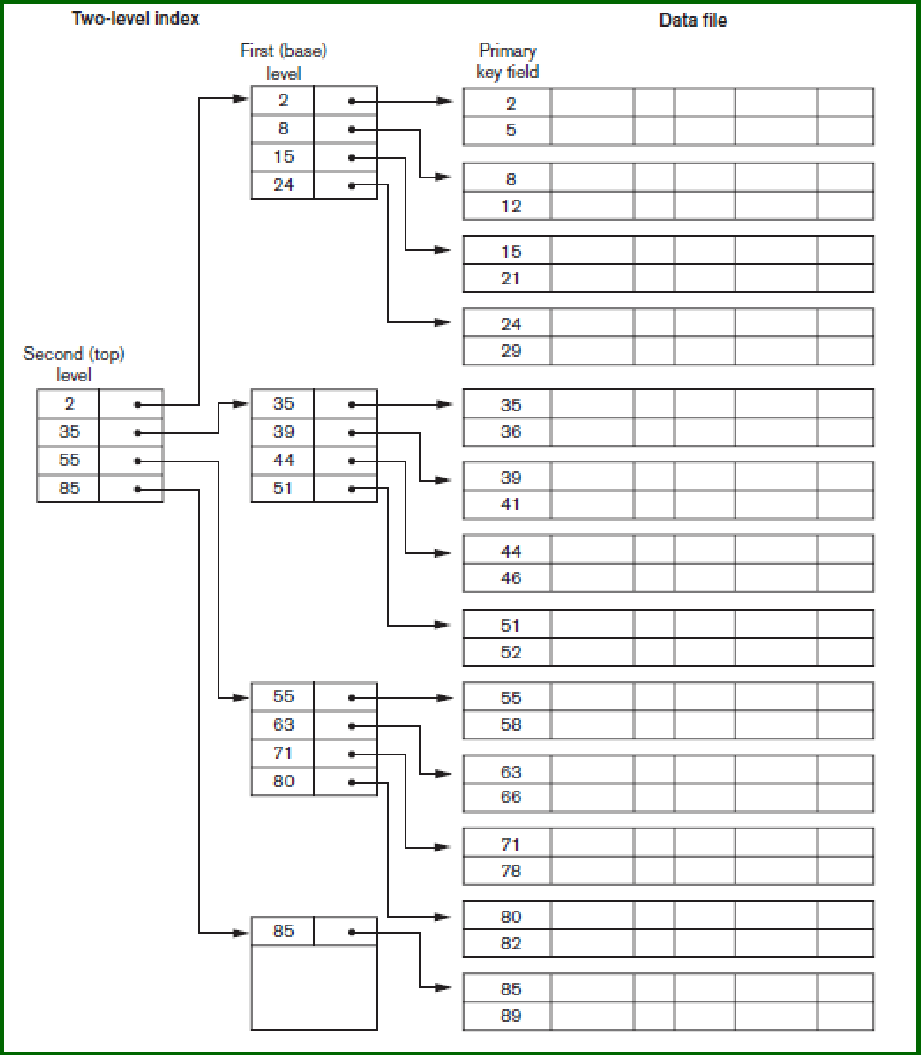
Multi-level Indexes

- Because a single-levelUpload and open index is an ordered file, we can create a primary index to the index itself; in this case the original index file is called the first-level index and the index to the index is called the second-level index.
- We can repeat the process, creating a third, fourth,… top level until all entries at the top level fit in one disk block.
- A multi-level index can be created for any type of first-level index as long as the first-level index consists of more than one disk block.

Multi-level Indexes

- Such a multi-level index is a form of a search tree; however, insertion and deletion of new index entries is a severe problem because every level of the index is an ordered file.

Multilevel Index (Cont.)



