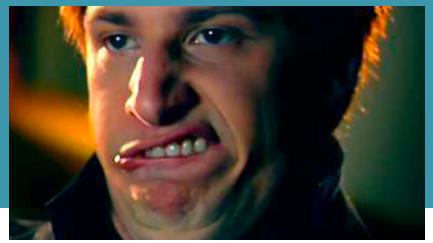
External Sorting

How to sort very large datasets

Chapter 13

Why Sort?



- User wants query answers in some order, e.g. decreasing order of salary

```
SELECT e.name, e.salary
FROM Employees E
ORDER BY e.salary DESC
```

- First step to bulk-loading B+ Tree
- Eliminate duplicate records
  - `SELECT DISTINCT`
- Sort-merge join algorithm (later)

Why Bother in 484?

- But we already know how to sort from 281 ☺

```
DEFINE FASTBOGOSORT(LIST):
    // AN OPTIMIZED BOGOSORT
    // RUNS IN O(N LOG N)
    FOR N FROM 1 TO LOG(LENGTH(LIST)):
        SHUFFLE(LIST):
        IF ISSORTED(LIST):
            RETURN LIST
    RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
```

- New Problem:
  - How to sort 1 TB of data with 10 GB RAM?
  - External Sort – Minimize disk access cost

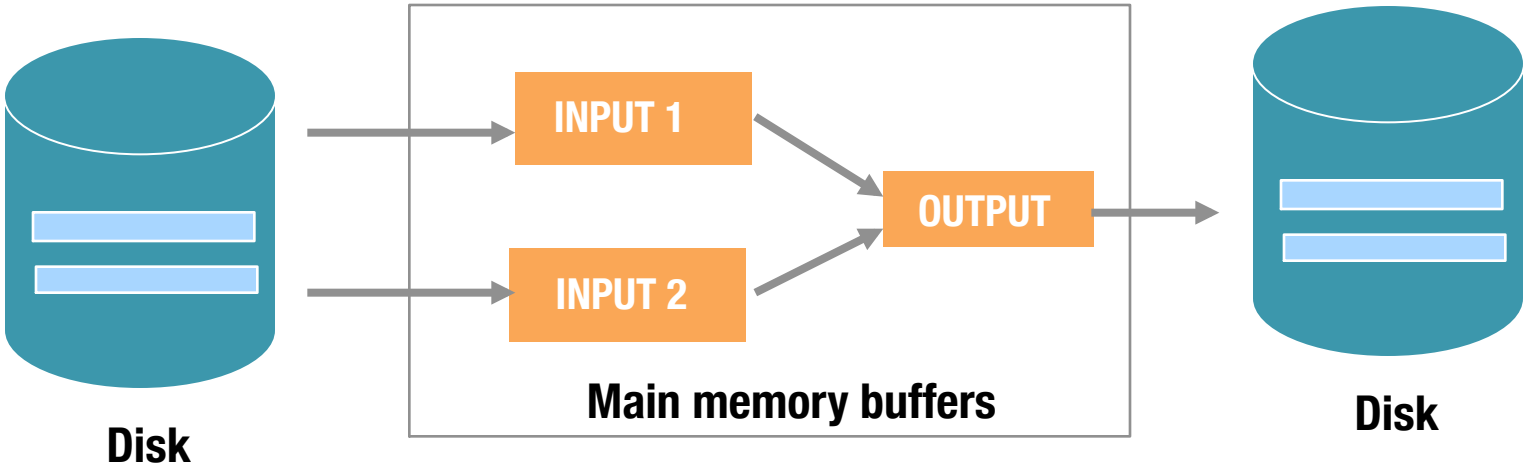
Sorting Records: Contest

Sortbenchmark.org

- Results from 2016:
- Gray Sort: How many Terabytes/min while sorting 100TB?
  - Daytona: Tencent Sort/Tencent Corp (44.8 TB/min)
  - Indy: Tencent Sort/Tencent Corp (60.7 TB/min)
- Cloud Sort: How many dollars/TB to sort on a public cloud?
  - Daytona: NADSort/Nanjing University, Databricks, Alibaba ($1.44/TB)
  - Indy: NADSort/Nanjing University, Databricks, Alibaba ($1.44/TB)
- Penny Sort: How much can you sort for a penny? (since 2011)
  - Daytona: Psort/Univ of Padova, Italy (286GB)
  - Indy: Psort/Univ of Padova, Italy (334GB)

Two-Way External Merge Sort

- Pedagogical use only!
- Requires 3 buffer pages
- Pass 0: Read a page, sort it, write it (a run = sorted sequence)
  - only one buffer page is used
- Pass 1, 2, …, etc.:
  - three buffer pages used

