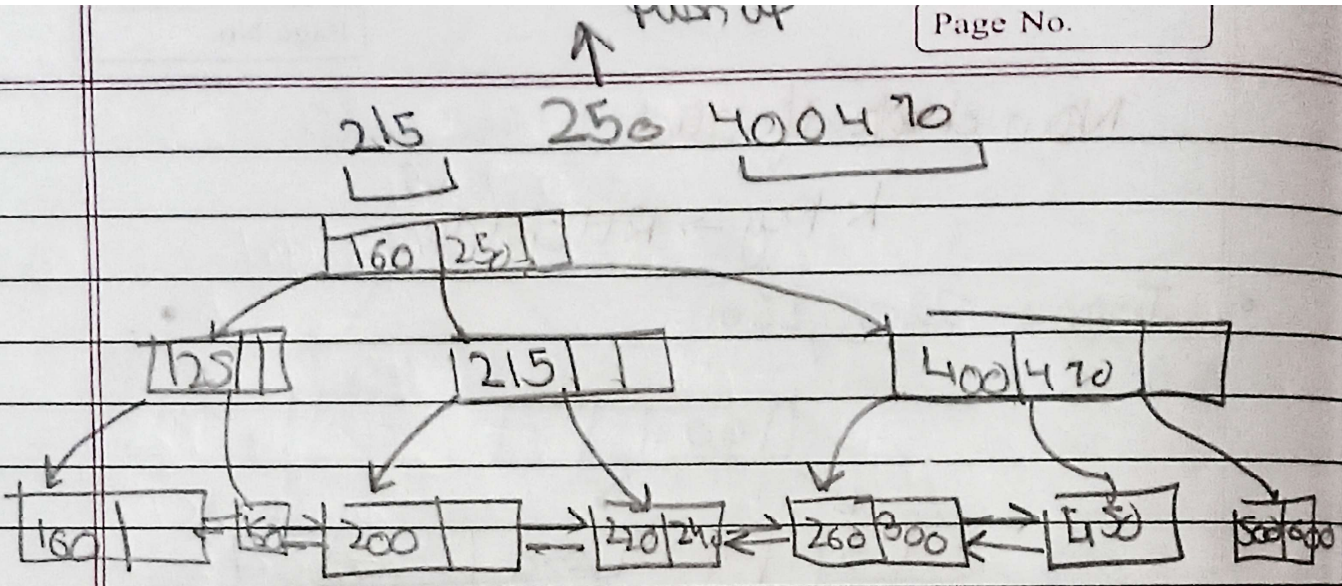
Page No.



d/t
∞

Hash based index
↓
We can quickly tells where

DBMS Deletion

- Simplest
  Not Found ✓ ☺
- If capacity ≥ more than equal → Delete value
  d+1 record ↑ just arranged the key

delete 300
#do BS

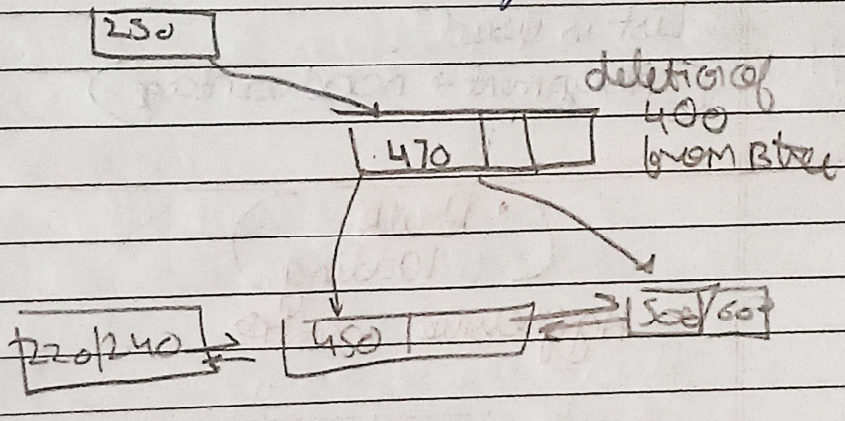
- Whenever have d
  ~~less than~~
  delete 260
  (Note seperators will not effect

