AVL trees

AVL Trees

1 .These are height balanced binary search trees, We balance height of a BST, because we don't want trees with nodes which have large height

2. This can be attained if both subtrees of each node have roughly the same height.

3. AVL tree is a binary search tree where the height of the two subtrees of a node differs by at most one

Height of a null tree is -1

AVL Tree

Que. How to find out balance of a BST.
Ans: By finding the balance factor of a BST.

Balance factor = height of left subtree – height of right subtree

All node's balance factor should be {-1, 0, 1}

If any node's balance factor is less than -1 or more than 1 then it is not balanced search tree.

If an insertion cause an imbalance, which nodes can be affected?

Nodes on the path of the inserted node.

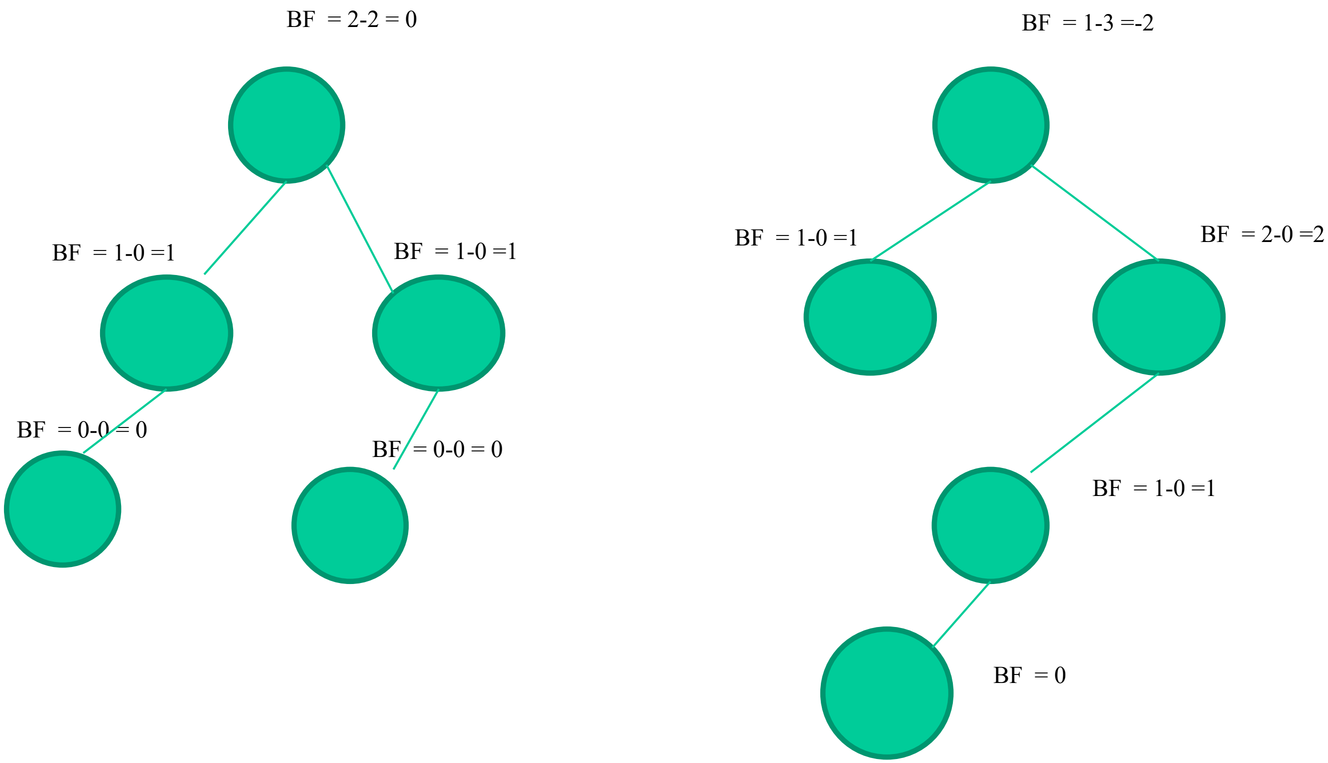
Let U be the node nearest to the inserted one which has an imbalance.

