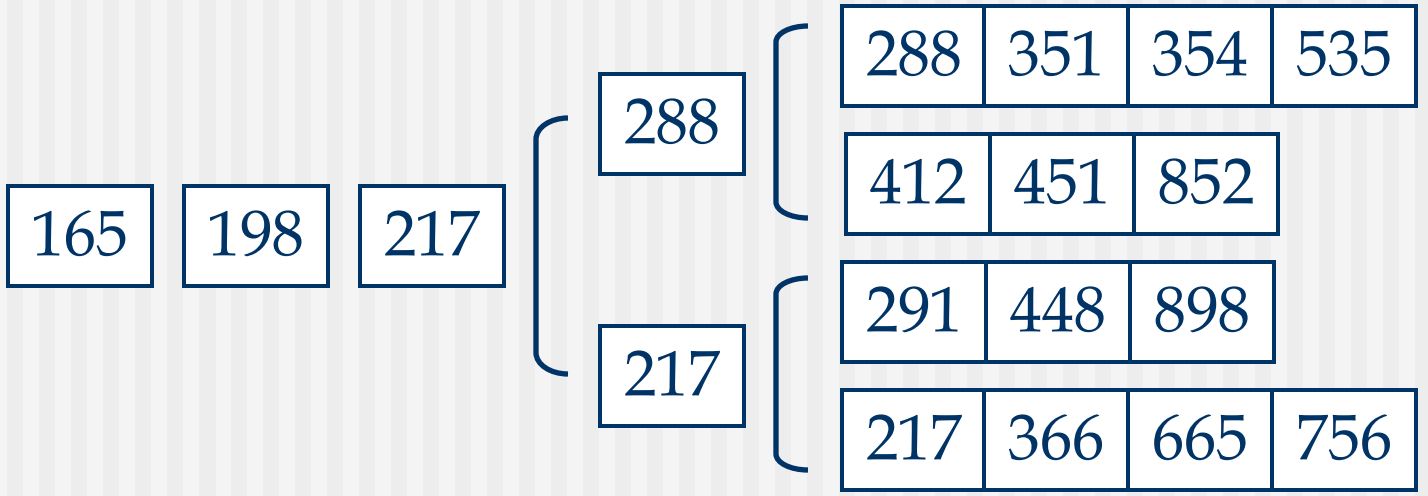
Step 1: Pulling smallest element forward



Always the smallest element is taken out of the top of the tree
New elements are pulled forward in the current branch
Repeats until all branches of the selection tree are empty

Selection Tree (cont)

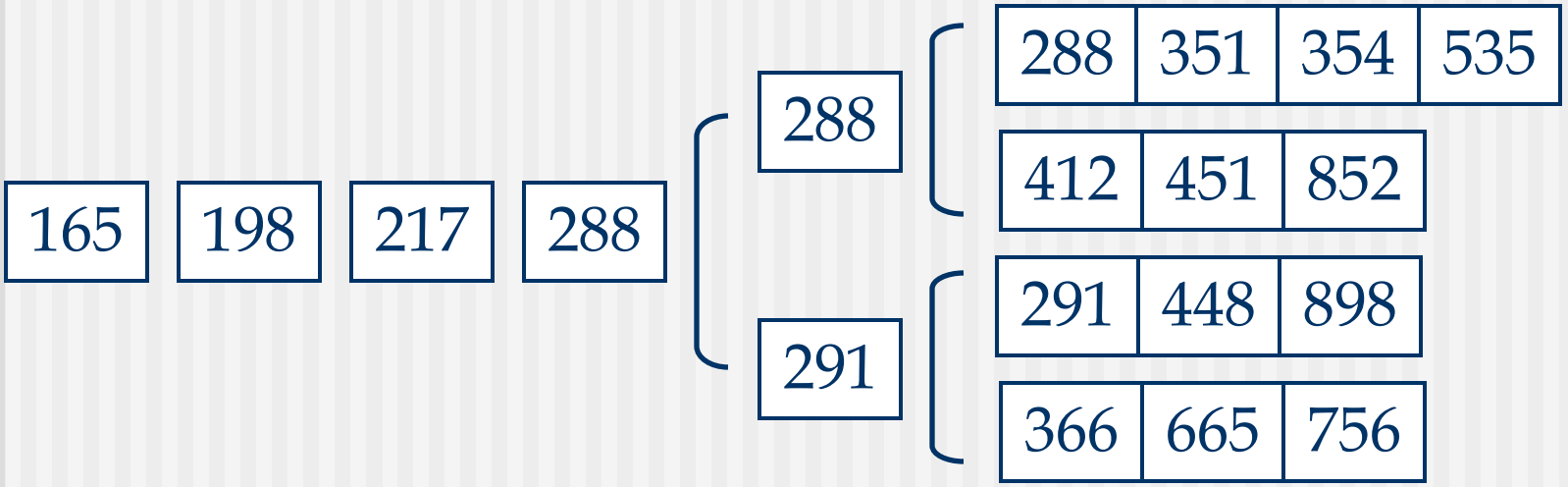
Step 2: Pulling smallest element forward