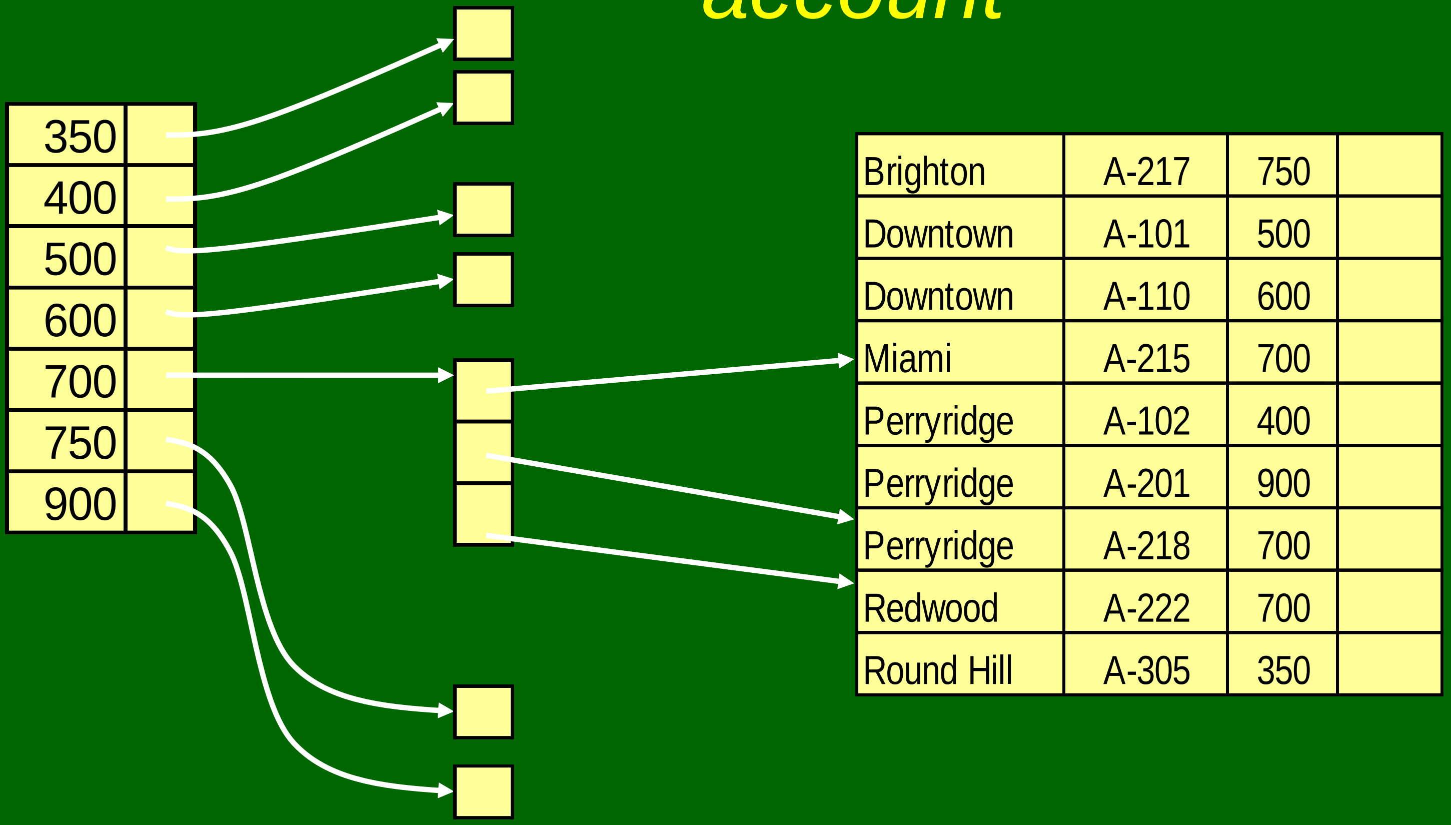
Index Update: Deletion

- If deleted record was the only record in the file with its particular search-key value, the search-key is deleted from the index also.
- Single-level index deletion:
  - Dense indices – deletion of search-key is similar to file record deletion.
  - Sparse indices – if an entry for the search key exists in the index, it is deleted by replacing the entry in the index with the next search-key value in the file (in search-key order). If the next search-key value already has an index entry, the entry is deleted instead of being replaced.