Date : / /
Page No.

d → d-1
record

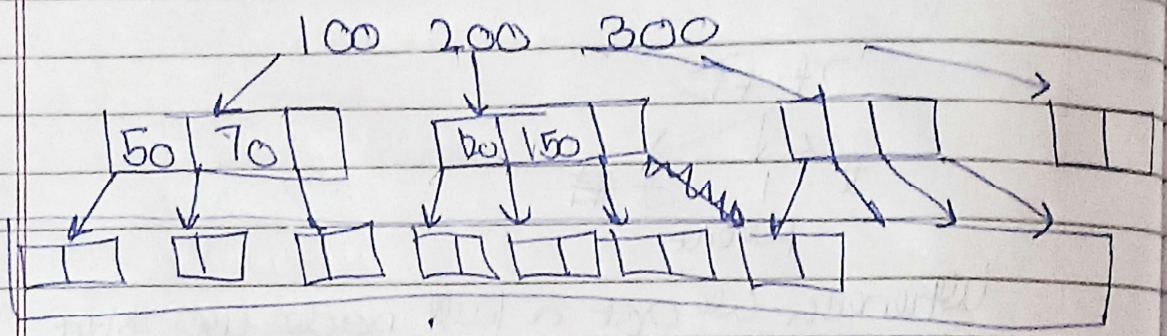
In case
of B tree
it will ask from right or left
neighbour
If both can't give use will
merge

left sibling
and right is at min
we will throw separator
+ merge



Page No.

Deletion from B+ tree



Merge & take down separator
(separator row)

Quiz

Take down separator when we delete a separator

Merging with left
bring down separator again

Hash # equality search
BTree # equality & range search

File Organization

How to represent data in files
char → ascii code fixed
(rule follow)
basic datatypes → fixed

Records → Collection of various Datatypes

Rel → no of columns fixed
rec space for row is fixed then fixed

Date : / /
Page No.

representation

Record → [| | | | .] 20 bytes

A1 A2 A3 AN Record1

attributes

[| | | |] Record2

Sizeof each attribute fixed

if some attributes variable size

#

Better way

after every attribute we indicate by ✡ special value

7th April

CS → Game of Allocations

Date : / /
Page No.

Cols → fix size
All Record → fix size

Record →

fix amount of space we know before hand

Variable lenght → We use some special char to show end of attribute

Linear hashing

We have to go sequentially (disadvantage)

Can be helpful for addr default value

For random access we need ptr (List)

01	Ashwin Anand 00
02	Amit Aseken 302

We can locate in constant time

Extendable hashing

no of slots = no of attributes

If this is always then we wouldn't discuss earlier method

address pt to start

No of attribute fix

not req end adress we will have one more slot → tells where it end

and

Can be helpful for adder default value

e it end

Date : / /
Page No.

unit of memory DBMS would read/ write in 1 time

↑

data → page → file

Page → ID #

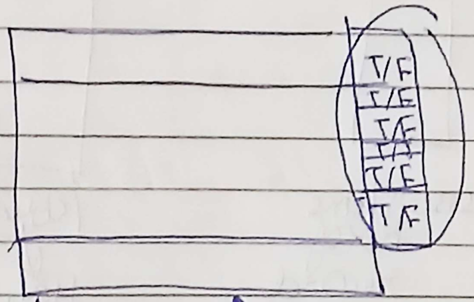
All computer memory are 1D

For convience we see in 2D

- We see mostly Row oriented But Col oriented also possible

Fixed lenght

No Of Records in a page ↓ fix



not better soln

Record size also fix (Each slot Occupied/free)

↑ Book keeping part

tells next page prev page

// Implementing storage

Instead of maintaining we can keep count

(# maintain a directory no of slots fixed

Each slot → occupied / not 2 again)

