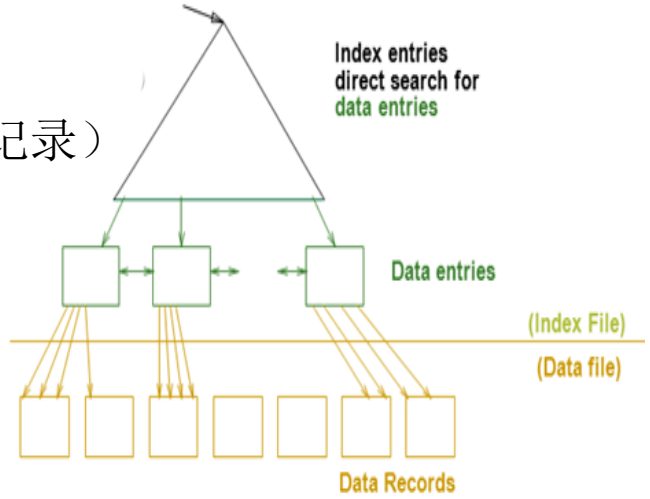
External Sorting
外排序

SUN YAT-SEN UNIVERSITY

Review

- Cost Model（代价模型）
  - 只关心磁盘块的IO次数
- 文件由页组成，而每个页包含一组记录
  - *Record id = <page id, slot #>*
- 索引文件由两部份组成
  1. 数据项部分
     - Data Entry(数据项) ⟺ data record（数据记录）
  2. 引导部份
     - 树索引技术
     - Hash索引
- 索引的 clustered？



Why Sort?

- Data requested in sorted order
  - e.g., find students in increasing *gpa* order
- First step in *bulk loading* B+ tree index.
- Useful for eliminating *duplicates* (Why?)
- Useful for summarizing groups of tuples
- *Sort-merge* join algorithm involves sorting.

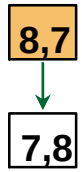
Why External Sorting(外排序)?

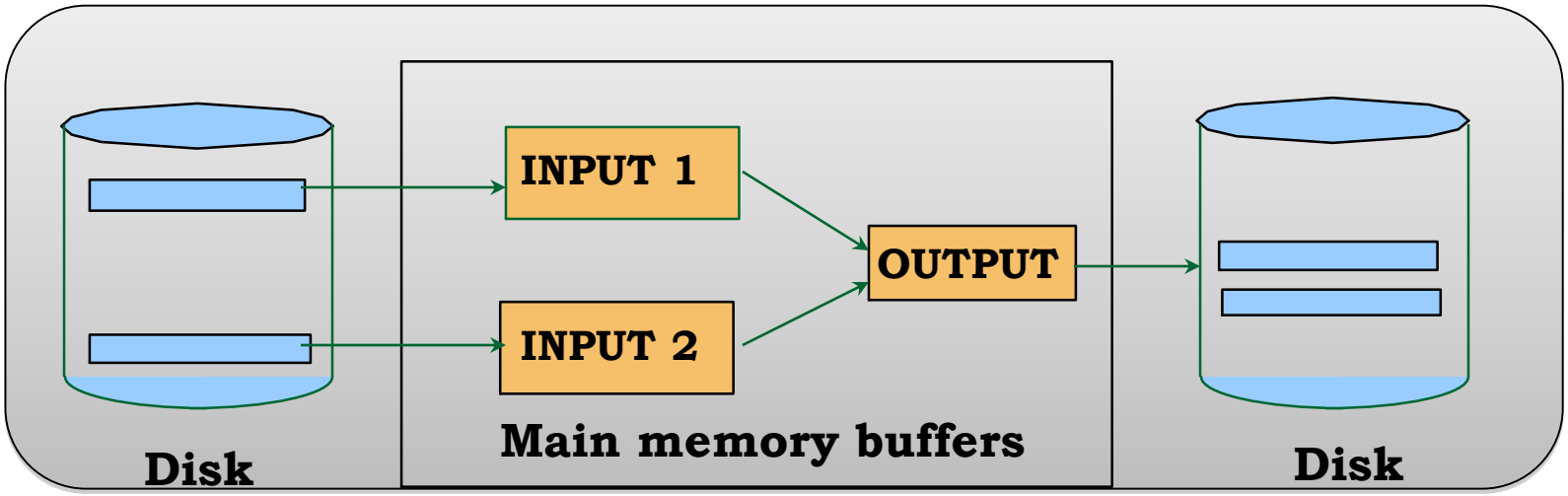
- Problem: sort 100Gb of data with 1Gb of RAM.
  - why not virtual memory?