Always the smallest element is taken out of the top of the tree
New elements are pulled forward in the current branch
Repeats until all branches of the selection tree are empty

Selection Tree (cont)

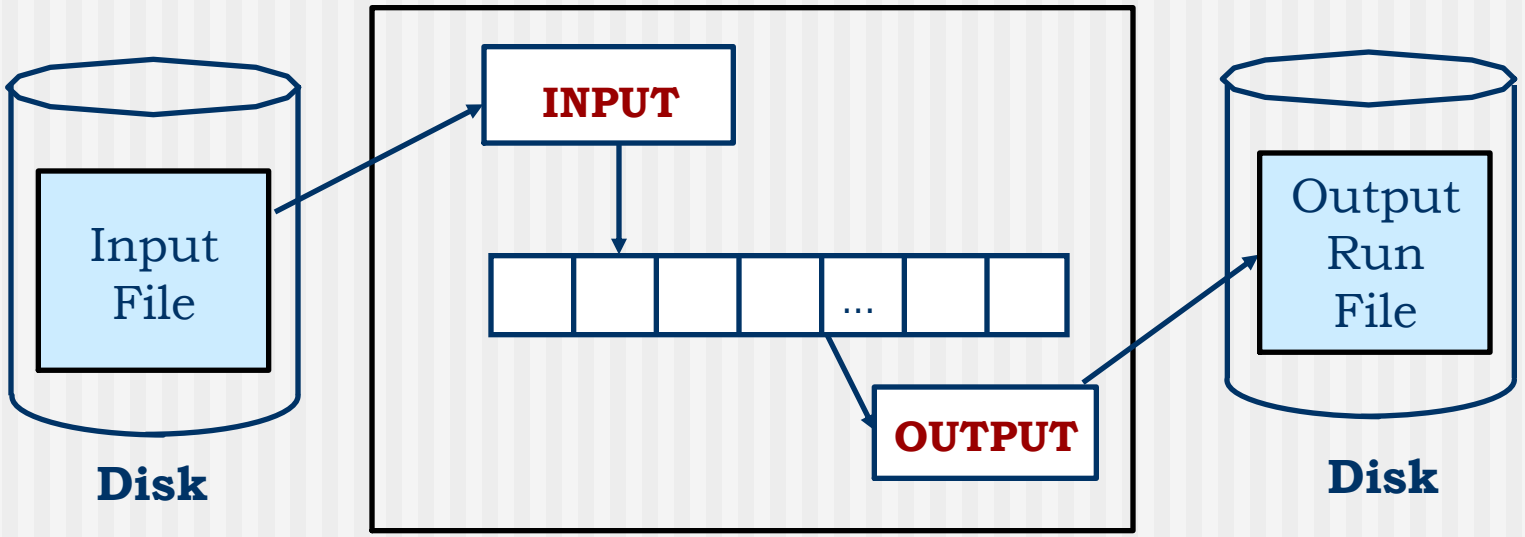
Step 3: Pulling smallest element forward



Always the smallest element is taken out of the top of the tree
New elements are pulled forward in the current branch
Repeats until all branches of the selection tree are empty

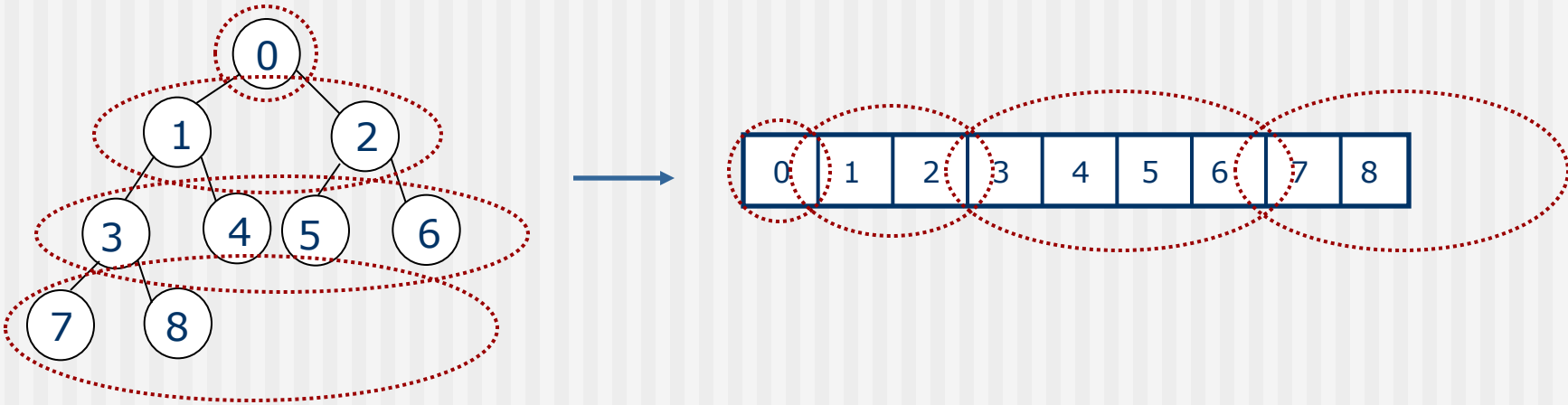
Internal Sort Algorithm

- Quicksort is a fast way to sort in memory.
- Replacement selection
  - Based on building min-heaps (complete binary trees; value of each node is less than child values)
  - Break available memory into an *array* for the heap, an *input buffer* and an *output buffer*.