Deletion ~~Insertion~~ there Problem NO →

We can ~~don't need~~ do shift sequentials else just ~~sea~~ swap

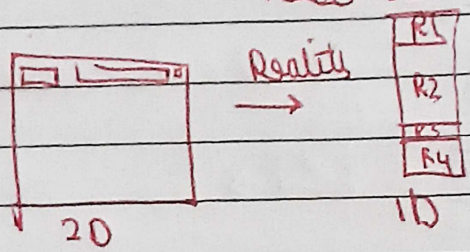
we can move record without Map

Maintain a global mapping

rid | page slot

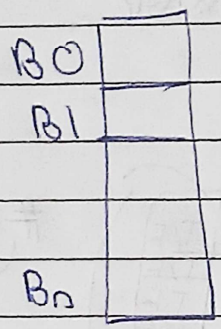
we can move our record around in file

Variable Length Records



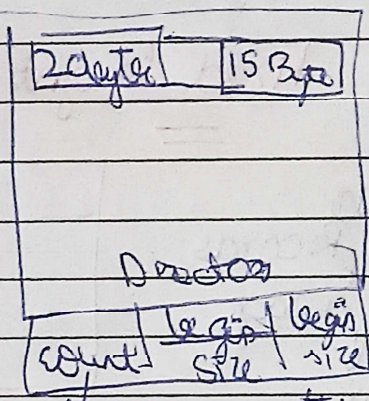
No of attribute fixed

- Record size variable
- No of slots also variable



#linear Hashing

everytime no of record we increase by 1



How many Records

we keep track to used

// same part we are keeping fixed is last
// But Here we are increasing slots directory

we keep total is no of page

total no of record // # total space record

total slots

File Organisation

Heap file
Sorted
Clustered B+ tree (Tree based indexing)
Unclustered B+ tree

(No benifit of sorting on Hash value)

Operations	Scan	Eq Search	Range search	Insert	Delete
Heap	BD	$\frac{1}{2}$ BD	BD	D	Search + D
Sorted	BD	$O(\log_2 B)$	Eq Search + qualifying pages D	Eq search + BD	Search + $O(\log_2 B)$ BD + D
Clustered					
Unclustered B+ Tree					
Unclustered Hash					

Compute cost + I/O cost (// Hard disk reads)

Compute cost → Order of GHz → Time → Order of ns → 10^{-9} sec

I/O cost → 10^{-3} sec → Order of ms → Read sector

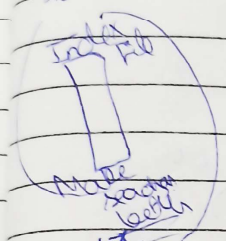
Hard disk // magnetic storage

(• simple
• and very less difference between orders is going only)

We ignore compute cost so

✡ D → amount of time req. by I/O
Anything we read/write

(We are not considering network cost etc (in case of 3rd party)



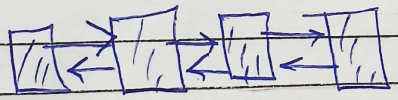
✡ B pages
Go & Read every page

- Organising room

Scan a room // Go through all things
(Shoes/clothes/CS245 book)
(It will completely fetch) Heap file

Cost → O(B*D)

- Eq Search



Still go all We can't do better

worst → Complete Scan

Avg → $\frac{1}{2}BD$ sometime early

O(whole size of file)

- Range Search

//CPI range search
again scan whole data

- Insert O(2D)

D → Fetch in last page + put it back

We just pick up last page + throw

3D // incase of page full someone would say create new ok

This doesn't + Math

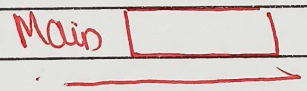
- Delete

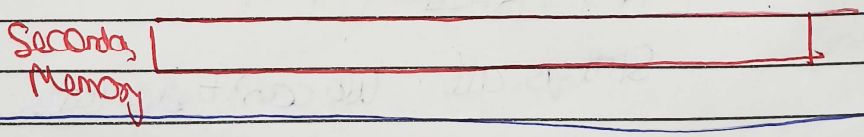
↓

Time for search + D

(locate that record) (take it back)