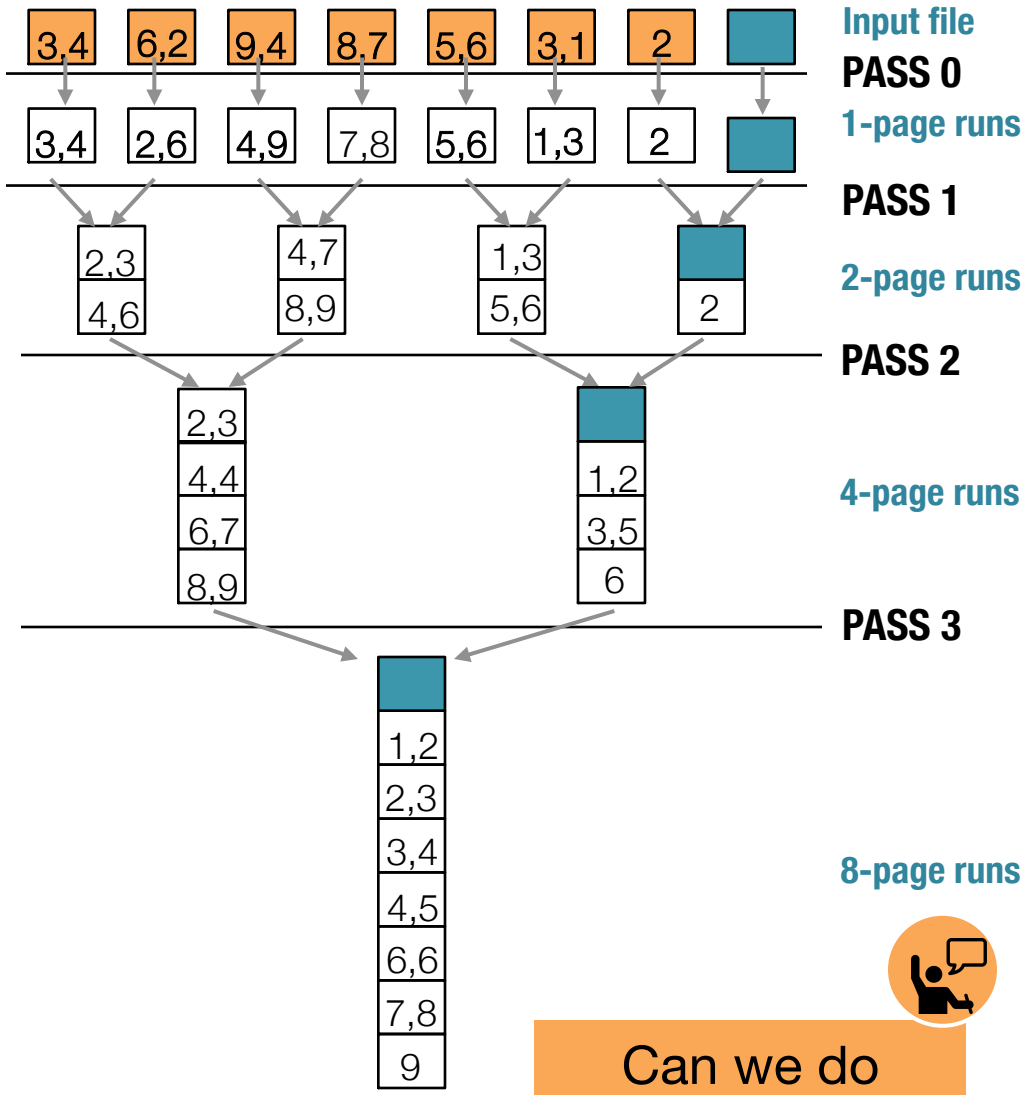


Two-Way External Merge Sort

- Read & write entire file in each pass
- Divide and conquer
- #of passes: $1+\lceil \log_2 N \rceil$
- Cost (# of page I/Os):

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

- In this example:
  - 56 page I/Os
- (Dark pages show what would happen if there were 8 pages)



Can we do better?

How to Make External Sorting Faster?