- Idea
  - 方法:对数据进行多轮处理,
  - 效果:从而可以使用较少内存排序庞大的数据集

Two-Way External Merge Sort: Requires 3 Buffers
两路归并外排序

- Pass 0(生成有序段/子文件): Read a page, sort it, write it.
  - each sorted page (or subfiles) is called a run(有序段、归并段).
  - only 1 buffer page is used.
- Pass 1, 2, 3, …, etc.(合并有序段):
  - merge pairs of runs into runs twice as long 新的有序段长度加倍
  - 3 buffer pages used





Merging Runs(合并有序段)

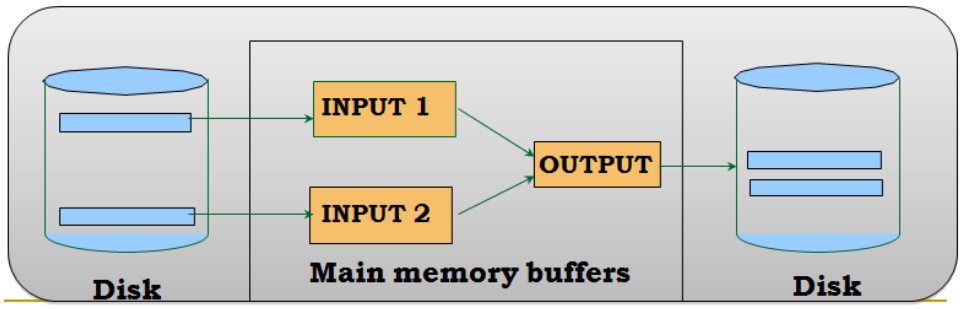
Two-Way External Merge Sort: Requires 3 Buffers
两路归并外排序

- Each pass(每轮) we read + write each page in file.
- N pages in the file => the number of passes