Assumption

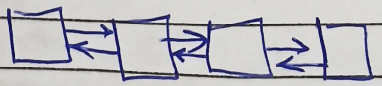




- Sorted

// based on any attribute

Sorted → BD search



Equality search : // Binary search

min value in that page

max value is that page

// max and min value of data
// along with address

// Multiway Sortin-

B·Search
with page → compute cost

D($\log_2 B$) ← Eq Search
Inside page // D

D → Each page
$\log_2 B$ → no of time we have to cross pa

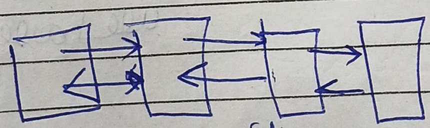
Range search (// We can locate end but still have to go & send all b/w)
↳ Eq Search + How many Extra IO's req? → depends on range
How many page

D($\log_2 B$) + (the no of qualifying pages) D → Each seein, Can it be

100 – 200
If 100 not there
IOS
we will have lower IOS

Insert →

Eq search + B · D ← // Shiftin
if full push → no of IO's



if full we will send records to next page & next page data of more seg

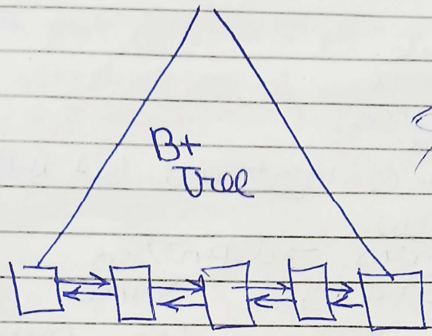
Deletion →

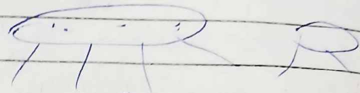
Eq Search + BD ← // shifting WB — Eq Search + D // lazy

Clustered B+ Tree

On this sorted file
make B+ index

no of pages





On the top

L → occupancy fa

//Documented Index

#L → What fraction of page is occupied

80% page fill

If D pages

Whenever crowding we split in agressive this factor will be diff

main

secondary

In case of B Tree we can have success earlier

$0 < L \le 1$

$\frac{B}{L}$ pages → % filled

//scan of Data

$\log_n\left(\frac{B}{L}\right)$

Not n=2 as it is B+ Tree we have max fan

non key attribute Ix can repeat #

We can index to index file multilevel indexing

Address node of root & address

$D\log_n\left(\frac{B}{L}\right) + \frac{B \times P}{L}$

Dbg F(B/L)
going leaf node

Range Search
// ~~last~~ Eg scan // next available page

Eq search + (qualifying pages) D

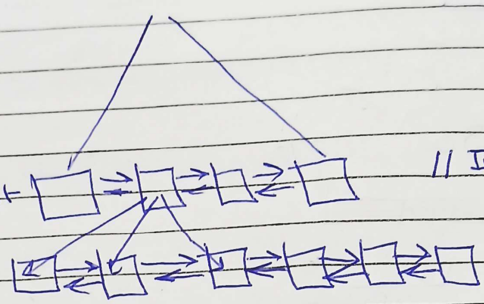
Insert
Start root
come to right page
insert

Eq search + D
Modify page + write

Delete
Eq search + D
Modify + write back

Unclustered B+ tree + Unclustered HW Hash
// Records in random order + on top B+ tree

B+ Tree
Unclustered



$\frac{B}{L}$

We are storing attribute value & ptr

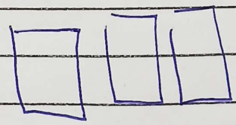
$0 < I \leq 1$ Index Entry / Record Size

$\frac{IB}{L}$ → Because

B+ Tree
↓
every page
Data level not complete

$O\left(\log_F \frac{IB}{L}\right)$ +
// Reaches 1st term
no of Records × D + $\frac{IBD}{L}$

Storing entry & ptr



(Bad idea of scan because no of Records)

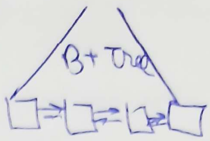
// Directly scan last line → given random order

File
↓
Millions of Records
page → 1000 records

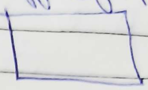
Student
Database
Pn no
Hostel
roll no → Primary key
No of class attended