1. Reduce # of passes

- Using more buffers at each step
- Using replacement sort at Step 0

2. Make each pass cheaper

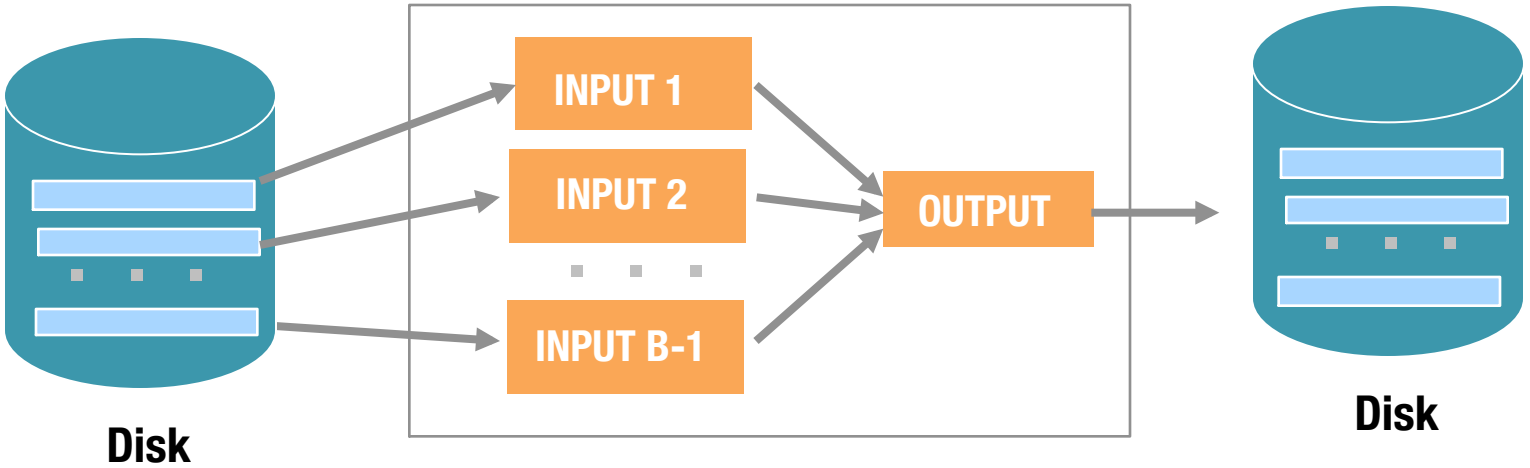
- Blocked I/O
- Double Buffering

3. Not use External Sort

- Clustered B+ tree if available
- Unclustered B+ tree if available and cheaper

General External Merge Sort

- Sort a file with N pages using B buffer pages:
  - Pass 0: use B buffer pages and sort them internally, producing $\lceil N/B \rceil$ sorted runs (each B-page long)
  - Pass 2, 3, …: merge B-1 runs, using (B-1)-way merge



Cost of External Merge Sort

- Number of passes: $1+\lceil \log_{B-1} \lceil N/B \rceil \rceil$
- Cost = 2N * (# of passes)

Example: With 11 buffer pages, how many page I/Os are needed to sort a 1000-page file?

- Pass 0: ⌈ (1000/11) ⌉ = 91 runs of 11 pages each (Note: the last run is 10 pages)
- Pass 1: ⌈ 91/10 ⌉ = 10 sorted runs of 110 pages each (last run is only 10 pages)
- Pass 2: ⌈ 10/10 ⌉ = 1 sorted run of 1000 pages

Sorted File in 3 passes
Total I/O: 2000 pages/pass or 6000 pages

Number of Passes of External Sort

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

N (# of pages)	B=3	B=17	B=257
100	7	2	1
10,000	13	4	2
1,000,000	20	5	3
10,000,000	23	6	3
100,000,000	26	7	4
1,000,000,000	30	8	4

32K pg size, 32TB relation

@1ms page/sec, 2 * 30 * 1000M * 1ms = 694 days!

@1ms page/sec, 2222 hours = 92 days!

Discussion

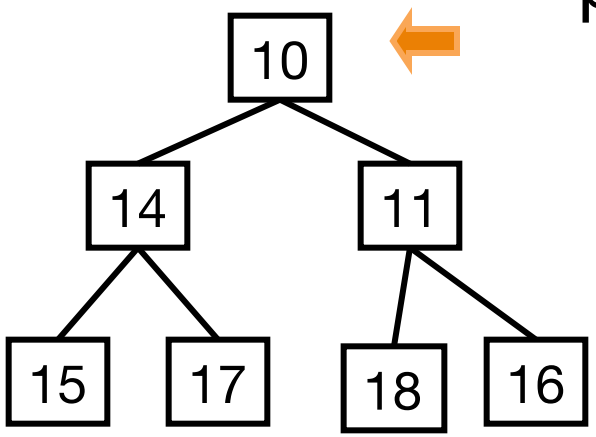
- Is 2-way merge sort a special case of external merge sort?

- Why do we need a dedicated output buffer in Pass 1, 2, ... but not in Pass 0?

Reducing Number of Initial Runs

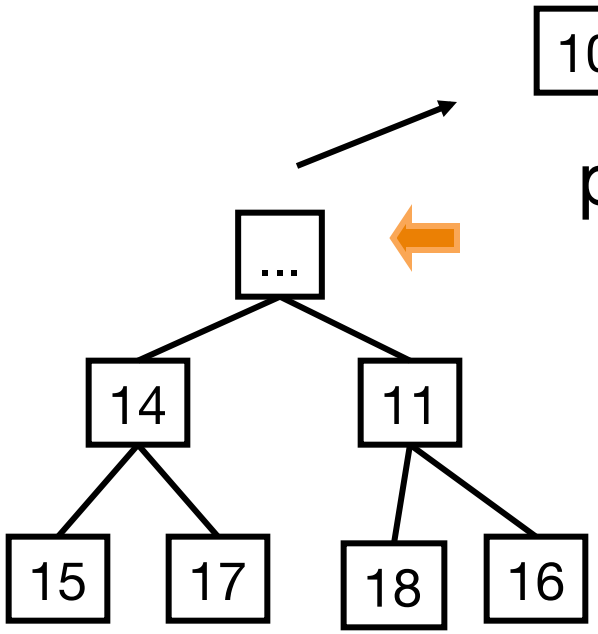
- In the basic scheme, with B memory pages, you will produce approx. N/B runs initially because you will sort B pages, creating a run.
- The eventual sorting cost is lower if we have fewer runs
- Can we produce fewer runs?
- Yes! Replacement sort can produce runs longer than B pages on the average!
  - Longer runs may mean fewer passes (less I/O)

Reminder: Heapsort



pick smallest, write to output buffer:

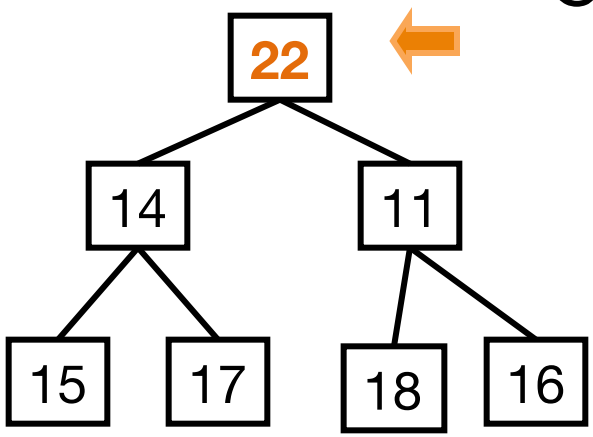
Reminder: Heapsort



pick smallest, write to output buffer:

Reminder: Heapsort

Output Buffer: 10

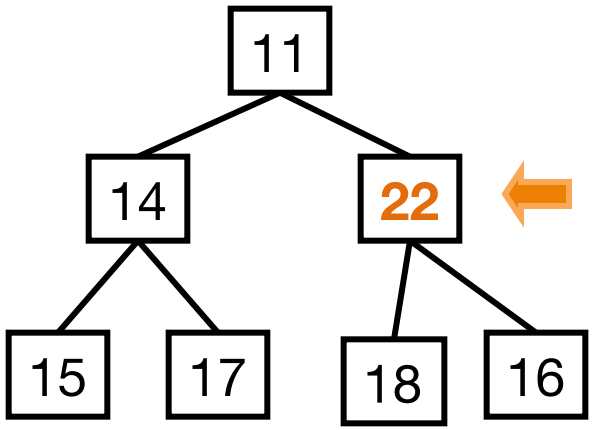


Get next key, put at top and ‘sink’ it

Reminder: Heapsort

Output Buffer: 10

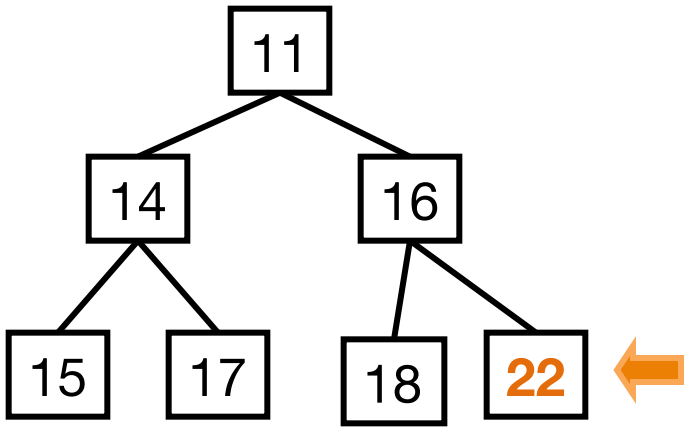
Get next key, put at top and ‘sink’ it



Reminder: Heapsort

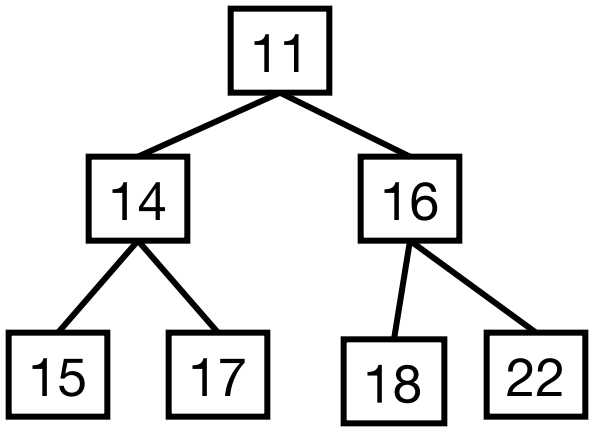
Output Buffer: 10

Get next key, put at top and ‘sink’ it



Reminder: Heapsort

Output Buffer: 10 …



When done, pick top (= smallest) and output it, if ‘legal’ (i.e. >=10 in our example)

This way we can keep on reading new key values (beyond the B ones of quicksort)

Replacement Sort (for Pass 0 only)

- One page each is dedicated as an input buffer and an output buffer
- Remaining B-2 (call it M) pages are called the current set

Input Buffer	Current Set	Output Buffer
13	2	3
5	7	6
	10	
	20	
	30	
	40	

Replacement Sort (for Pass 0 only)

- Start by reading a page from file into the input buffer
- Copy records from input buffer to current set
- Repeatedly pick smallest value from current set that is greater than largest value in output buffer
  - Write to output buffer (run). If buffer full, output
- Start a new run when no value in current set is larger than all values in output

On average, produces runs of size 2M, i.e. 2*(B-2) pages.

Example – Internal Sort Algorithm Replacement Sort

M = 2 pages, 2 tuples per page

Input Sequence: 10, 20, 30, 40, 25, 35, 9, 8, 7, 6, 5, …

		Current set: 10, 20, 30, 40
Output 10,	Read 25	Current set: 20, 25, 30, 40
Output 20,	Read 35	Current set: 25, 30, 35, 40 Flush output
Output 25,	Read 9	Current set: 9, 30, 35, 40
Output 30,	Read 8	Current set: 8, 9, 35, 40 Flush output
Output 35,	Read 7	Current set: 7, 8, 9, 40
Output 40,	Read 6	Current set: 6, 7, 8, 9

Flush output. Start new run. In memory…

On Disk: 10, 20, 25, 30, 35, 40

Average length of a run in replacement sort is 2M Pages

Quiz

Assume:

- We are using the general external sort algorithm
- We have 32 buffer pages
- We use replacement sort

Then, what will be the largest file, in terms of number of pages, that we can sort in two passes?