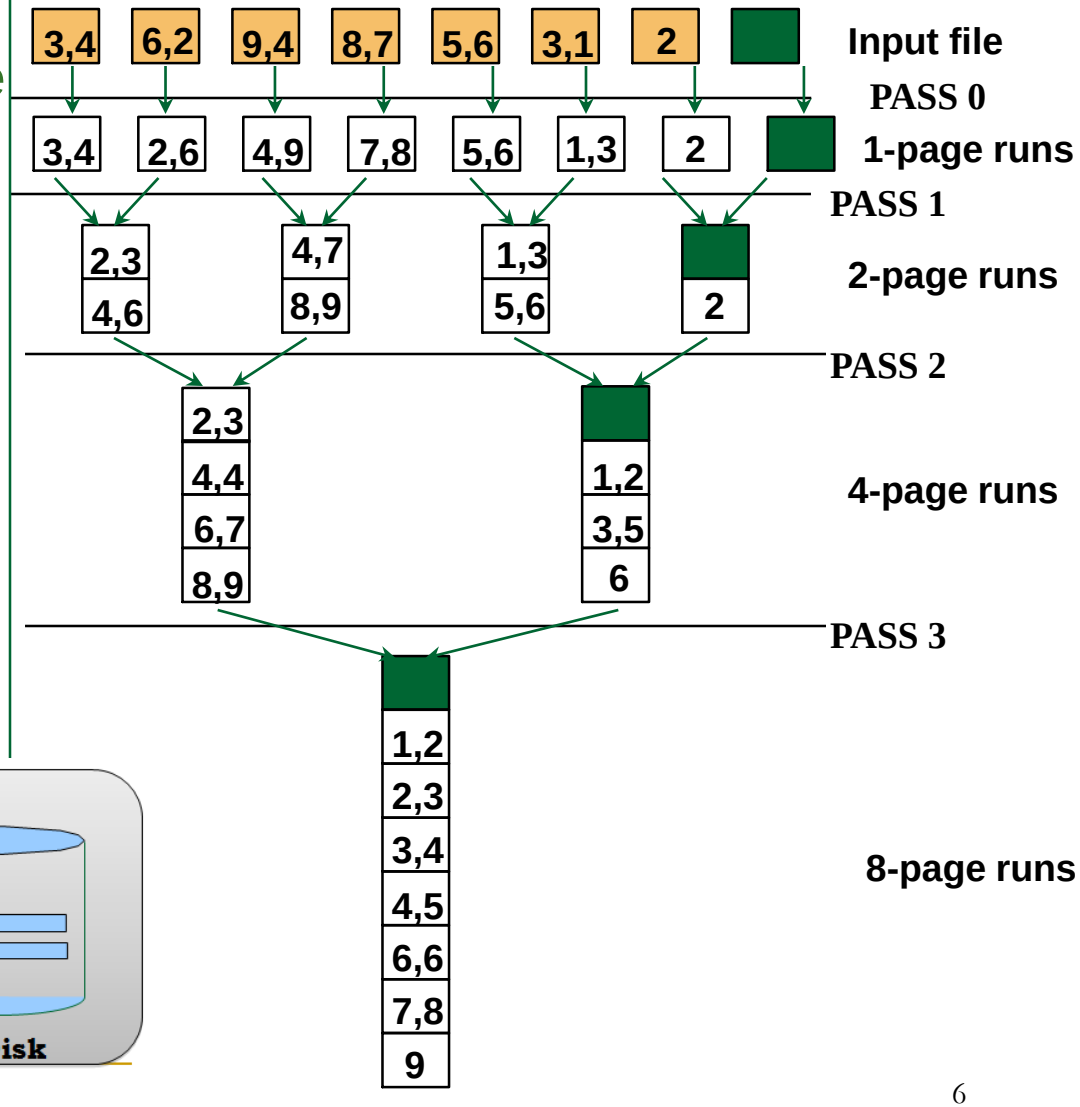
$$= \lceil \log_2 N \rceil + 1$$

- So total cost is:

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$



Merging Runs(合并有序段)