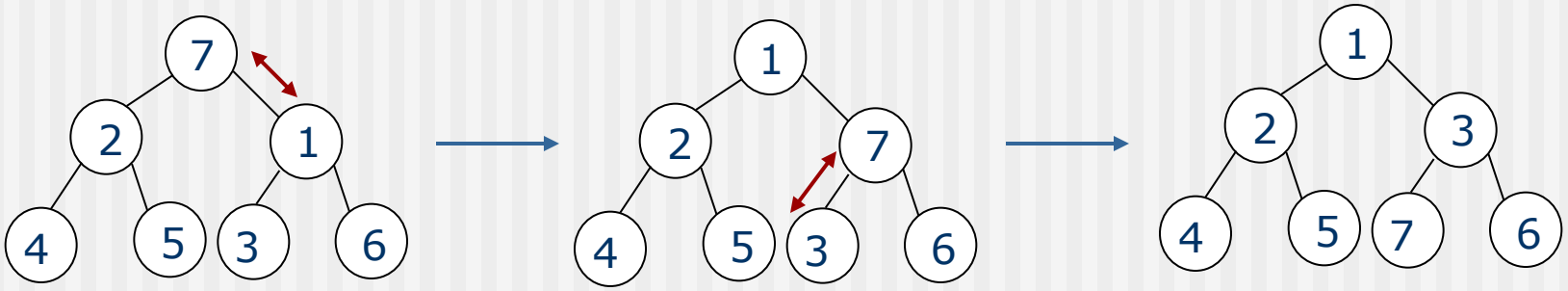
Min-Heaps

- Complete binary tree: If the height of the tree is *d*, then all leaves except possibly level *d* are completely full. The bottom level has all nodes to the left side.
- The values are partially ordered.
- Heap representation: Normally the array-based complete binary tree representation.



Min-Heaps (cont)

- Construction:
  - Work from high end of array to low end.
  - Call *siftdown* for each item (siftdown – put item at its place).
  - Don't need to call *siftdown* on leaf nodes.
  - Example: siftdown item 7



Replacement Selection Algorithm

1. Fill the array from disk.
2. Make a min-heap.
3. Send the smallest value (root) to the output buffer.
4. If the next key in the file is greater than the last value output:

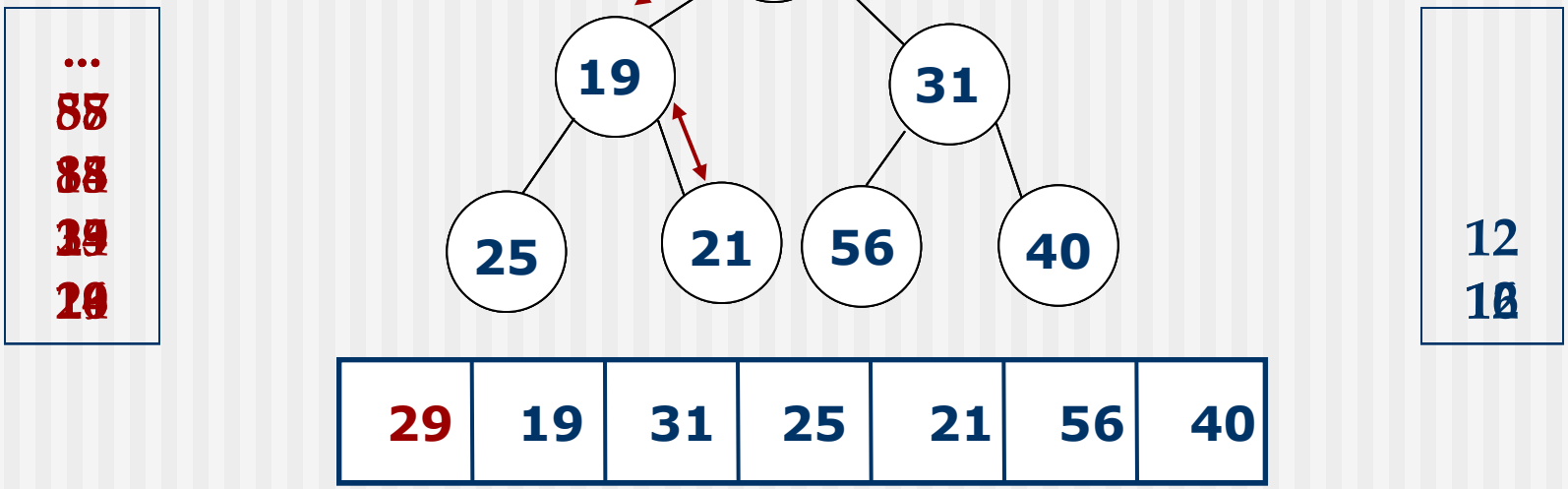
    - Replace the root with this key

   else

    - Replace the root with the last key in the array
    - Add the next record in the file to a new heap (actually, stick it at the end of the array).

