Agenda

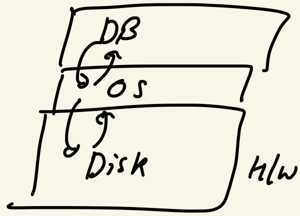
will add notes ⇐ ② BCNF (leftover) — 15-20 min

① In-depth Indexing, B-Tree, B+ Tree — 1 hr 30 - 2 hr on Indexing.

Files

DB Tables → In Disk as files.

Data is NOT in the RAM.



Objective: access the data on disk fast
↳ search, insert, modify, delete

C: fopen
fclose

DSA: LL, Array
BST / Balanced BST → assume data is in the RAM.
HashMap Binary Search main memory

Need to come up with DS that work on disks.
(secondary memory)

Q) Why do we need an index? → DS such that access based on attr. is faster.

PK.

customers

id	name	addr.
1	n_1	a_1
4	n_2	a_2
6	n_3	a_3

Search a customer with given id.

Search for customers with name prefix 'Sa'

Search based on some attribute very often

We often create indexes on a given table.
($>=1$)

Simple Linear file

CSV

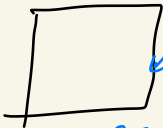
1, n1, a1,
2, n2, a2

id=16

Search
Insert
Delete
modify.

Disk

100GB.



RAM 8GB.

id	name	addr
1	—	—
2	—	—
5	—	—
6	—	—
9	—	—
10	—	—
12	—	—
13	—	—
16	—	—
17	—	—
18	—	—

chunk of data = block.

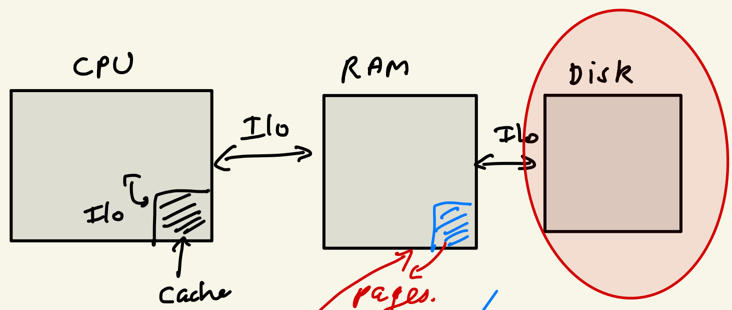
RAM much much smaller than db on disk.

Once data is in RAM, we can apply any DSA.

How does disk access work?

Accessing data on a disk

OS
Computer organization



ALU
Arithmetic
Logical
Unit

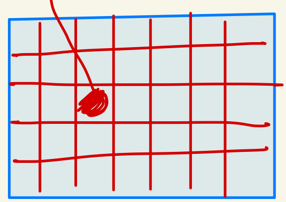
Logically broken into blocks

512 Bytes to 4096 Bytes
fixed size

Depends on disk & manufacturer
HD
SSD
SD card.

DISK-READ
intensive copy

Block gets loaded into the RAM.



>90% of your time in a SQL query is spent on copying data from disk to RAM.

1000 Rs Disk → RAM
10 Rs Search in RAM.

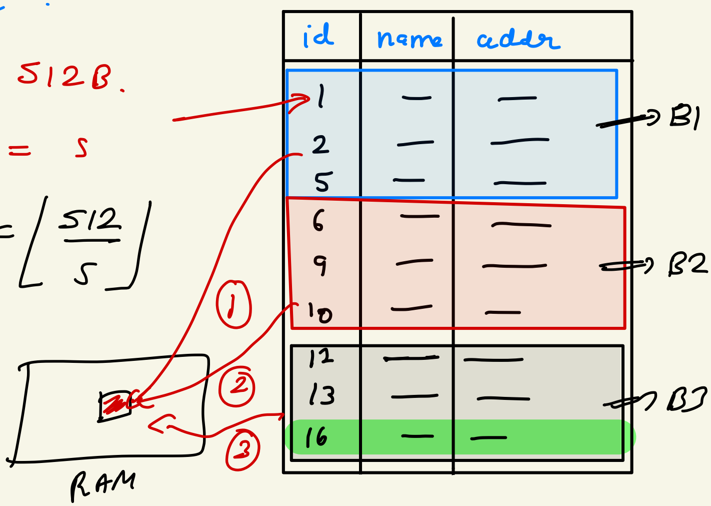
Cost of accessing data.