Ans: *# of passes = 1+ log* $_{B-1}$*(ceiling(N / (2B-4))) = 2*

➔log_{31} (*ceiling*(*N/60*))=1➔*ceiling*(*N/60*)=31➔*N*≤60*31=1860

How to Make External Sorting Faster?

1. Reduce # of passes

- Using more buffers at each step
- Using replacement sort at Step 0

2. Make each pass cheaper

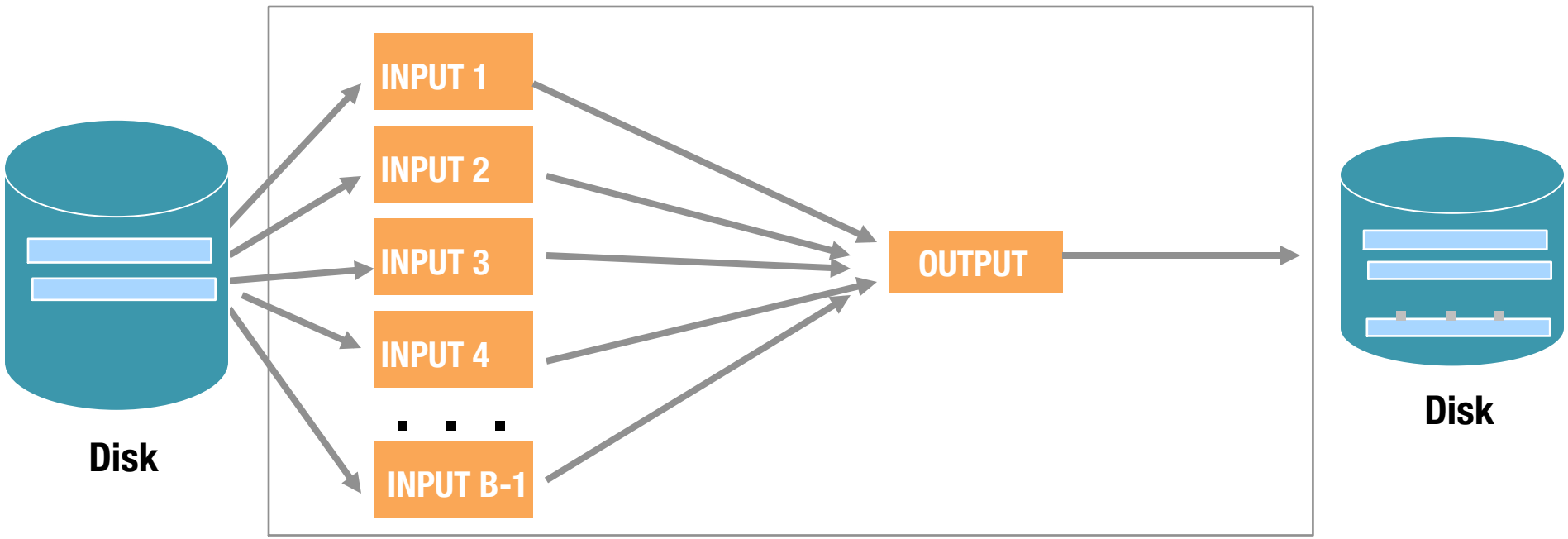
- Blocked I/O
- Double Buffering

3. Not use External Sort

- Clustered B+ tree if available