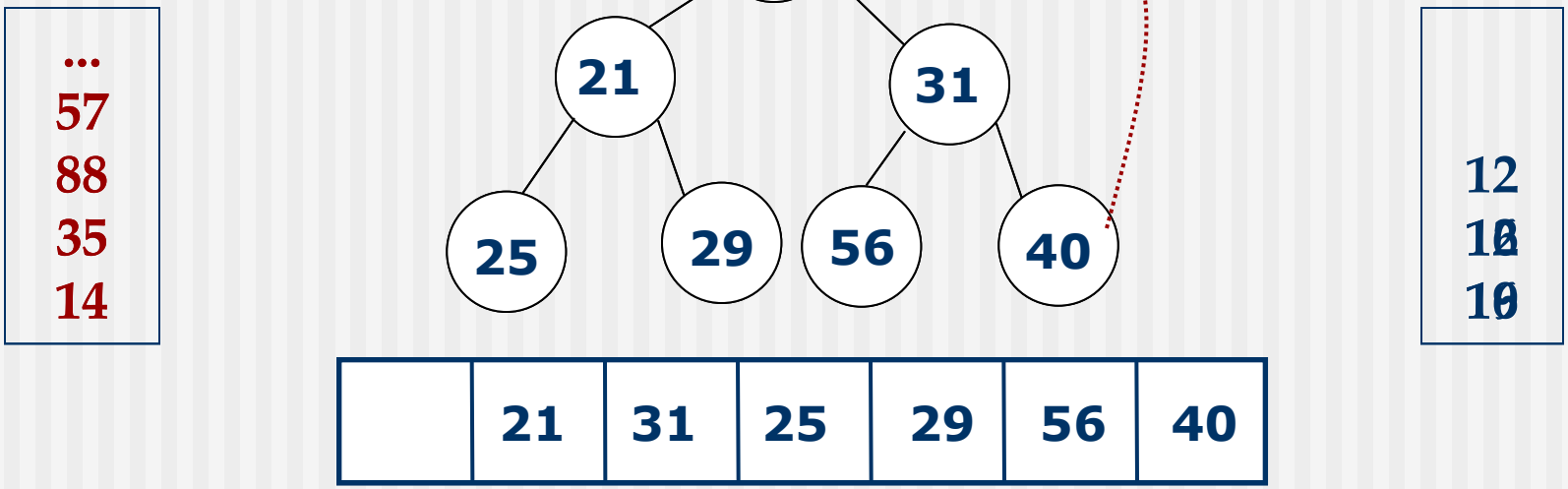
Example of Replacement Selection

Input | Heap-Array | Output



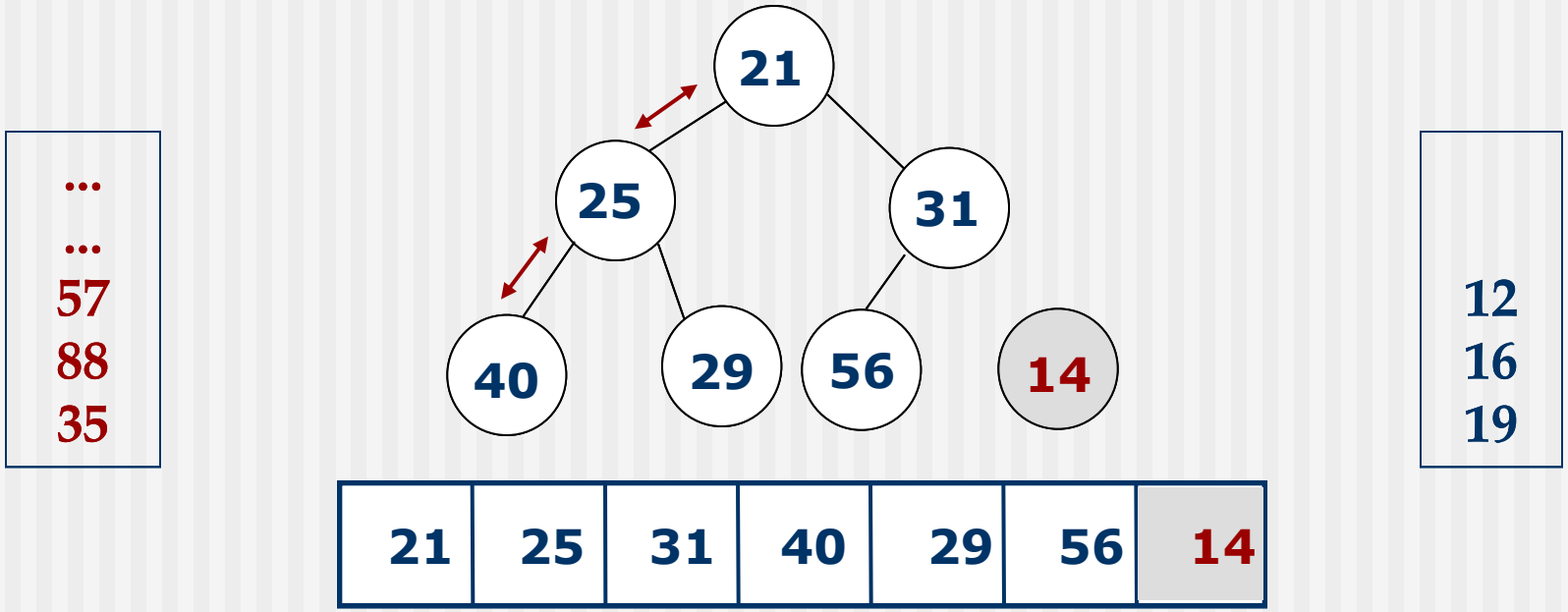
Example of Replacement Selection

Input | Heap-Array | Output



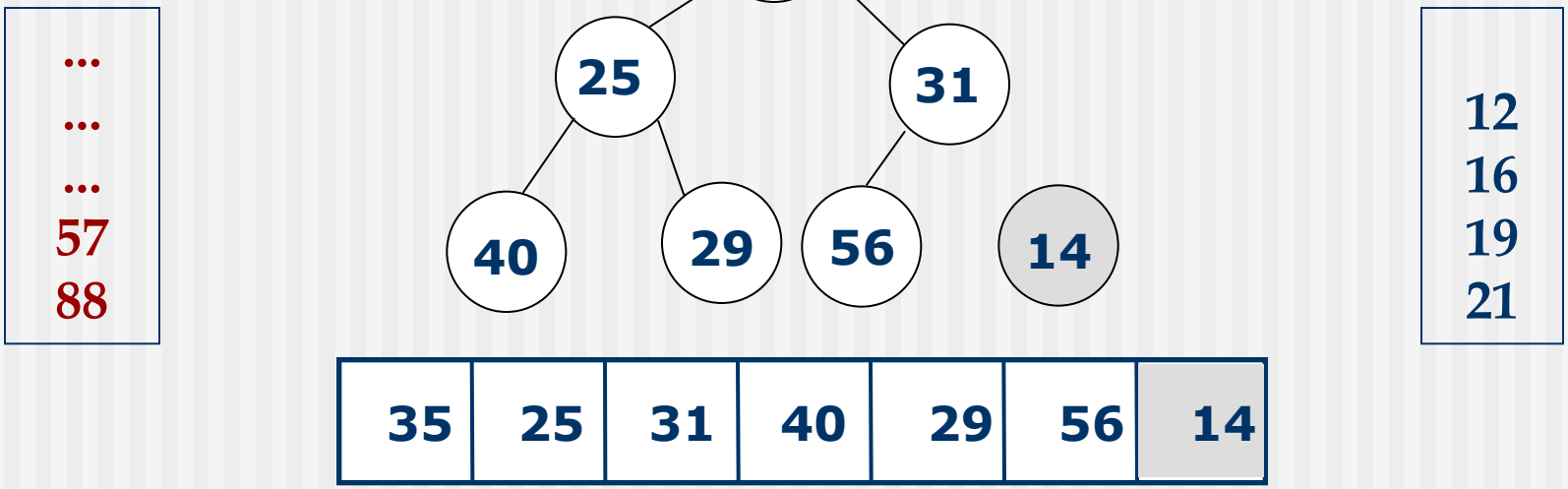
Example of Replacement Selection

Input | Heap-Array | Output



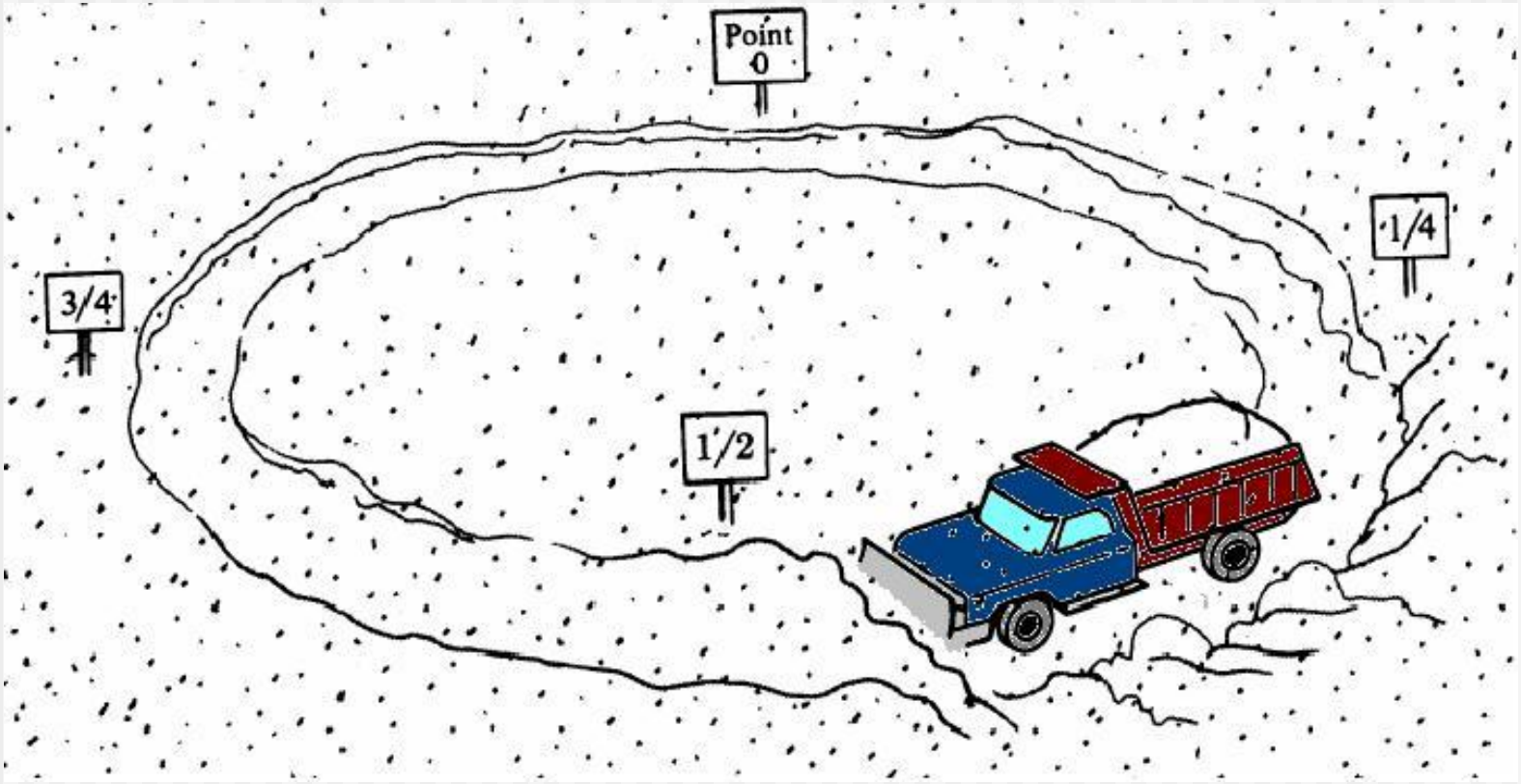
Example of Replacement Selection

Input | Heap-Array | Output



Replacement Selection – Snowplow analogy

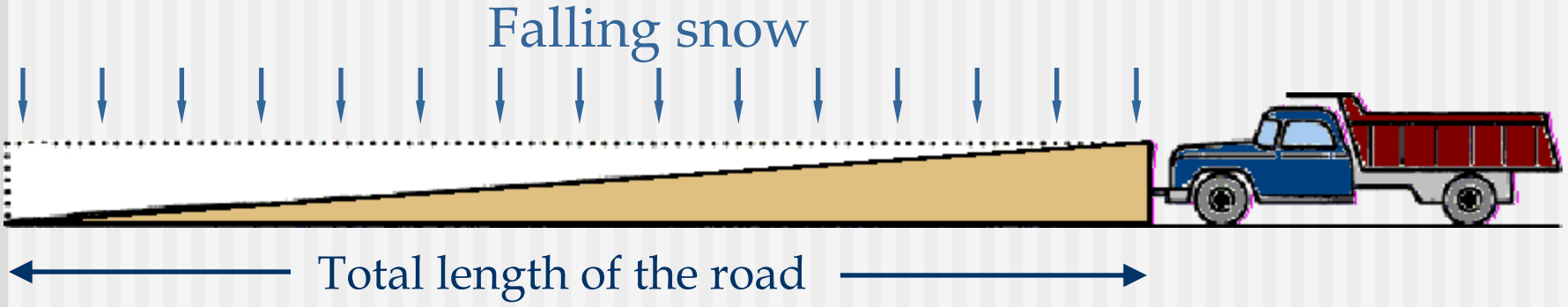
- We can expect initial runs of a length of 2 * q when q is the size of main memory. Why?



A snowplow is clearing a road with snow randomly distributed all over.

Replacement Selection – Snowplow analogy

- Because snow is falling at constant speed, this stable situation will never change:



- Rectangle is cut in half by the line representing the actual snow level
- Level of existing snow represents records in main memory
- At the end of the road, there is no snow from the prev. turn left
- All records from the last run are tagged with the marker, so a new run has to be created.
- The volume of snow removed in one circle (namely the length of a run) is twice the amount that is present on the track at any time.

I/O for External Merge Sort

- … longer runs often means fewer passes!
- Actually, do I/O a page at a time
- In fact, read a *block* of pages sequentially!
- Suggests we should make each buffer (input/output) be a *block* of pages.
  - But this will reduce fan-out during merge passes!
  - In practice, most files still sorted in 2-3 passes.