insertion in the left subtree of the left child of U

insertion in the right subtree of the left child of U

insertion in the left subtree of the right child of U

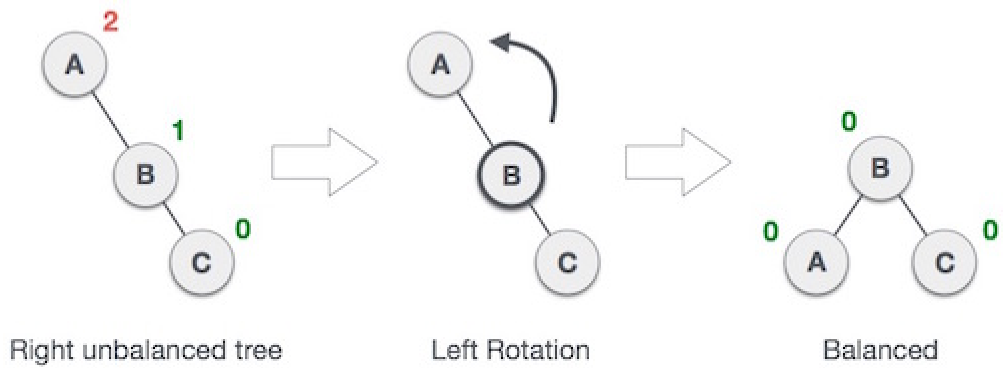
insertion in the right subtree of the right child of U



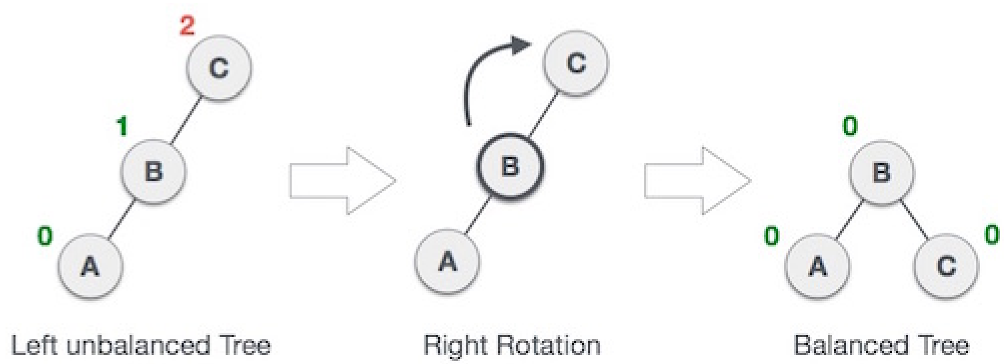
So, at the time of insertion we need to check balance factor of node and if BF is more than 1, then we need to perform rotation.