Index Update: Insertion

- Single-level index insertion:
  - Dense indices – if the search-key value does not appear in the index, insert it.
  - Sparse indices – if index stores an entry for each block of the file, no change needs to be made to the index unless a new block is created. In this case, the first search-key value appearing in the new block is inserted into the index.

Secondary Index on *balance* field of *account*



Brighton	A-217	750	
Downtown	A-101	500	
Downtown	A-110	600	
Miami	A-215	700	
Perryridge	A-102	400	
Perryridge	A-201	900	
Perryridge	A-218	700	
Redwood	A-222	700	
Round Hill	A-305	350	

Bitmap Index

- A bitmap index for attribute A of relation R is:
  - A collection of bit-vectors
  - The number of bit-vectors = the number of distinct values of *A* in R.
  - The length of each bit-vector = the cardinality of R.
  - The bit-vector for value v has 1 in position i, if the ith record has v in attribute *A*, and it has 0 there if not.
- Records are allocated permanent numbers
- There is a mapping between record numbers and record addresses.

Example

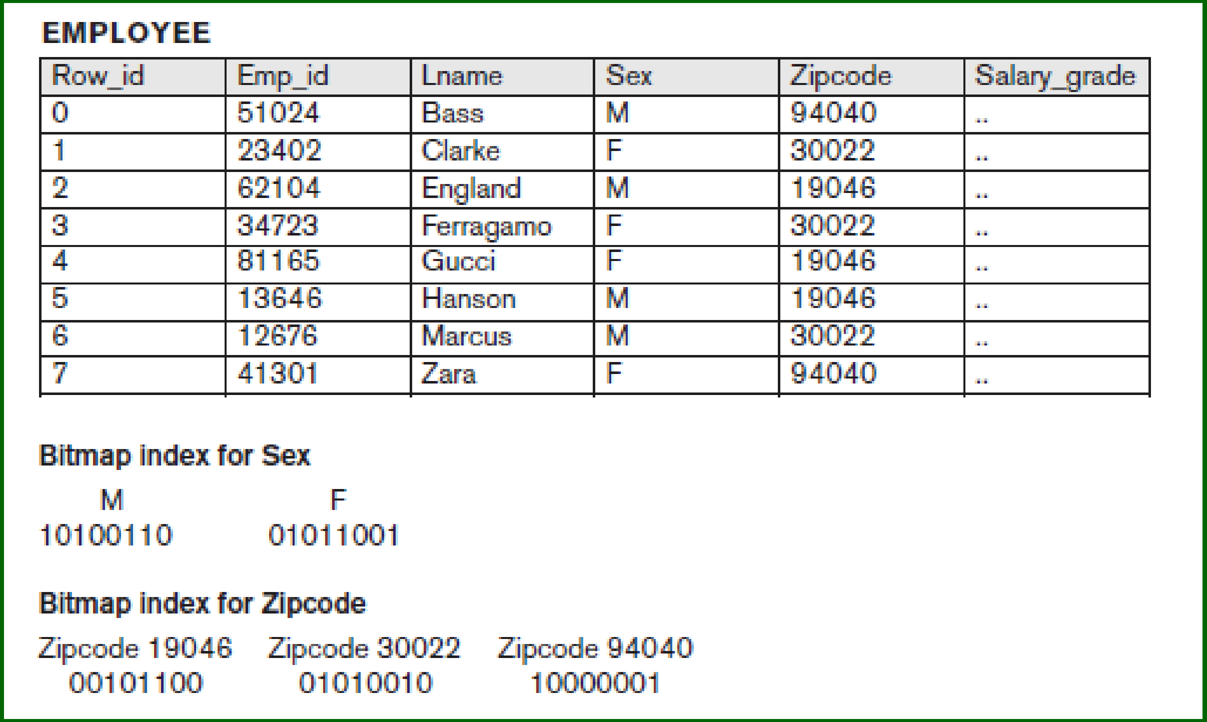
- Assume relation R with
  - 2 attributes A and B.
  - Attribute A is of type Integer and B is of type String.
  - 6 records, numbered 1 through 6 as follows:

	A.	B
1.	30,	foo
2.	30,	bar
3.	40,	baz
4.	50,	foo
5.	40,	bar
6.	30,	baz

- A bit map for attribute B is:

value	Vector
foo	100100
bar	010010
baz	001001

Bitmap Index



EMPLOYEE

Row_id	Emp_id	Lname	Sex	Zipcode	Salary_grade
0	51024	Bass	M	94040	..
1	23402	Clarke	F	30022	..
2	62104	England	M	19046	..
3	34723	Ferragamo	F	30022	..
4	81165	Gucci	F	19046	..
5	13646	Hanson	M	19046	..
6	12676	Marcus	M	30022	..
7	41301	Zara	F	94040	..