Size of 1 block = 512B.

size of 1 row = 5

Max record/block = $\left\lfloor \frac{512}{5} \right\rfloor$

Search for id = 16



Worst case

disk reads = 3.

1
2
5 } Is 16?

File size = F bytes.

1 block = 512 bytes

32.5 → 33

$X \longrightarrow x * 512 = F$

$\Rightarrow x = \left\lceil \frac{F}{512} \right\rceil$ ceil.

4GB $\frac{4096}{512} = 8$

1GB $\frac{1024}{512} = 2$

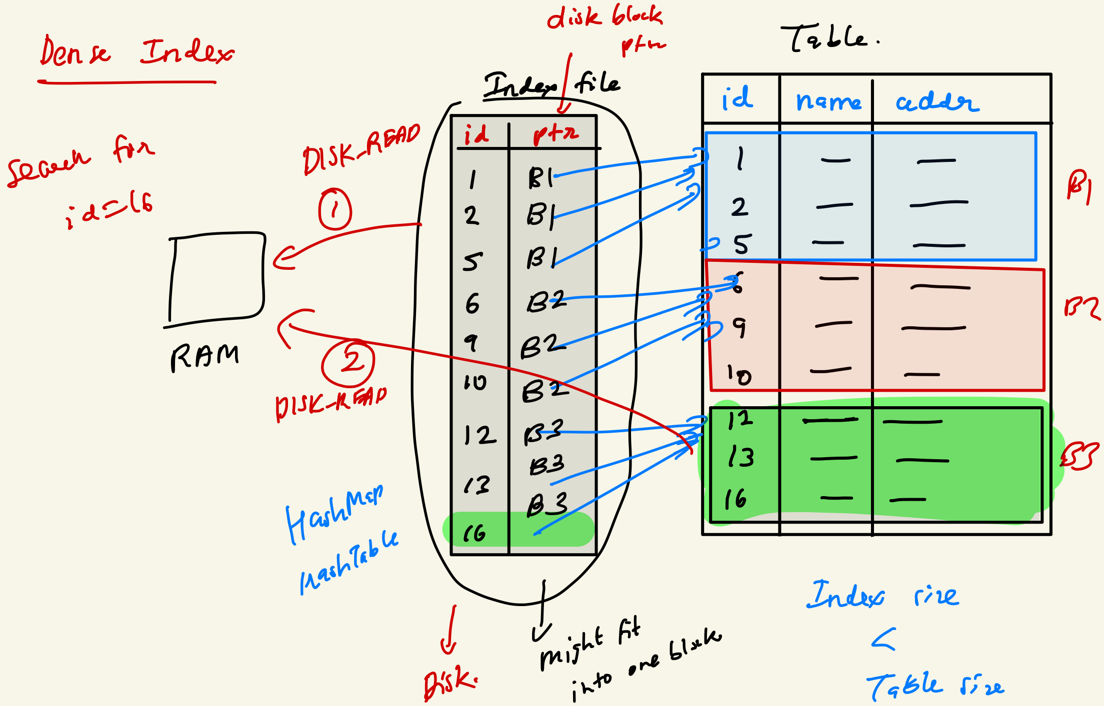
10GB → $\underline{20}$

10GB

Min. no. of disk reads.

addr of next node in RAM.

Dense Index



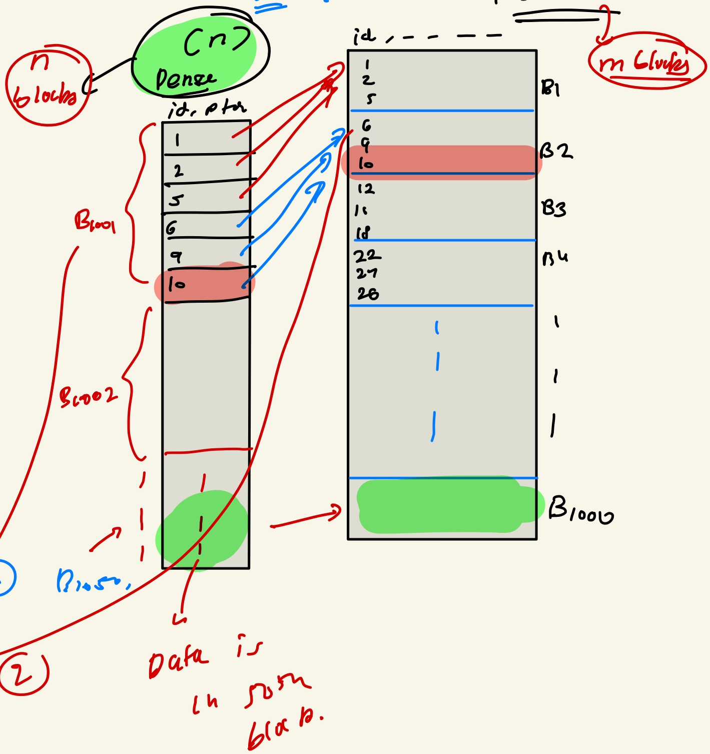
Problem: If whole index doesn't fit into a single block, this doesn't work.