Rotation is performed always on 3 nodes

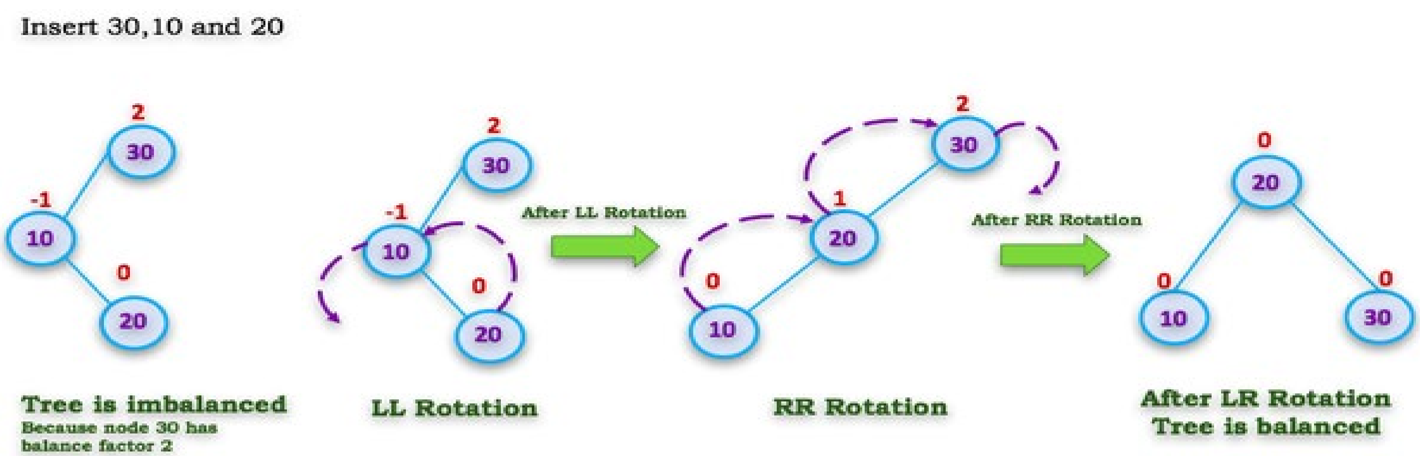
LL rotation ------------- RR imbalance



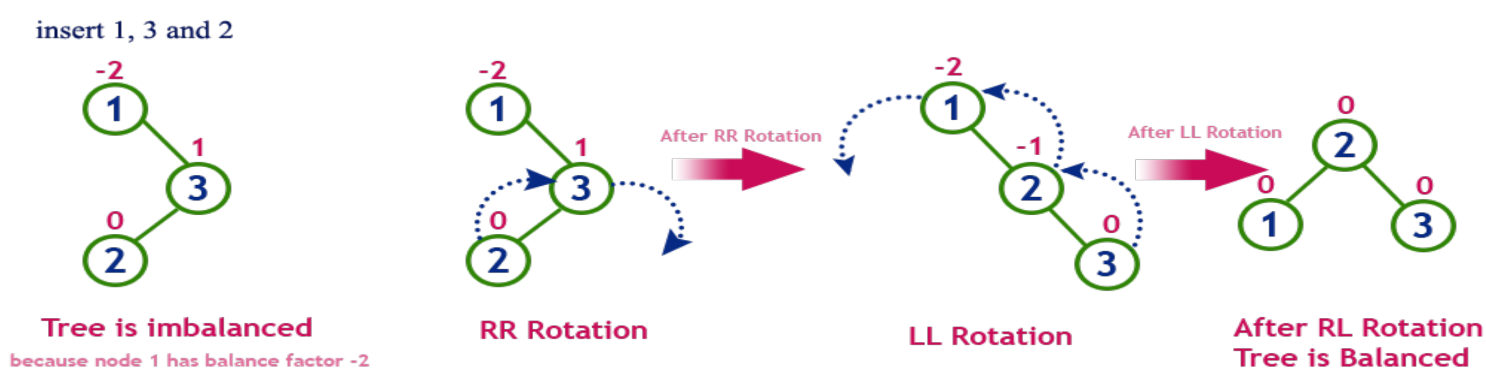
RR rotation ------ LL imbalance