We are doing

B + Tree on roll no



Buffer of 3 pages



If we want all records on roll no

- Eq search

Index entry
notebook we maintain
// Read that entry + ptr L

$$\left(\log_F \left(\frac{BI}{L}\right)\right) D + D$$

- Range Search

// Eq search

$$\left(\log_F \left(\frac{BI}{L}\right)\right) D + \left(\text{no of qualifying Records}\right) \times D + \frac{x\,BID}{L}$$

Bottleneck

every time u read a record go to heap file and add it to buffer

- Insert

Eq Search
→ at last level insertion heap file
↓
create index entry
↓
insert in B+ Tree clustered file

Clustered → leaf
unclustered → ptrs

• Delete

Cost of Eq search
+ D × c const no of time modify after pe

// Deletion of index entry

B I
L

I# Rat

IB
L

Take all data
B page is read
Each page fraction is occupied
no of page

Heap →

B → no of page in heap

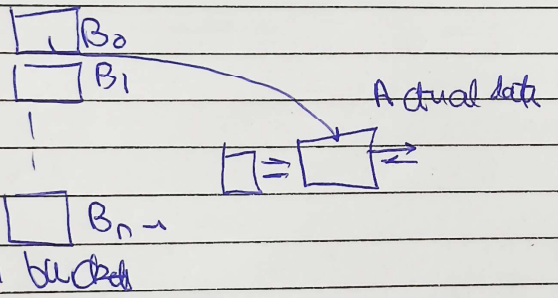
Now in B+ tree unclustered

#records = B × no of records per page

Adv → # Heap & unclustered
Scan better Eq Search with Range S

We will ask what is most freq Opr

- Hash Index
  Unclustered Index
  because no benefit of sort in Hash



n buckets

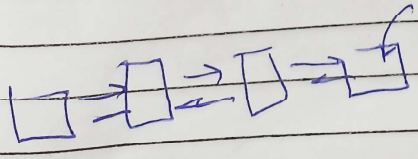
// We have to go to each page of bucket

Scan Index + no of X.D # records

Bucket Opr

Hash → Eq S

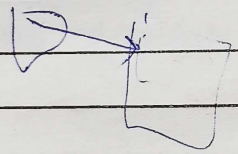
// compute Hash of k
We know which bucket to ad page of bucket & it should fit in main memory
& Perform search in bucket



We have to fe

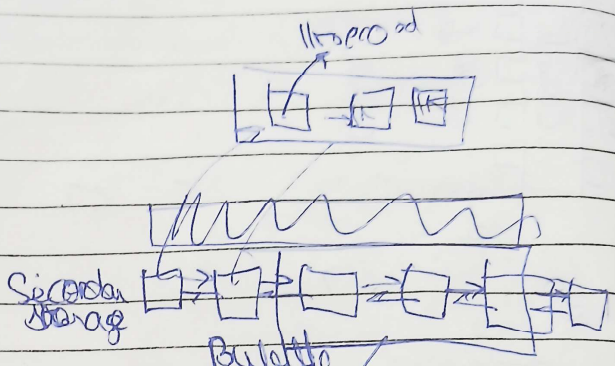
- Range search
  Heap
  range search (Heap property still holds)

- Insert
  ~~HeapFun~~. Heap insert const no of IO's
- ~~Delete~~
  Hash / Find Record / go → Eq search
  ↓
  2.0 // const