- Unclustered B+ tree if available and cheaper

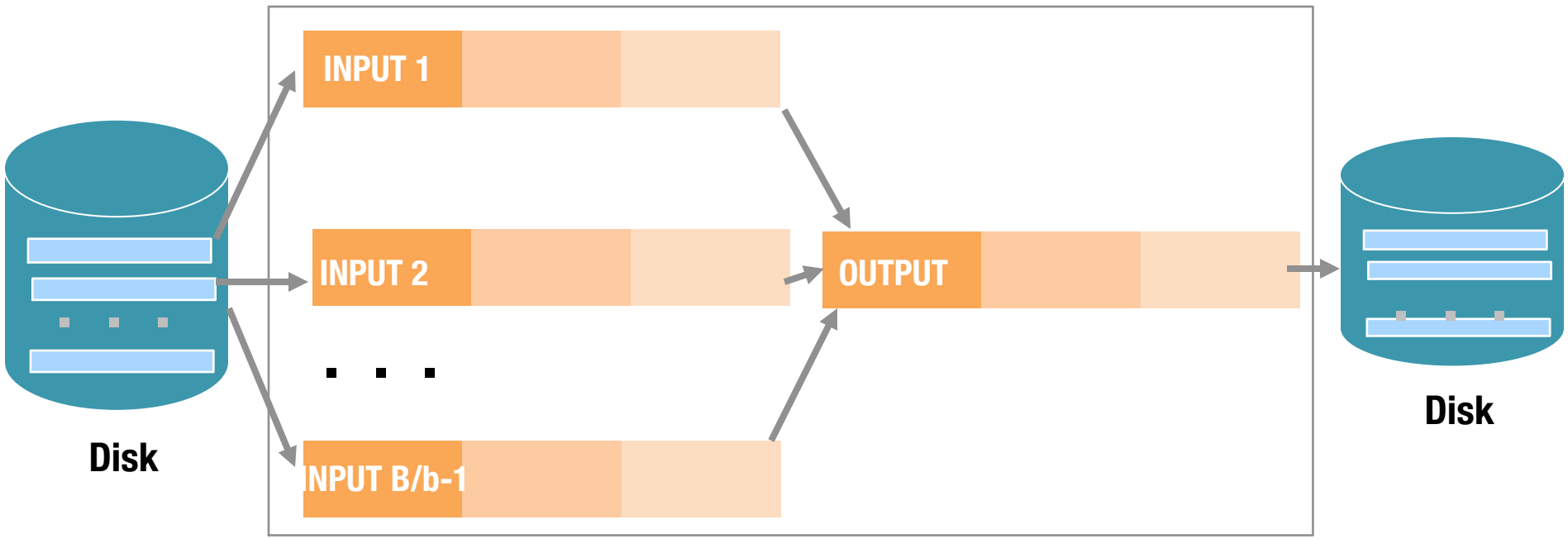
Blocked I/Os

- Single request to read a block of pages often cheaper than independent requests for each page – Why?
- **Normally**: B buffers of size 1KB



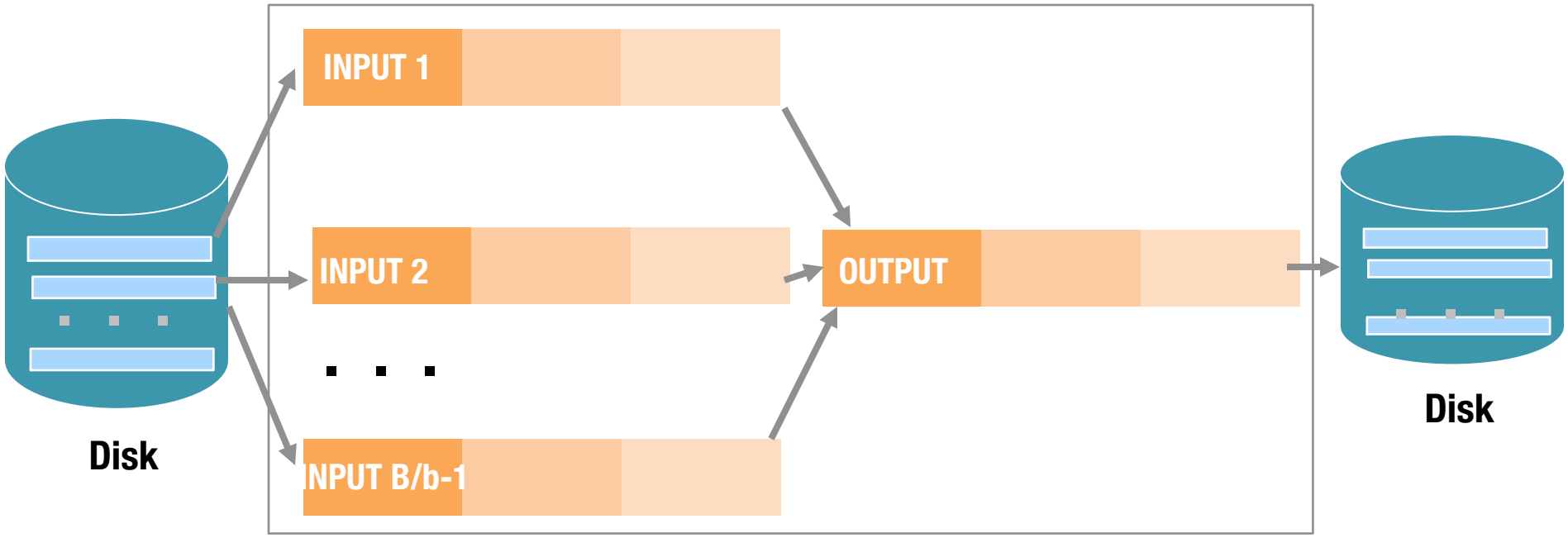
Blocked I/Os

- Single request to read a block of pages often cheaper than independent requests for each page – Why?
- **Alternatively**: B/b buffers of size b KB