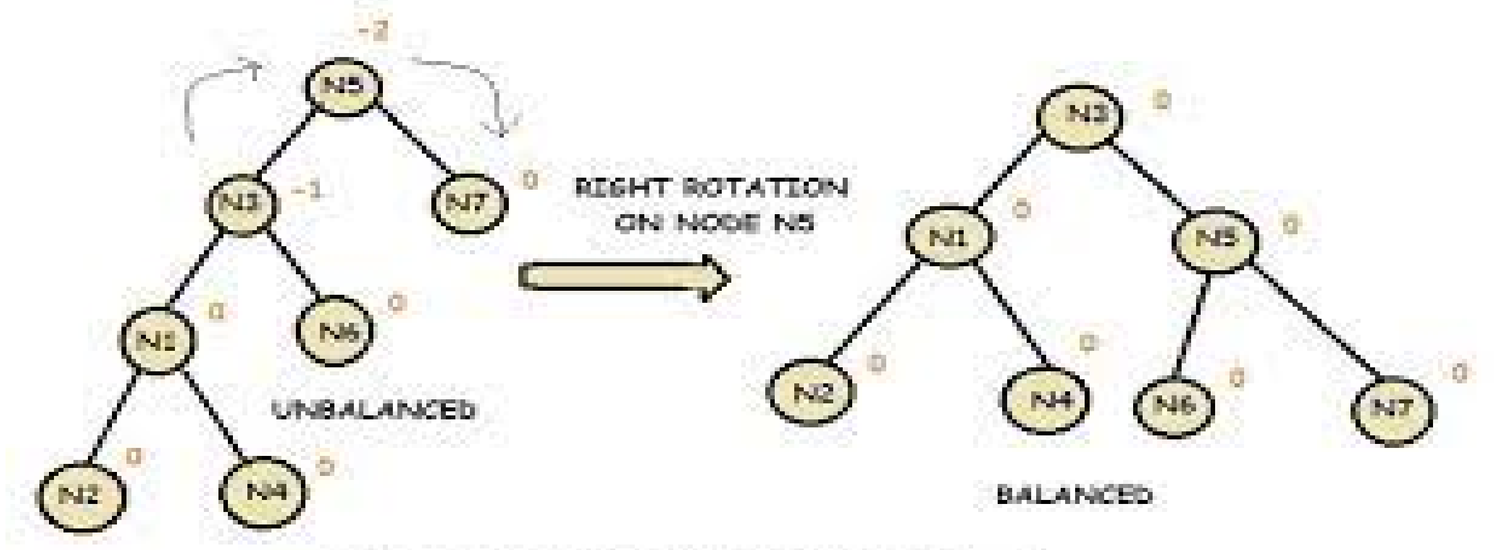


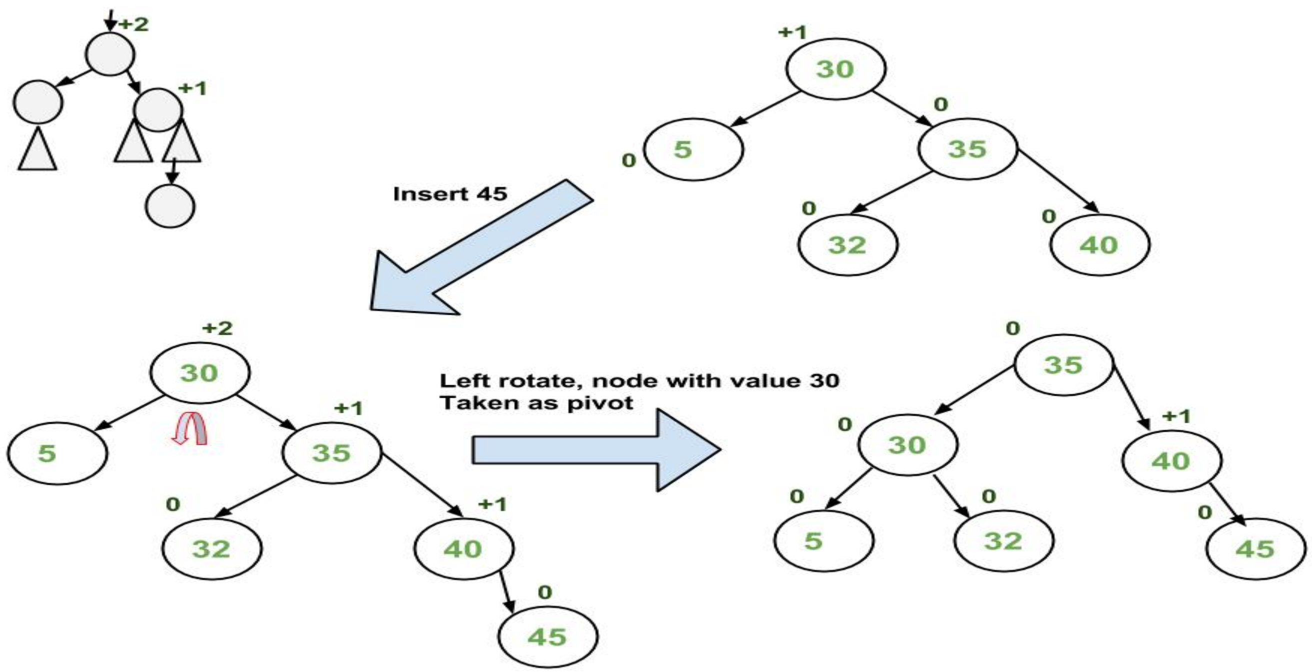
LR rotation



RL rotation

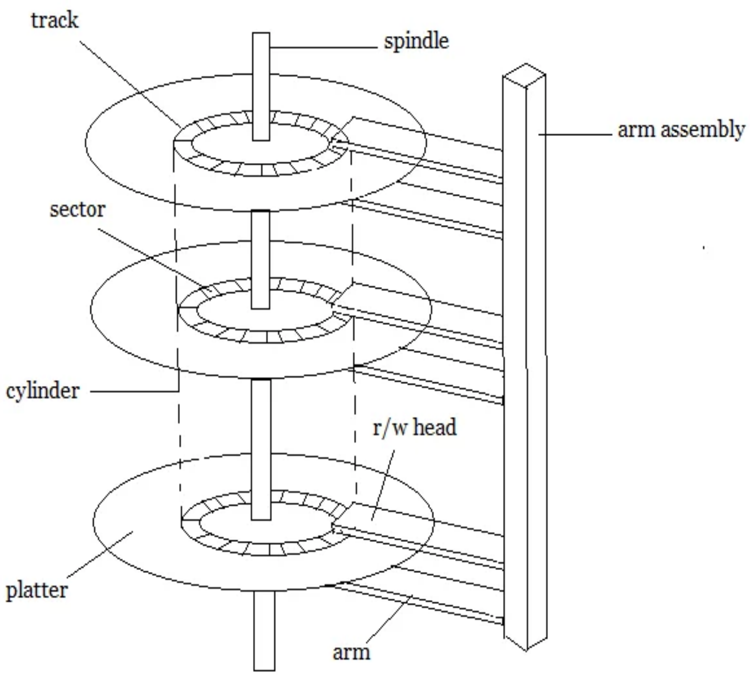


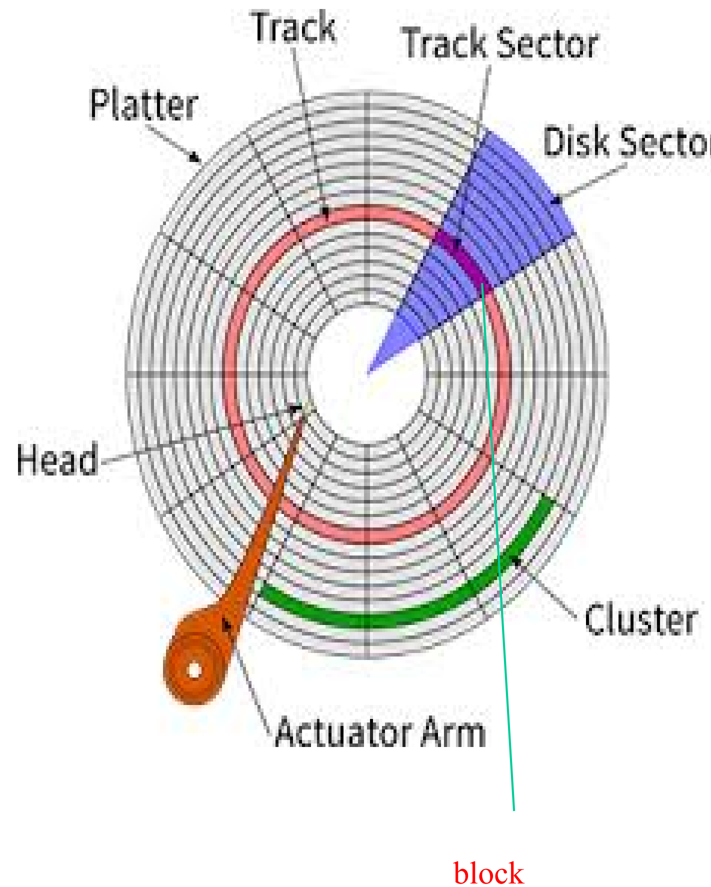

N5
N3
N7
N1
N6
N2
N4
UNBALANCED
RIGHT ROTATION
ON NODE N5
BALANCED


Insert 45
Left rotate, node with value 30
Taken as pivot
30
5
35
32
40
45

Multiway Search Tree:

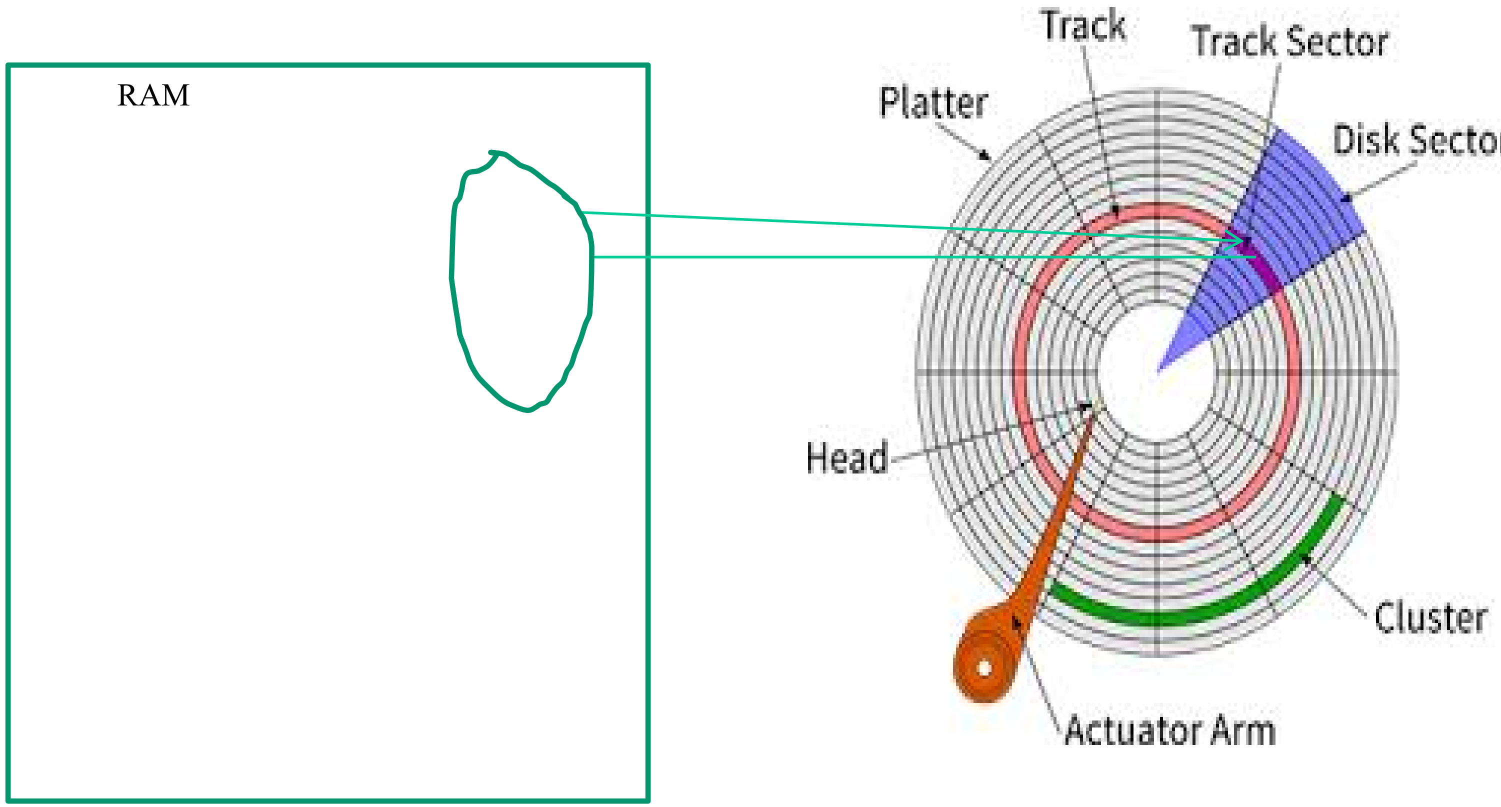
1. Disk structure
2. How data is stored
3. What is indexing
4. Multilevel indexing
5. M-Way tree
6. B tree
7. B+ tree





Block: block address means track number + sector number
Typical block size is 512 bytes.
We always Read and write in terms of block

Each byte can be accessed by offset.