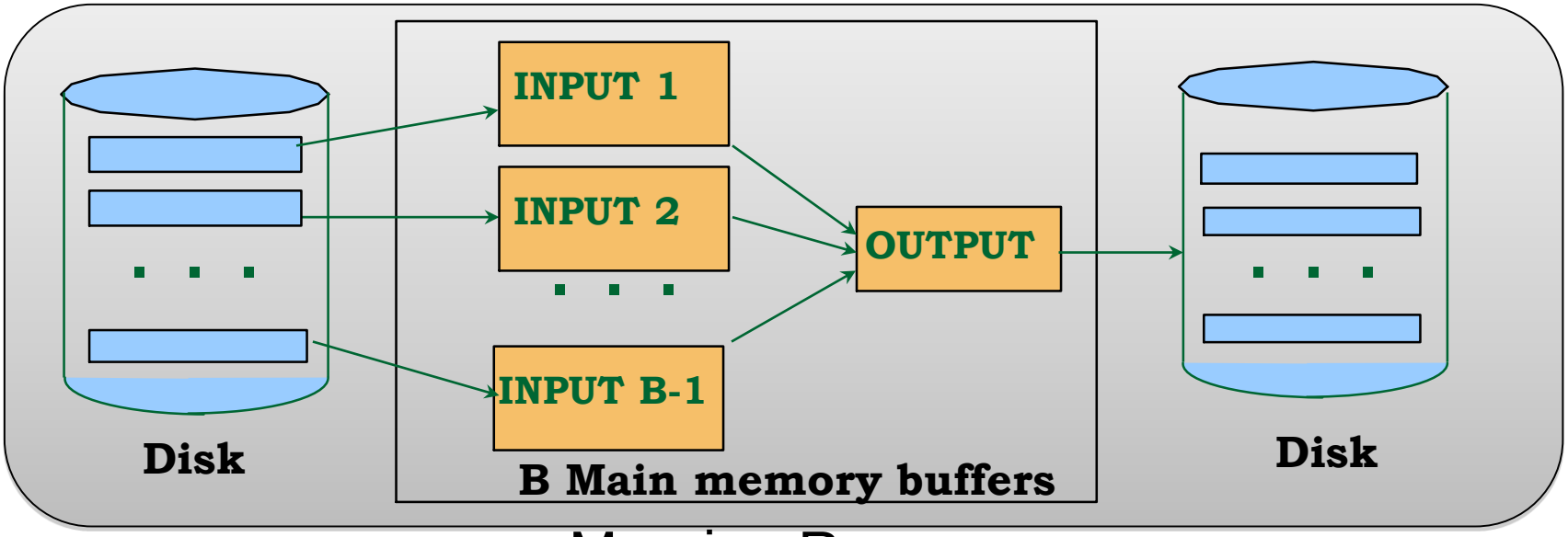


General External Merge Sort
外归并排序

$$2N\left(\lceil \log_2 N \rceil + 1\right)$$

More than 3 buffer pages. How can we utilize them?

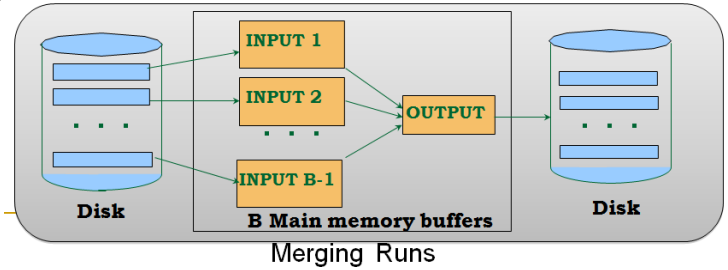
- To sort a file with *N* pages using *B* buffer pages:
  - Pass 0: use *B* buffer pages. Produce $\lceil N / B \rceil$ sorted runs of *B* pages each.
  - Pass 1, 2, …, etc.: merge *B-1* runs.



Merging Runs

Cost of External Merge Sort $2N(\lceil \log_2 N \rceil + 1)$

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



Merging Runs

Formula check: $\lceil \log_4 22 \rceil = 3 \ldots + 1 \rightarrow$ 4 passes √

Number of Passes of External Sort

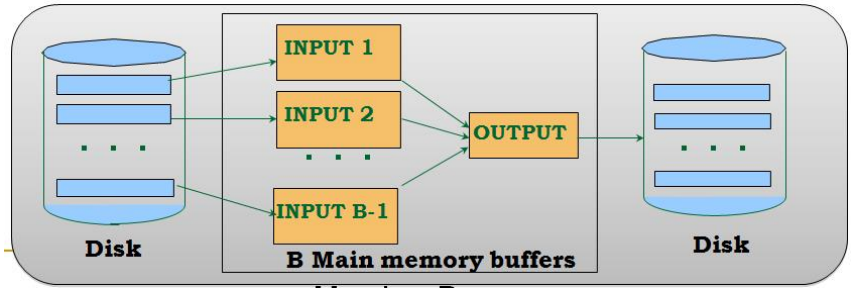
$$1+\lceil \log_{B-1}\lceil N/B \rceil \rceil$$

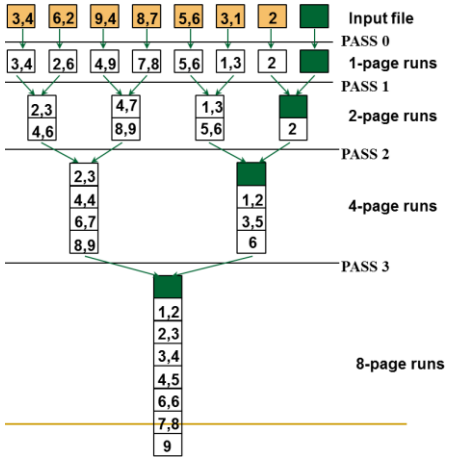
- 缓冲区越多, 轮数越少
- (I/O cost is 2N times number of passes)