External Sort

1. Bubble sort
2. Selection sort
3. Insertion sort
4. Quick sort
5. Merge sort



Put in main memory

Compare with next three

One complete pass through file and we get largest file

2nd Largest →

Smallest

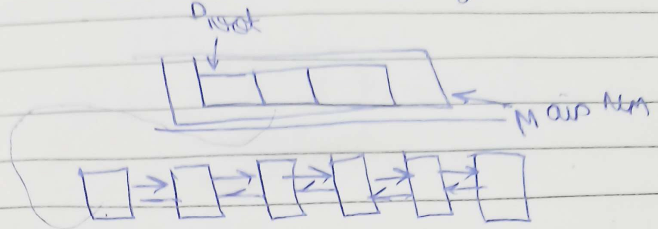
1. Quick
Middle page

pivot

Bubble → compare adjacent

Selection → select smallest

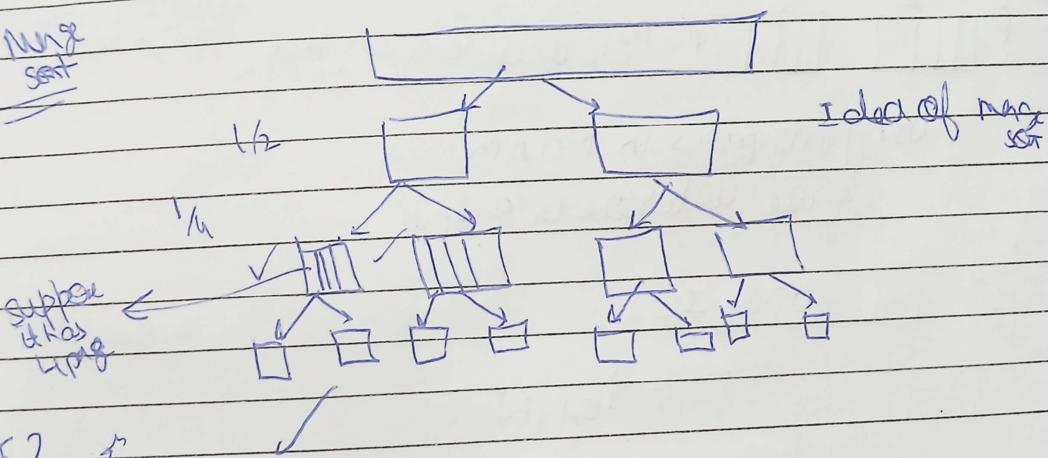
// we put in 1st page



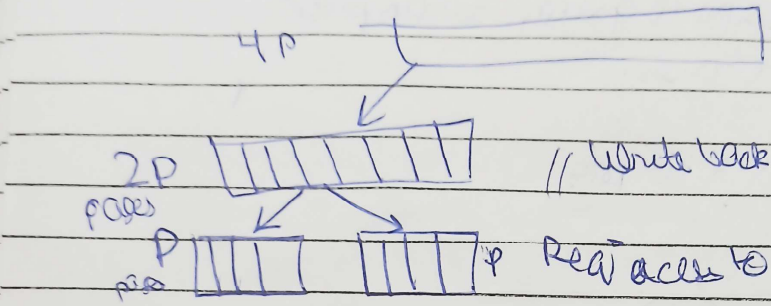
First page in MM
some part is greater than pivot
some is less

Fetch first page & last page

Merge sort



F2
Forged [illegible]
Born 2 [illegible]
Fo[illegible]
Sorted in ascen



bring two pages in main memory
& use will store in output

2P+2P
P+P
1P IO's
Out

no of IO's

$\log_2 N$

Iteration No	Run size		
1	1	2N	// sort page in main mem
2	2	2N	No of IO's
3	4	2N	n IO's to reach Each page & write each page
	N		

Step pages will change

2N as reads N 2N

$\log N + 1$

No of

$O(\log_2 N)$

Each one → 2N

We sorted blocks

LD

$2N \times \log_2 N$

3 pages enough to sort all

Instead of 3 we have 20 pages

Req^d 1 page for 1 run

20 pages

Lazy → use 3, Leave 17

We can increase → Start with size 20 & double

$2N \log_2(N/20)$

Merge sort
3 part

Goal → Do better

How many we can combine?

↓

19

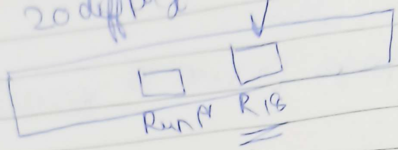
Each page combine

$2N \log_{k-1}(N/k)$

Parameters -> k

20 diff pag

//All records are sorted

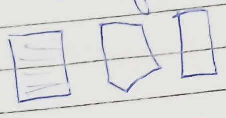


Extra page

. Get pos 2

Insted of one give two page

Min no of all participatio



whenever page finished
we have to bring

Fetching stop

(R-1)/2 runs at a time

Data File N pages
K page Main Mem

K1 page Input
K2 page Output buffer

At time of fetch ...