Dense

~~Sparse~~ Indexing

Is $n < m$??

$n \downarrow$ 50 $\quad m \downarrow$ 1000



Worst case no. of disk reads.

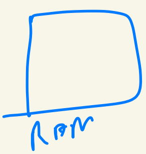
1) 1000

2) 50

3) 51

DISK-READS = Blocks in dence index +1

Search for 10



HM = { (Khagen, 100)
(Rahul, 100) }

HM ["Khagen"]

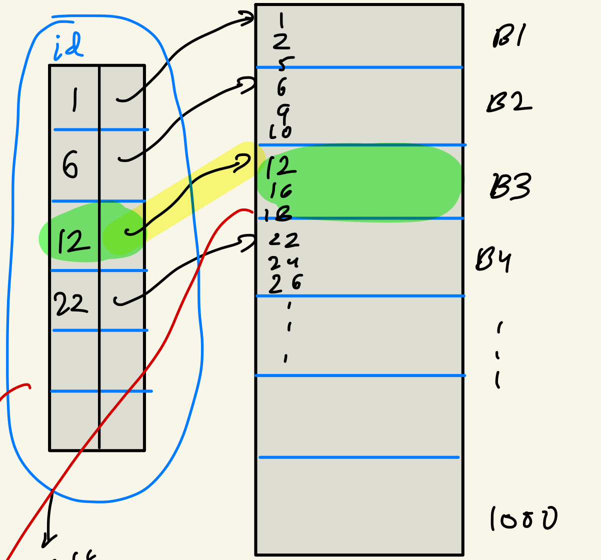
Sparse Index

Start record's id of every block. => ptr to 1st row of each block.

Goal: Min. # DISK-READS.

Sparse Index

Large table (n Blocks)



might fit into 1 block.

Search 18

Index 1st val <= 18

RAM

(1)

(2)

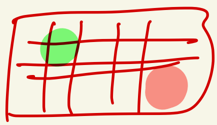
2 disk reads.

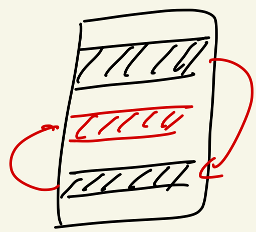
What if doesn't fit into 1 block.

1, 6, 12, 22

18

LL





(Q)