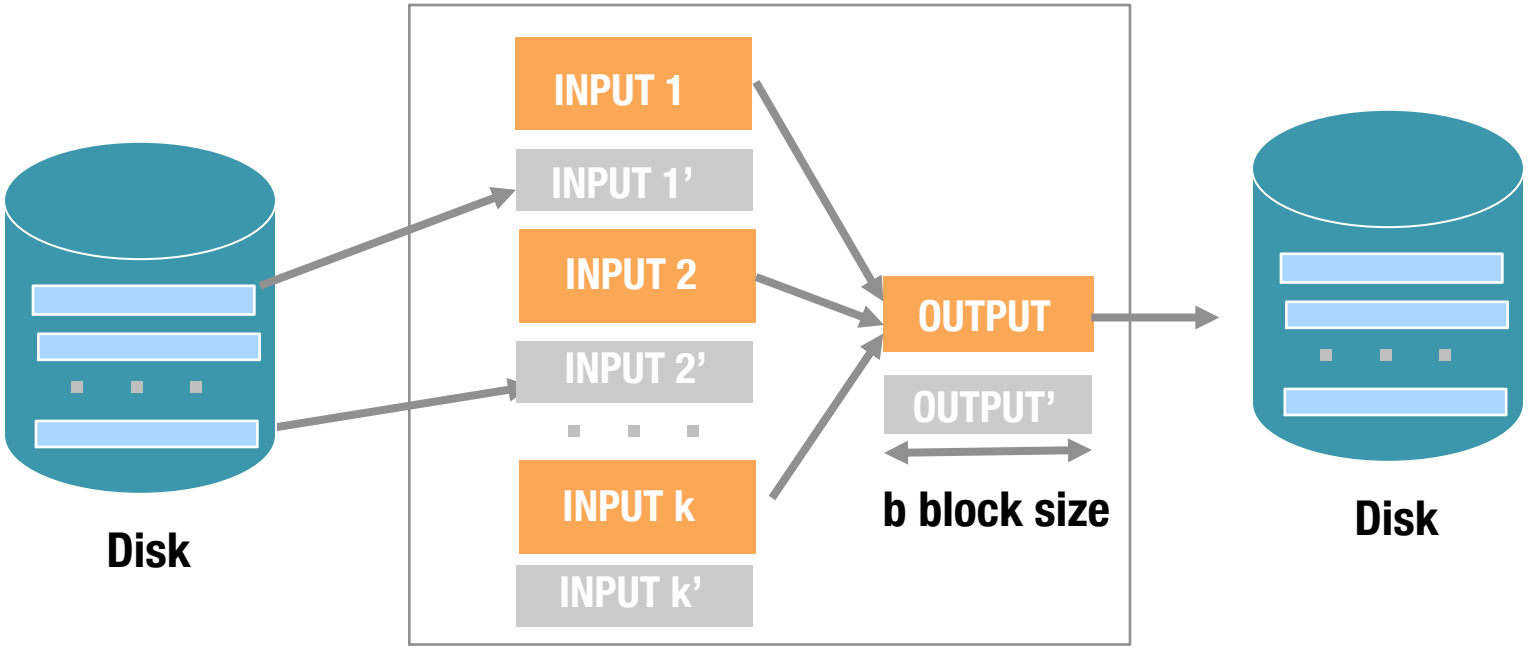
Blocked I/Os: Pros & Cons

- (+) Reduces cost per I/O, because more accesses are sequential
- (-) The fanout is lower and thus # of passes could increase
- This can be cheaper or more expensive: must do the math
- In practice, often 2-3 passes are sufficient



Double Buffering

- Overlap CPU and IO processing
- Prefetch into shadow block
  - Potentially, more passes; in practice, 2-3 passes



2*(K+1)*b main memory buffers, k-way merge

Quiz

- We are using the general external sort algorithm
- We have 32 buffer pages
- We **do not** use replacement sort
- We **do not** use double buffering
- We use the blocking scheme with 10-page blocks

What will be the largest file, in terms of number of pages, that we can sort in two passes?

Ans: *# of passes* $= \log_{floor(B/b)-1}(ceiling(N/B)) + 1 = 2$

$B=32,\ b=10,\ 1+ \log_2(ceiling(N / 32)) = 2$

➔$\log_2(ceiling(N/32))=1$➔$ceiling(N/32)=2$➔$N \leq 32*2= 64$

How to Make External Sorting Faster?

1. Reduce # of passes

- Using more buffers at each step
- Using replacement sort at Step 0

2. Make each pass cheaper

- Blocked I/O
- Double Buffering

3. Not use External Sort

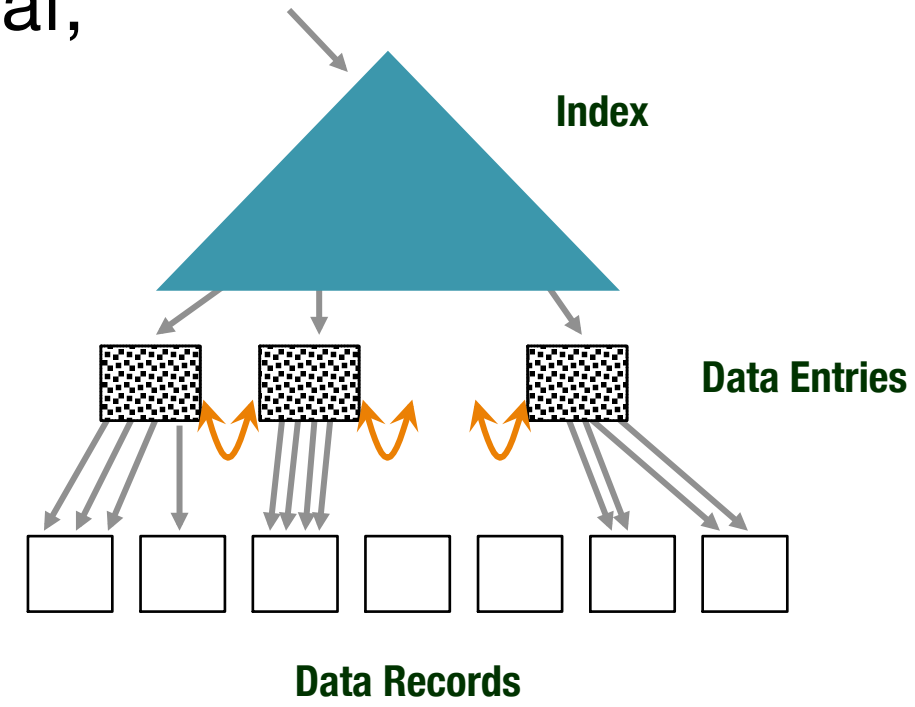
- Clustered B+ tree if available
- Unclustered B+ tree if available and cheaper