Data has to be brought into RAM, now how is stored in HDD is DBMS and how it is placed in RAM is DS.

Size of row is 128 bytes
First name – 25
Last name – 25
Address – 50
City – 18
Id - 10

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

No. of records per block = 512/128 = 4
For 100 records = 100/4 = 25 blocks for 100 records
In case u perform search for an employee, we need no. of blocks, we need to access 25 blocks. Can we do it faster???

Yes, create index and keep it on disk say on a block.. How many blocks will be needed??

Index: 10 for id and 6 for pointer = 16 bytes

Id	pointer
1	Adress
2	Address
3	address

For index
No of entries per block
512/16 = 32
Total 100 records
100/32 = 3.2 say 4 blocks are needed for index

1000 records : 250 blocks

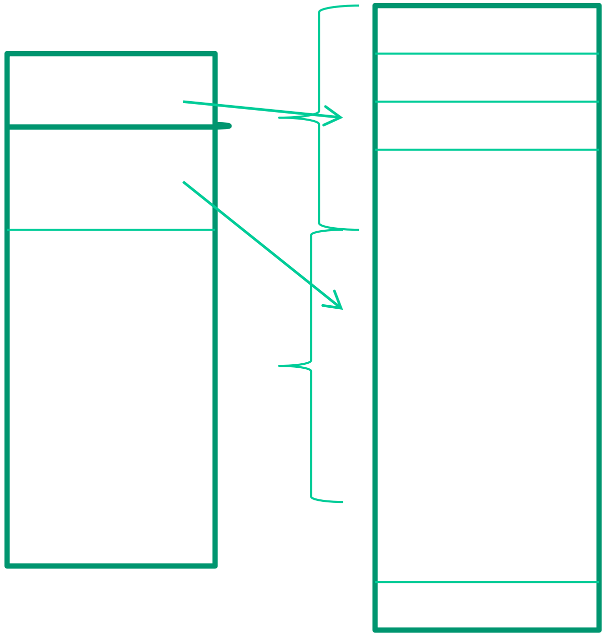
After indexing : 40 blocks

so for 100 records 4 blocks

for 1000 – 40 blocks

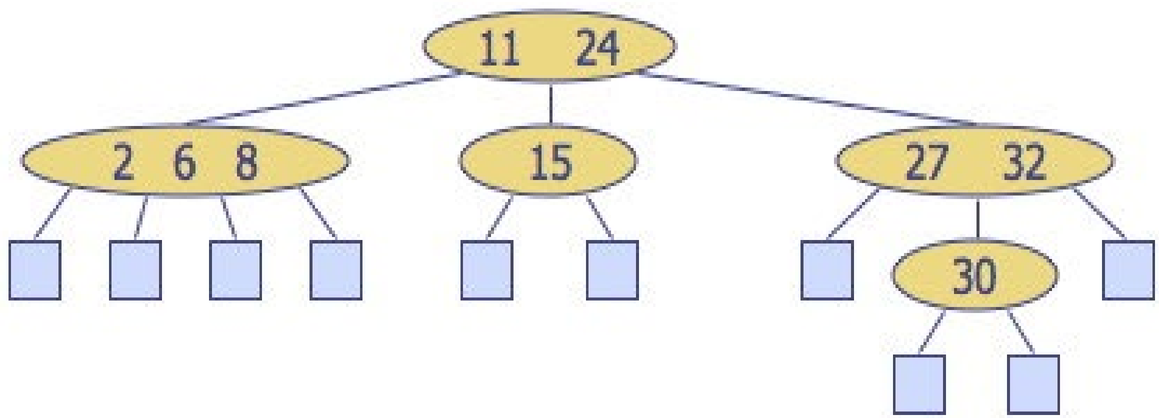
Create index above an index is possible:

So now per block we can have 32 entries



M-way search tree: each node can have m children, and m-1 keys

Keys – 2 keys
Children – maximum 3, so this is 3-way search tree



Height balanced m-way search tree is B tree.

B-Tree:

1. Root can have min 2 children
2. All leaf nodes are at same level
3. All non leaf nodes should have at least m/2 children
4. The creation process is bottom up.