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

In practice, most files sorted in 2-3 passes

Blocked I/O for External Merge Sort
改进1: 块I/O --连续读





- In fact, read a *block (chunk)* of pages sequentially!
- Suggests we should make each buffer (input/output) be a *block* of pages (如 32 pages).
  - But this will reduce fan-in during merge passes!
    减少归并段数

$$1+\lceil \log_{B-1}\lceil N/B \rceil \rceil$$

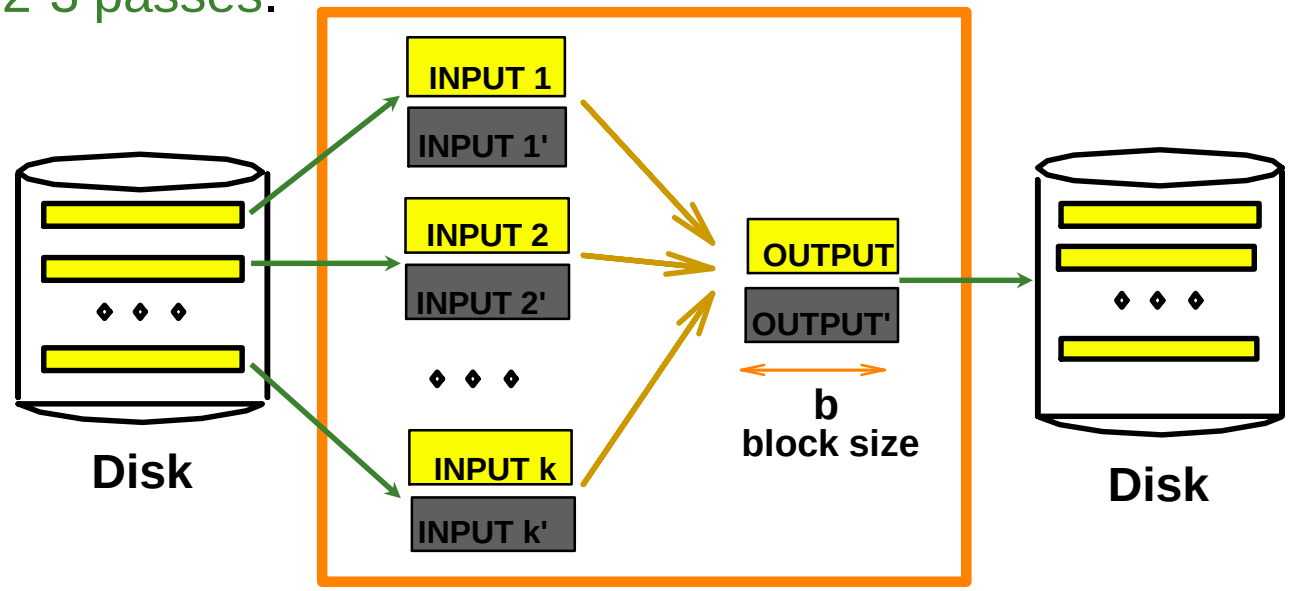
  - In practice, most files still sorted in 2-3 passes.

Double Buffering
改进2: 双缓冲 -- CPU与I/O重叠

- Goal: reduce wait time for I/O requests during merge
- Idea: 2 blocks RAM per run, disk reader fills one while sort merges the other

$$1+\lceil \log_{B-1} \lceil N/B \rceil \rceil$$

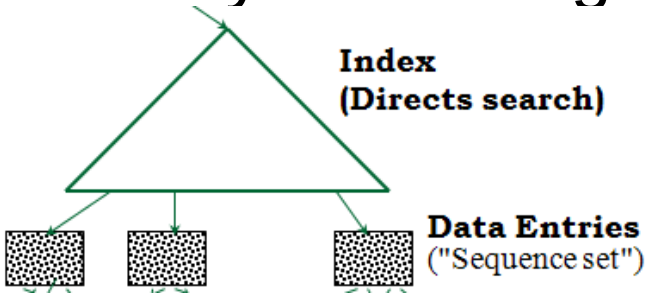
  - Potentially, more passes; in practice, most files *still* sorted in 2-3 passes.