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 for Sorting

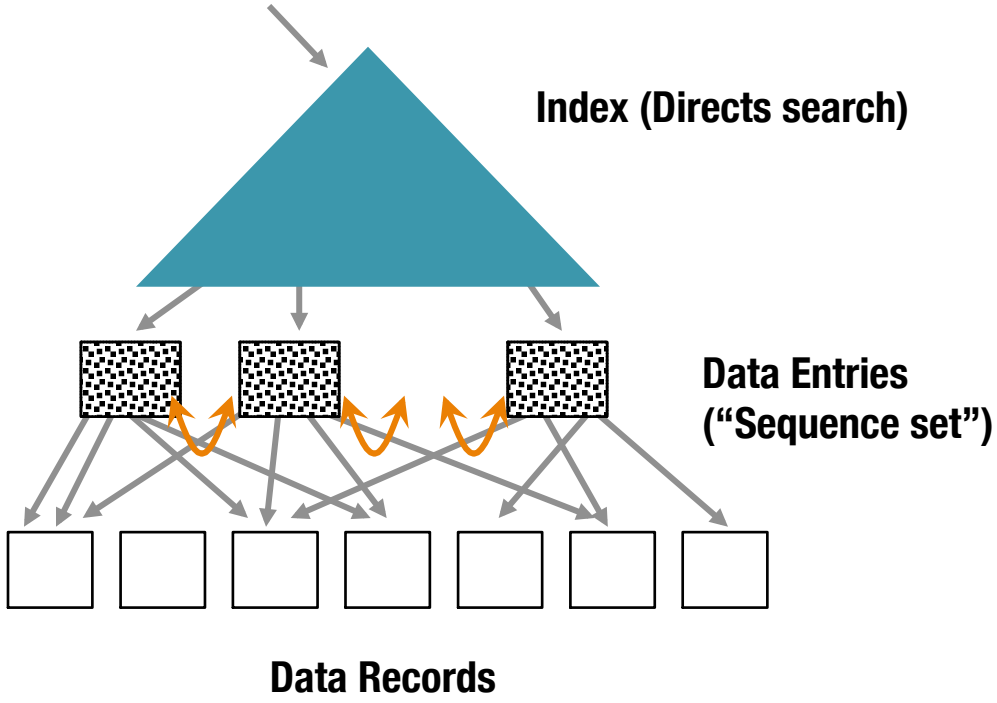
- Go to the left-most leaf, then retrieve all leaf pages
- Alt 1: Done!
  - # pages?
- Alt 2: Retrieving data records, each page fetched just once
- Faster than external sorting!



$$2N\left(1+\lceil \log_{B-1}\lceil N/B\rceil\rceil\right)$$

Unclustered B+ Tree for Sorting

Alternative 2: In general, one I/O per data record!



When can this be useful?

Ans: For extremely selective queries

Cost of External Sorting vs. Unclustered Index

N: # pages
p: avg # of records per page -- p = 100 is realistic
B = 1,000 and block size = 32

N	External Sorting	Using Unclustered B+ Tree		
		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

Summary: External Sort

- Important operation
- Minimize disk I/O cost, use the (large) buffer pool:
  - Larger runs
  - Fewer merges
  - Blocked IOs
  - Double Buffering
- Choice of internal sort algorithm may matter
  - Pass 0: Run size B or 2(B-2)
- Can use indices
  - Clustered Index: Great! Always better than external sort
  - Unclustered Index: Use with caution

Blocked I/Os

- Single request to read a block of pages often cheaper than independent requests for each page – Why?
- Make each buffer a block of q pages instead
  - Reduces cost per page I/O
  - First Pass: Each run 2(B-2) pages, $\lceil N/2(B-2) \rceil$ runs (where B is the size of the buffer pool in #pages)
    - Assuming we use replacement sort optimization…
  - # of runs merged in each pass (fanout): $F = \lfloor (B-q)/q \rfloor$
  - # passes: $\lceil \log_F(\text{\# of runs from first pass}) \rceil + 1$
  - Cons: The fanout is lower and thus # of passes could increase
  - Pros: Each pass doing more efficient I/O
  - This could be cheaper or more expensive – need to do the math
  - In practice, often 2-3 passes are sufficient

Blocked I/Os

- First Pass: Each run 2(B-2) pages, $\lceil N/2(B-2) \rceil$ runs (where B is the size of the buffer pool in #pages)
  - Assuming we use replacement sort optimization…
- # of runs merged in each pass (fanout):

  $F = \lfloor (B-q)/q \rfloor$
- # passes:

  $\lceil \log_F(\text{\# of runs from first pass}) \rceil + 1$
- This could be cheaper or more expensive – need to do the math
- In practice, often 2-3 passes are sufficient