m + 1

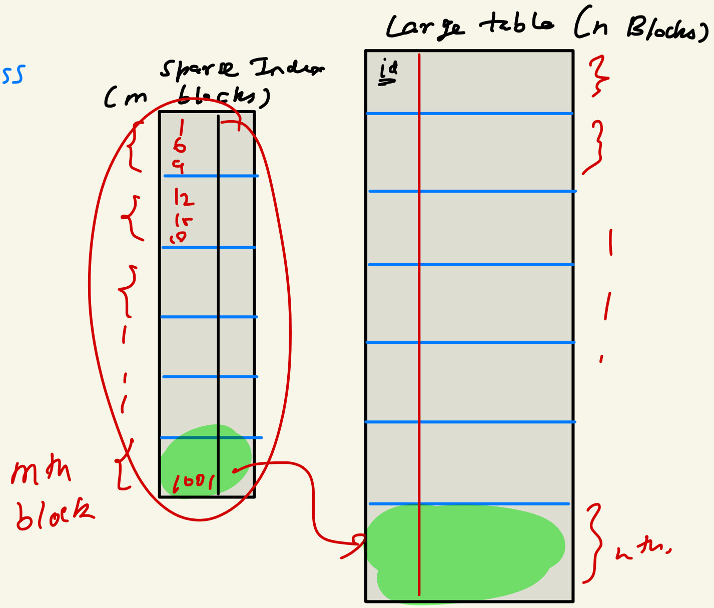
DISK_ACCESS

of

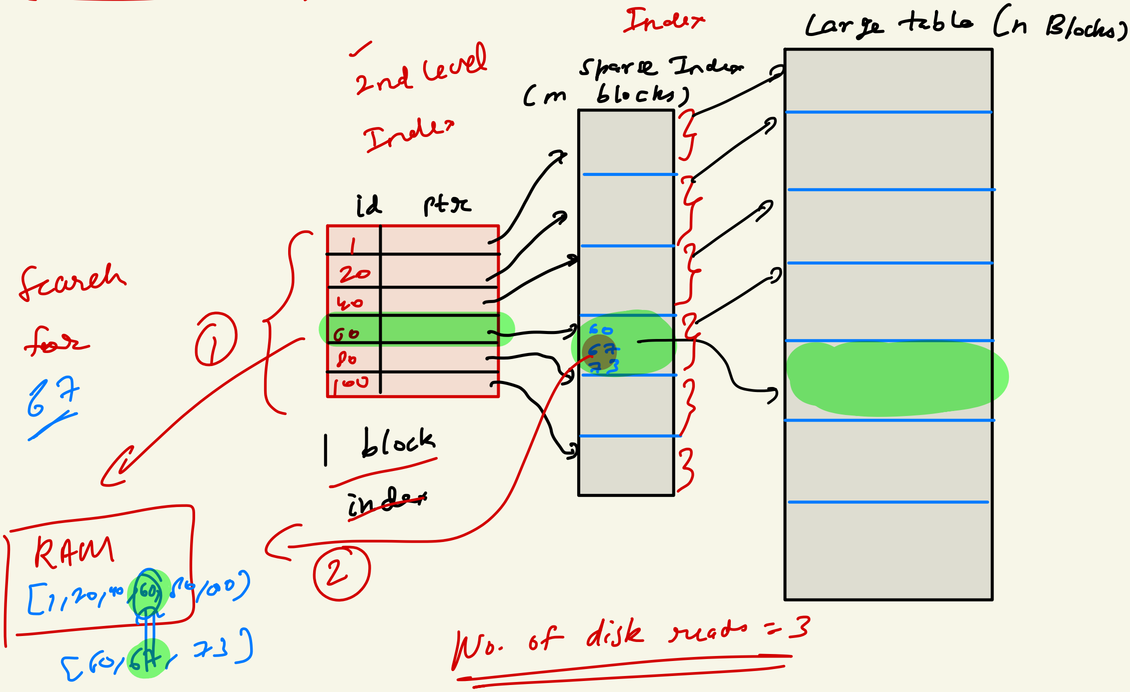
m+1

blocks

1001



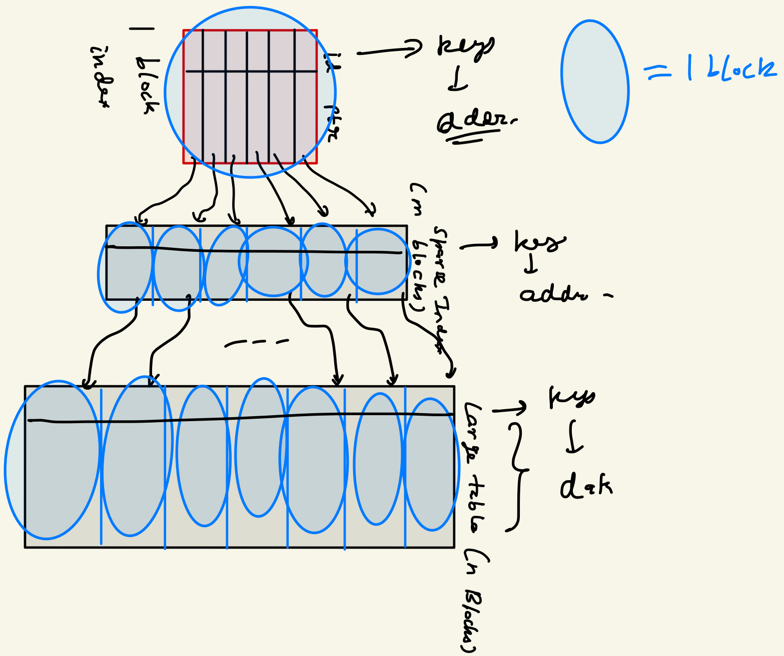
Multi-Level Index



No. of disk reads = 3

N levels of index

No. of disk access = $\underline{N+1}$



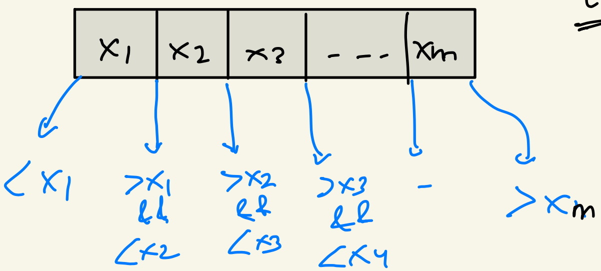
Every block as a node.