B main memory buffers, k-way merge

Using B+ Trees for Sorting
改进3:不排序

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

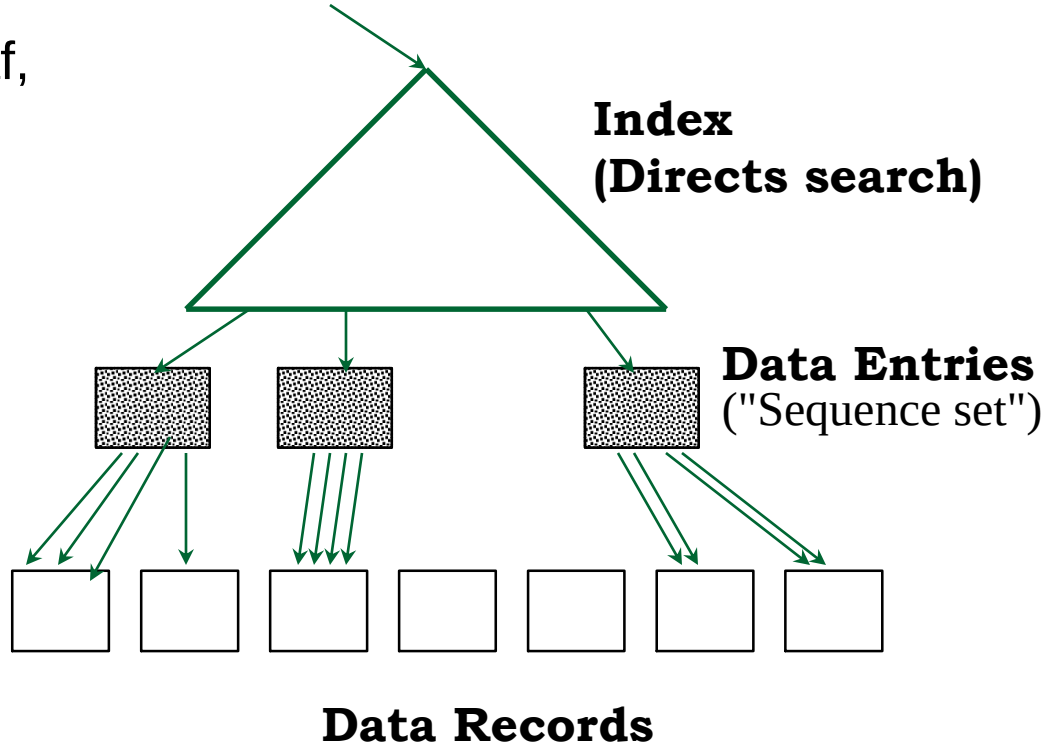


Good idea!

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!

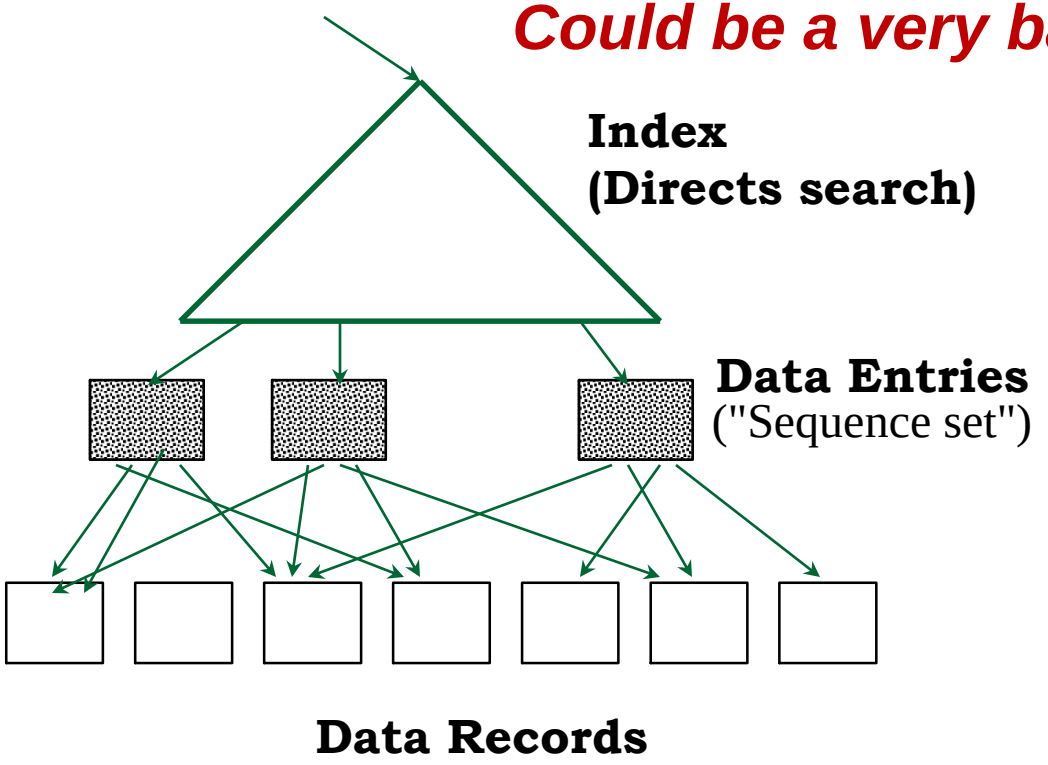
Good idea!