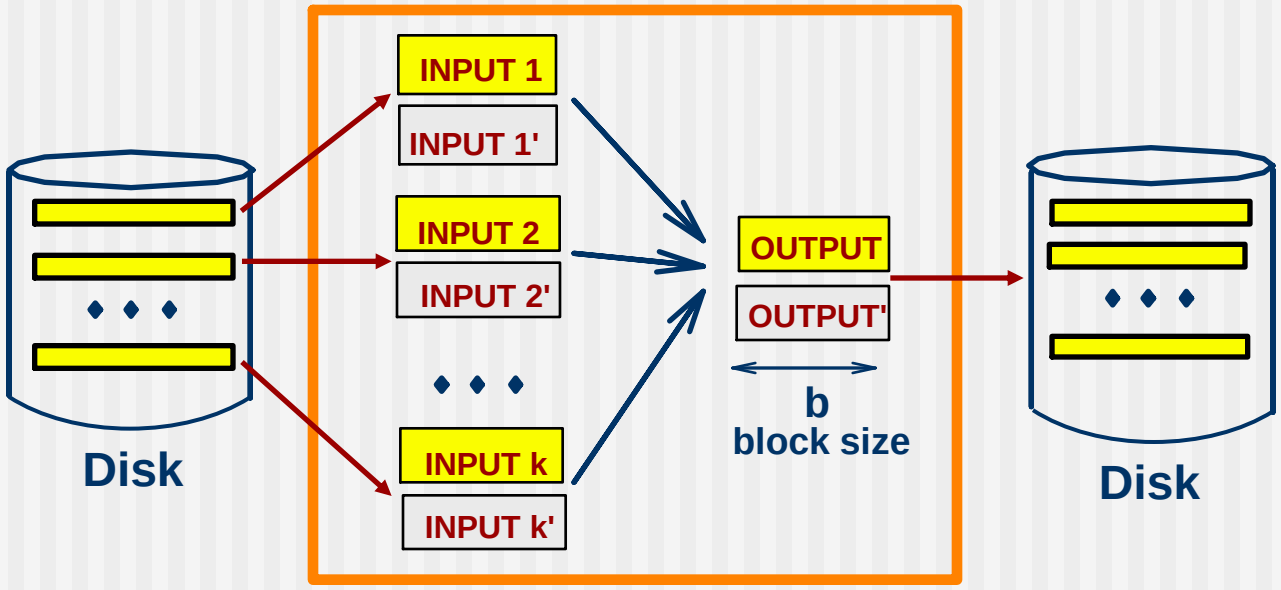
Number of Passes of Optimized Sort

N	B=1,000	B=5,000	B=10,000
100	1	1	1
1,000	1	1	1
10,000	2	2	1
100,000	3	2	2
1,000,000	3	2	2
10,000,000	4	3	3
100,000,000	5	3	3
1,000,000,000	5	4	3

* *Block size = 32, initial pass produces runs of size 2B.*

Double Buffering

- To reduce wait time for I/O request to complete, can *prefetch* into `shadow block'.
  - Potentially, more passes; in practice, most files *still* sorted in 2-3 passes.



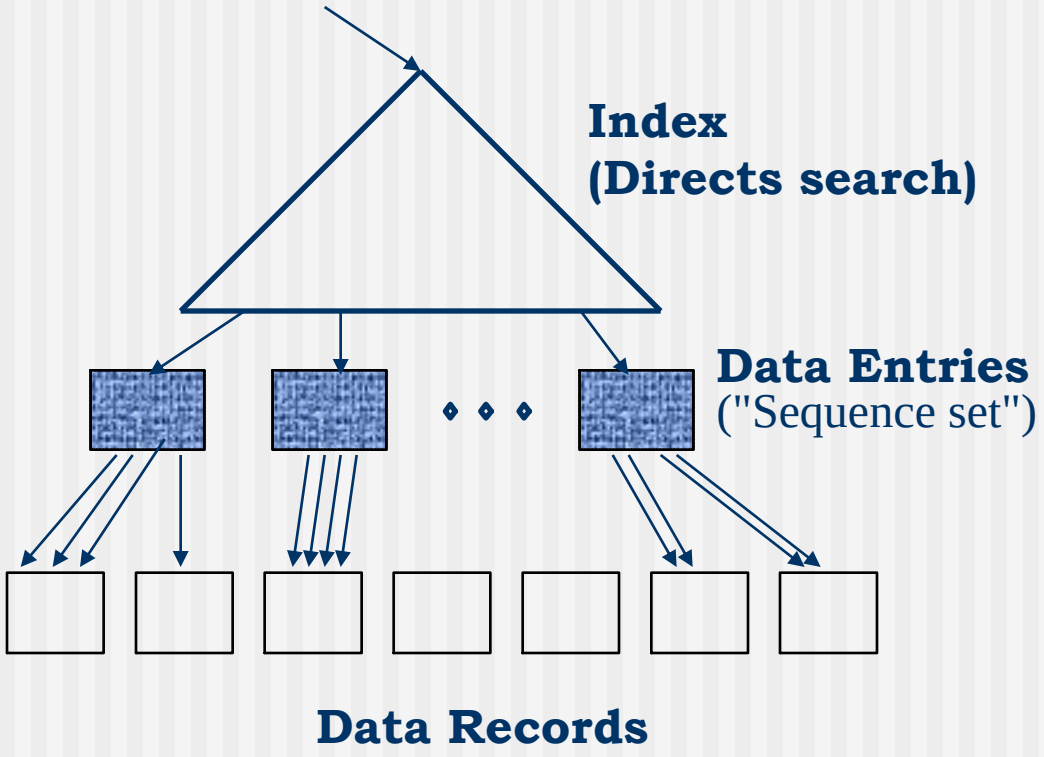
B main memory buffers, k-way merge

Using B+ Trees for Sorting

- Scenario: Table to be sorted has B+ tree index on sorting column(s).
- Idea: Can retrieve records in order by traversing leaf pages.
- ***Is this a good idea?***
- Cases to consider:
  - B+ tree is clustered → ***Good idea!***
  - B+ tree is not clustered → ***Could be a very bad idea!***