Multi-level index can be visualised as m-ary tree

Indexes as m-ary search trees

extension of BST.

m+1 max child nodes



x1
x2
:
xm

} keys.

Search

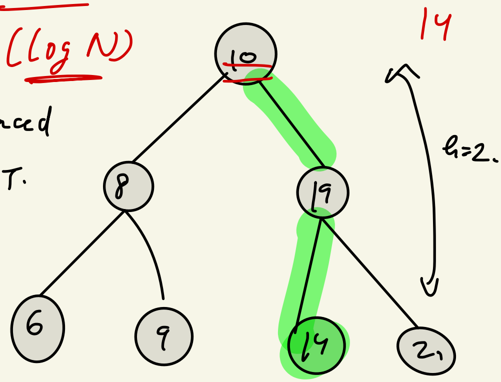
$O(\log N)$

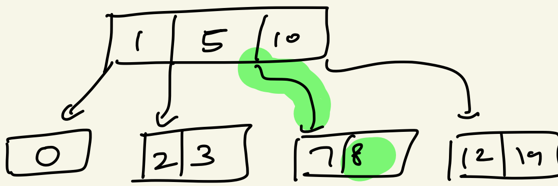
Balanced BST.

$O(h)$

BST (2-ary trees)

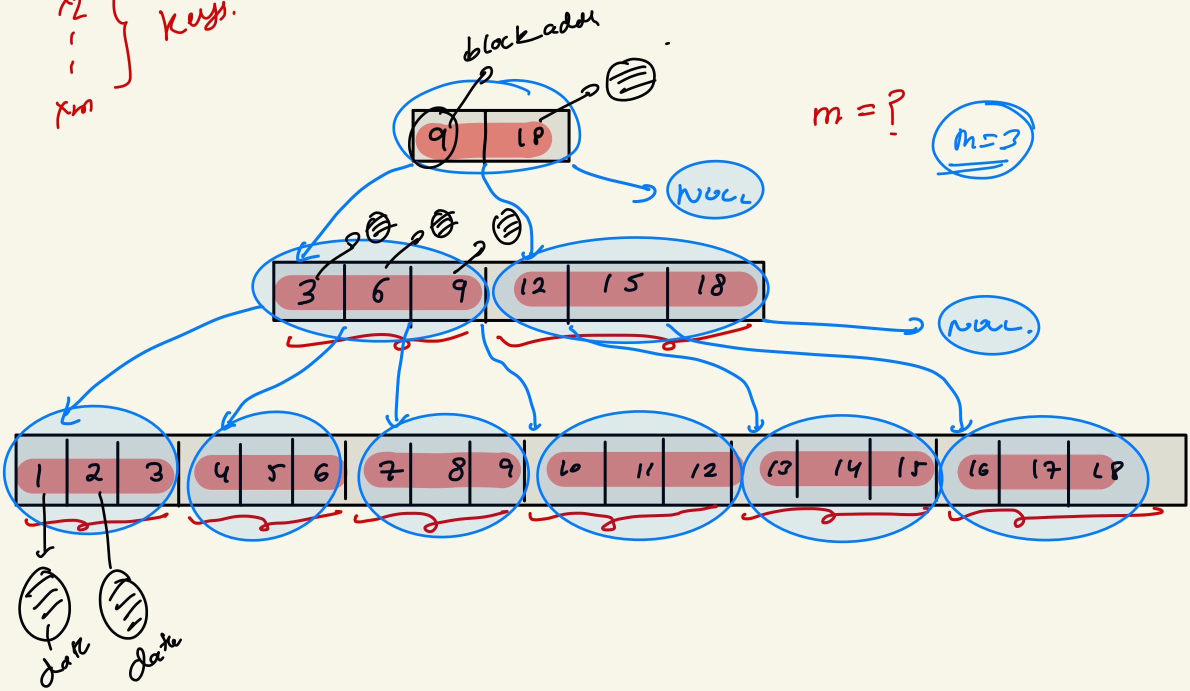
14

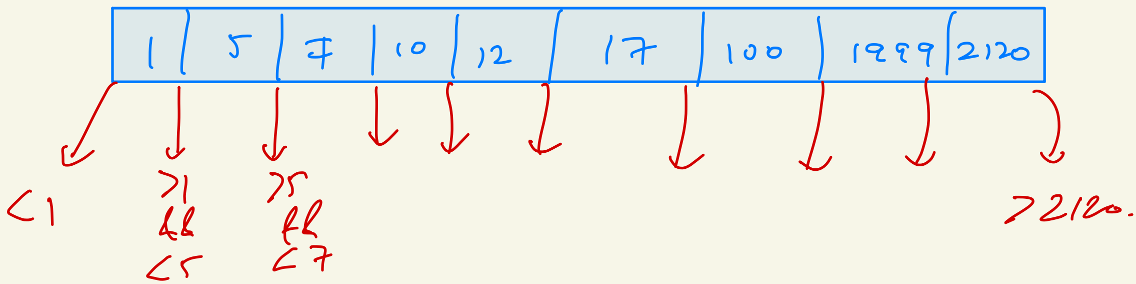




m = ?

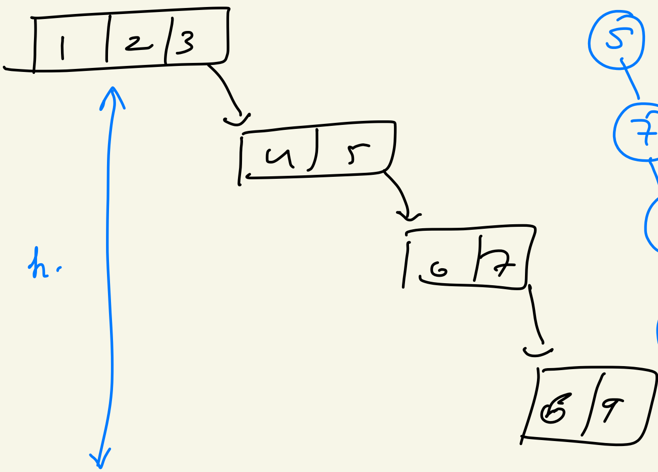
m=3





Goal: Min. no. of disk_reads

Construct it to min height



5, 7, 10, 12

✓ B-Tree

✓ B+-Tree

BST

vs

AVL

vs

Redblack

B-Tree inspired from m-ary trees.

height balanced m-ary tree.

---

properties

a size which can fit in a block size.

degree ⇒ t

order ⇒ 2t

(1) Every node is a block.

(2) Root node is stored in the RAM.

---

(3) All the leaves are at the same level.

(4) All the keys of a node are stored in increasing order.