Unclustered 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!

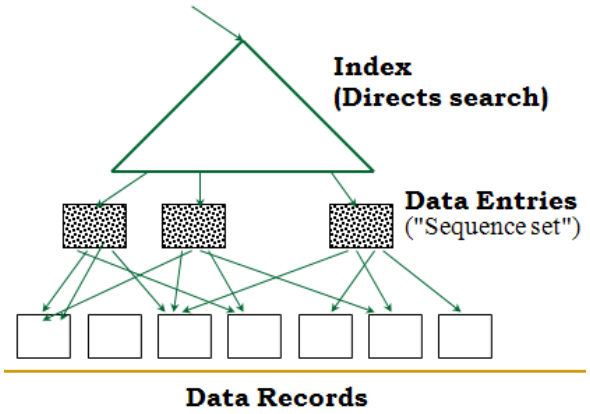
Could be a very bad idea!



Cost of External Sorting vs. Unclustered Index

***N*: number of pages of records**

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



$$1+\lceil \log_{B-1}\lceil N/B \rceil \rceil$$

B=1,000 and block size=32 for sorting

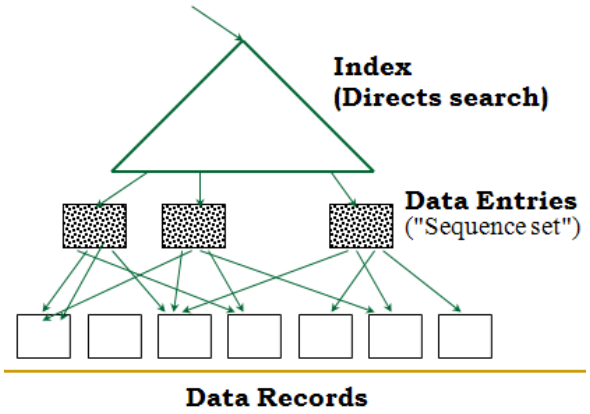
Cost of External Sorting vs. Unclustered Index

Could be a very bad idea!

one I/O per data record!

***N*: number of pages of records**

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



$$1+\lceil \log_{B-1}\lceil N/B \rceil \rceil$$

B=1,000 and block size=32 for sorting

***p*: number of records per page**

p=100 is the more realistic value.

Summary

- External sorting is important
- External merge sort minimizes disk I/O cost:
  - Pass 0: Produces sorted ***runs*** of size ***B*** (# buffer pages).
  - Later passes: ***merge*** runs. $1+\lceil \log_{B-1}\lceil N/B \rceil \rceil$
    - # 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 # runs merged.
  - In practice, # of runs rarely more than 2 or 3.

Summary, cont.

- Choice of internal sort algorithm may matter:
  - Quicksort: Quick!
  - Heap/tournament(树形选择排序) sort
- Clustered B+ tree is good for sorting; unclustered tree is usually very bad.

- 要求:
  - 掌握外排序算法
  - 理解并记住外排序IO开销的基本公式 $1+\lceil \log_{B-1} \lceil N/B \rceil \rceil$

Cost = 2N * (# of passes)