Total Run ...

Multiple ... and ... I/O

Deterministic ...

K-runs

1	20
2	2
3	4
4	8

Implementation

Operators
Relational Algebra

σ

×

⋈

∪

∩

−

σ(R)
□ → Condn

$R_{attr} = □$
\>
<
< > Range Search

Unclustered
locate ptr

• $\pi_{□}(R)$
projection

Reln at end, we have to eliminate Duplicates

R' (M' pages)
Sort → external Sort
(M' log M')
→ One pass for dupl. eliminn
(Can we avoid extra pass)

At time of fetch ... page empty

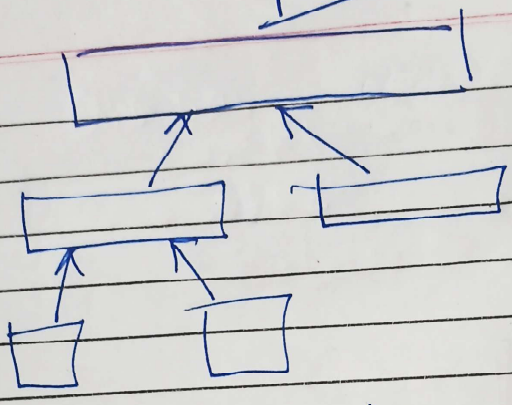
Total Run Time

Multiple ... of I/O
... reduce

...-runs

1 20
2 2
3 4
4 8

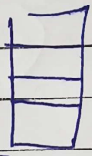
// Remove Duplicate is Merge while merging



So Just

External sort $M \log M$

Hashing



Bucket if (bucket >

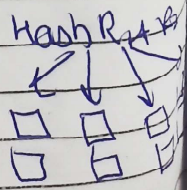
• Cross Product
// everything is output
same can't.

• ∪ → Union
R1 R2 Compatible
R1 records put with R2 records
Remove duplicates

Sorting → remove duplicate

Sort R_1 & R_2 → Sort of Merging

• Hash R_1 & R_2



Total buffer B

- ∩

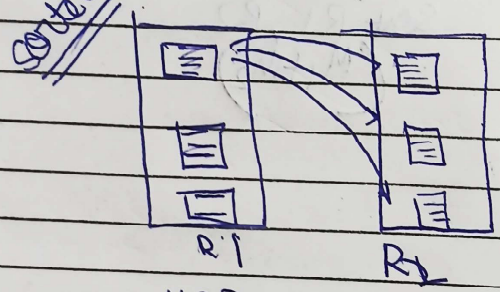
  Just keep duplicate
  Remove all others
  this

- Set diff - same to do

- Join →

~~Sort of~~ R1 ⋈ R2

M1 M2



Nested Join

Two ptr method

// R1 going // R2 going

M1 + (M1 * M2)

// P1 Records in page of R1

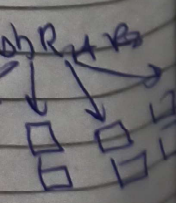
M1 + (M1 M2)

scan only once

do for a page

Smaller assets

Multiple pages at a time



Total left B

For each page in outer Rel
For each page in Inner Rel

block of pages

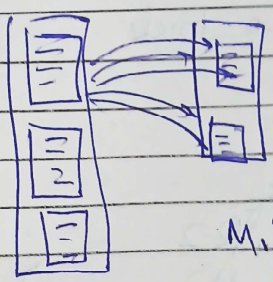
M1 + ⌈M1/(B-2)⌉ M2

We have to pay cost

Check * $M_1 + \left\lceil \frac{M}{B-1} \right\rceil M_2$

Other way
- Sort R1
- Sort R2
- Join based on equality value

Equality join → Sort



BOOK

$M_1 \log(M_1)$ ↓ Sort R1, R2 ↓ $M_2 \log(M_2)$

$\therefore (M_1 + M_2)$

Merging of two sorted lists