---

(5) Node must have atleast (t−1) keys. (except root) contain min of 1.

(6) All the nodes can contain max (2t−1) keys

1−30

$t=2$

18, 15, 25, 21, 24, 27, 30, 10, 17, 7, 16, 5, 2, 9

Every node will have keys

1 →

$t-1=1$

$2t-1 = 4-1 = 3$

min = 1

max = 3

(i)

18

$\boxed{18}$ → (18, blockaddr)

(ii)

15

~~$\boxed{18}$ $\boxed{15}$~~

$\boxed{15 \mid 18}$

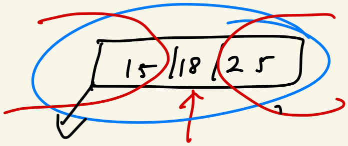
root ✓

leaf ✓

$t = 2$

iii)

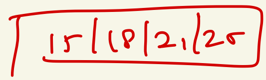
25



$2t-1$ keys

Full node

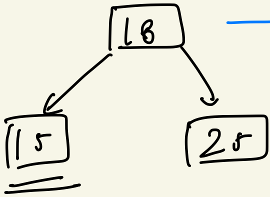
Bottom to Top



4 keys

iv) Split the root node

21



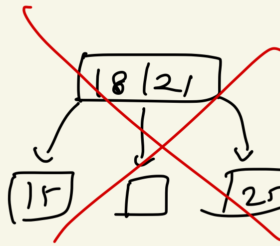
1

2 keys $= 2t-2$
$= \text{even}$

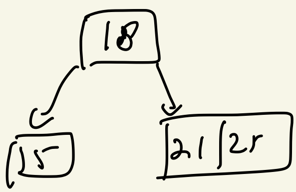
$2t-2$
$= 2(t-1)$

$\frac{2t-2}{2} = t-1$

$\frac{2t-2}{2} = t-1$

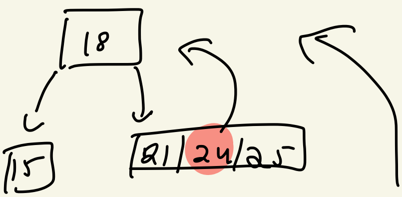


Always insertion happen at the leaf.