Clustered B+ Tree Used for Sorting

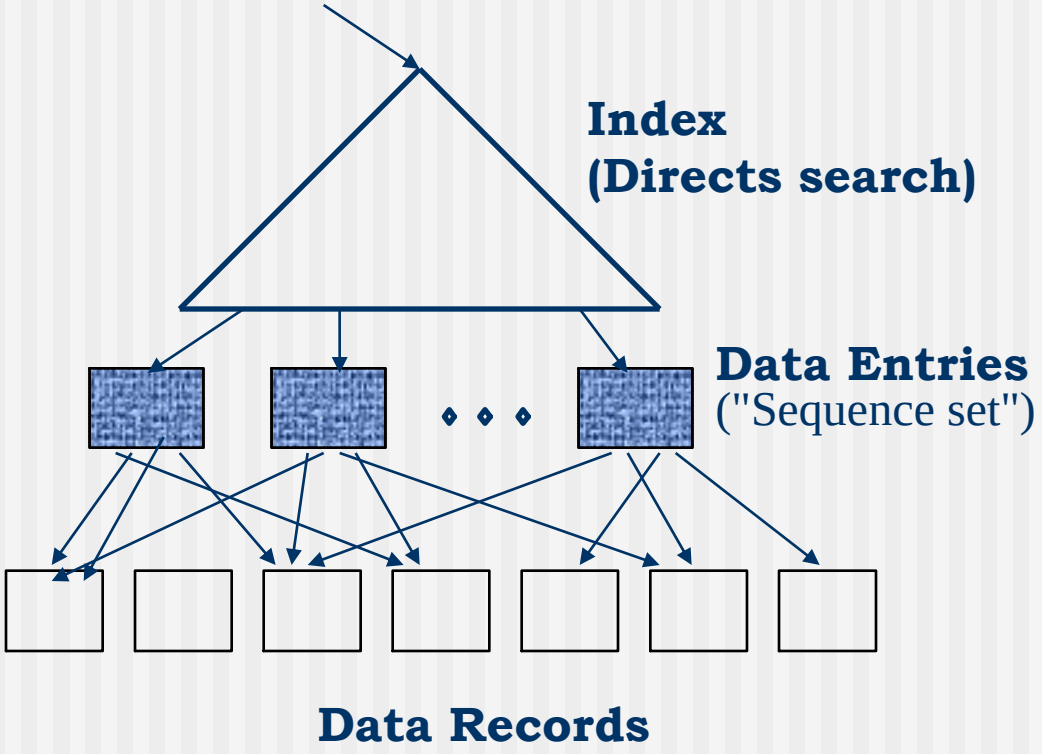
- Cost: root to the left-most leaf, then retrieve all leaf pages (Alternative 1)
- If Alternative 2 is used? Additional cost of retrieving data records: each page fetched just once.



* *Always better than external sorting!*

Un-clustered B+ Tree Used for Sorting

- Alternative (2) for data entries; each data entry contains *rid* of a data record. In general, one I/O per data record!



External Sorting vs. Unclustered Index

N	Sorting	p=1	p=10	p=100
100	200	100	1,000	10,000
1,000	2,000	1,000	10,000	100,000
10,000	40,000	10,000	100,000	1,000,000
100,000	600,000	100,000	1,000,000	10,000,000
1,000,000	8,000,000	1,000,000	10,000,000	100,000,000
10,000,000	80,000,000	10,000,000	100,000,000	1,000,000,000

* ***p*: # of records per page**
* ***B=1,000 and block size=32 for sorting***
* ***p=100 is the more realistic value.***

Sorting Records!

- Sorting has become a blood sport!
  - Parallel sorting is the name of the game ...
- Datamation: Sort 1M records of size 100 bytes
  - Typical DBMS: 15 minutes
  - World record: 3.5 ***seconds***
    - 12-CPU SGI machine, 96 disks, 2GB of RAM
- New benchmarks proposed:
  - Minute Sort: How many can you sort in 1 minute?
  - Dollar Sort: How many can you sort for $1.00?

Summary

- External sorting is important; DBMS may dedicate part of buffer pool for sorting!
- External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted ***runs*** of size ***B*** (number of buffer pages). Later passes: ***merge*** runs.
  - The number of runs merged at a time depends on ***B***, and ***block size***.
  - Larger block size means less I/O cost per page.
  - Larger block size means smaller number of runs merged.
  - In practice, number of runs rarely more than 2 or 3.

Summary (cont)

- Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Heap/tournament sort: slower (2x), longer runs
- The best sorts are wildly fast:
  - Despite 40+ years of research, we're still improving!
- Clustered B+ tree is good for sorting; unclustered tree is usually very bad.