v)

24

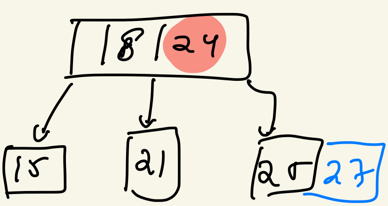


middle node goes up.

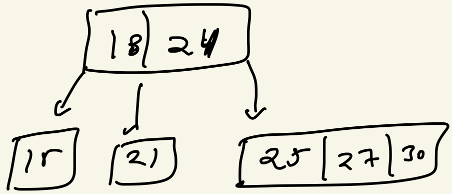
vi)

27

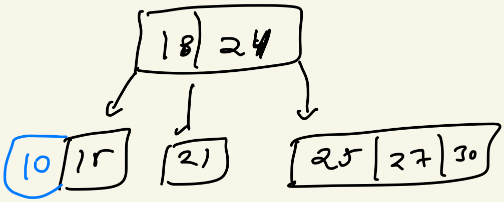
1st split then insert



vii) 30



viii) 10

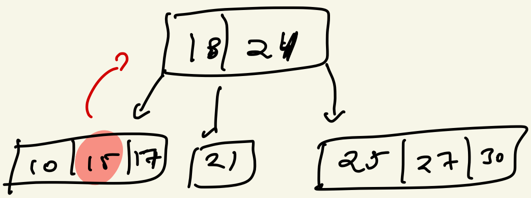


Tc to insert in sorted list of size N

$O(\log N) + \underline{O(N)}$

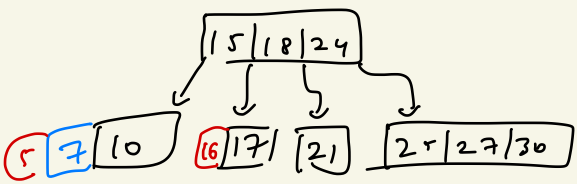
shift

ix) 17



x) Node is full, 1st split

7



$t = 2$

$\Downarrow$

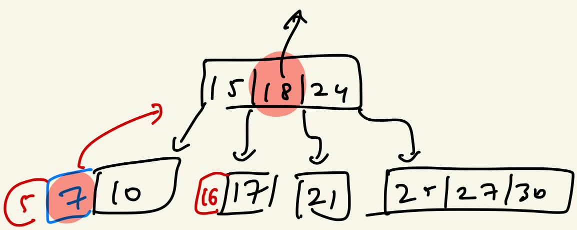
max child $= 2t = \underline{\underline{4}}$

16

5

2

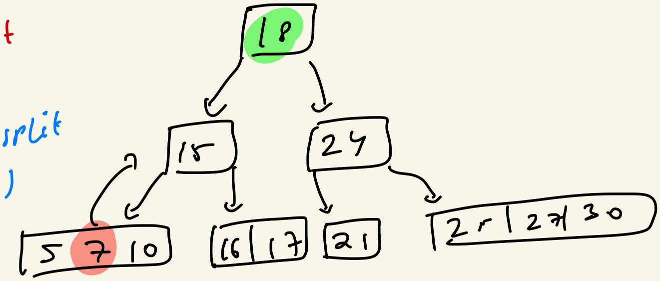
9



Further split

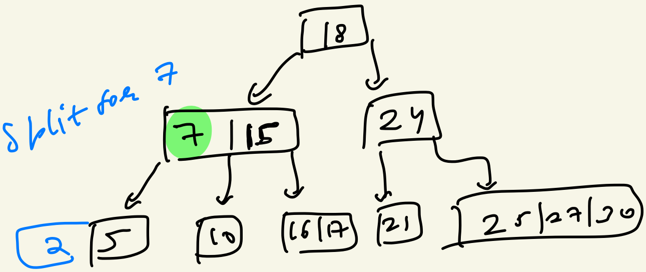
First split

Root split (18)



Rearrangement

Split for 7



Then we insert 2

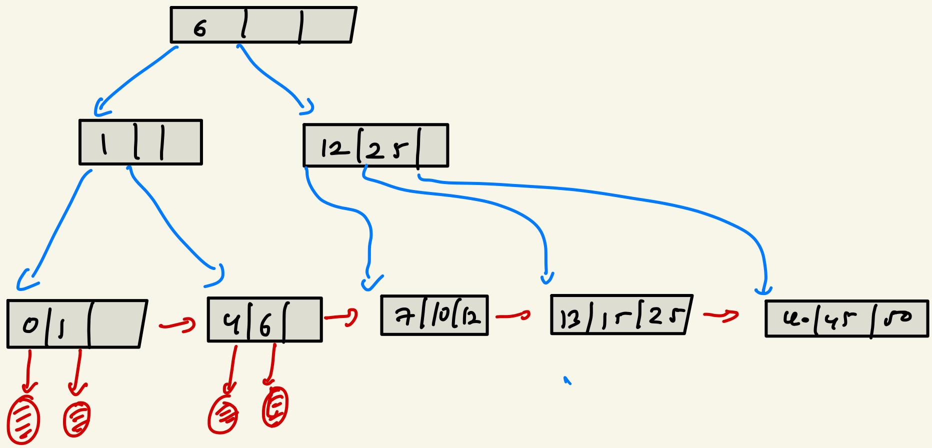
Order of B-Tree = max no. of children (terms of t)

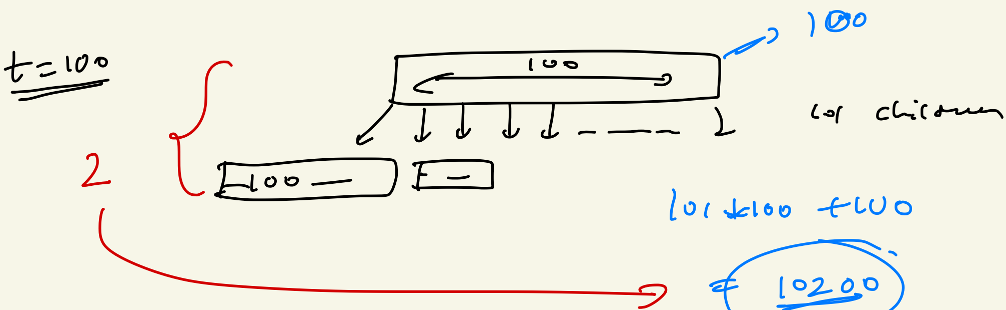
$$= (2t-1)+1 = 2t$$

B+ Tree it's an extension of B-Tree

=> only leaf nodes have the data

=> leaf nodes are connected to each tree





max disk access = Height order of tree

Depend on t.

Doubts

first_name, last_name

Composite Index

(first_name, last_name)

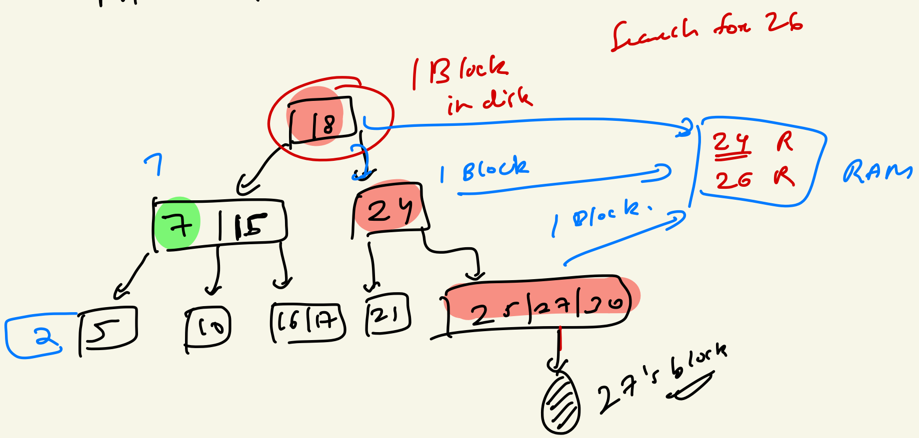
Distributed DB

Indexing ?

=> first_name =>

Let's think about it. Research.

27



list of keys

list of child block addr.

Search for 39