Bitmap index for Sex

M	F
10100110	01011001

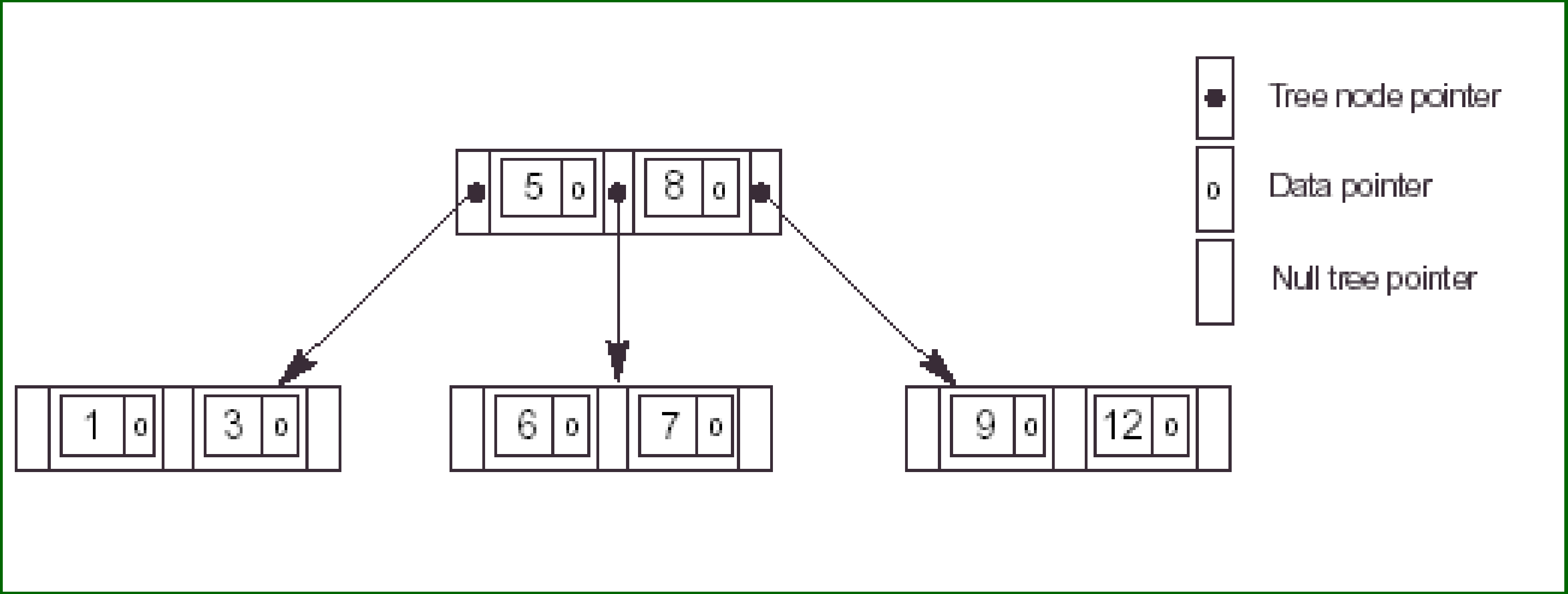
Bitmap index for Zipcode

Zipcode 19046	Zipcode 30022	Zipcode 94040
00101100	01010010	10000001

Dynamic Multilevel index

- To retain the benefits of using multilevel indexing while reducing index insertion and deletion overhead, designers adopt multilevel index that leaves some space in each of its blocks for inserting new entries
- This is called dynamic multilevel index and is often implemented by B-trees and B+ trees.

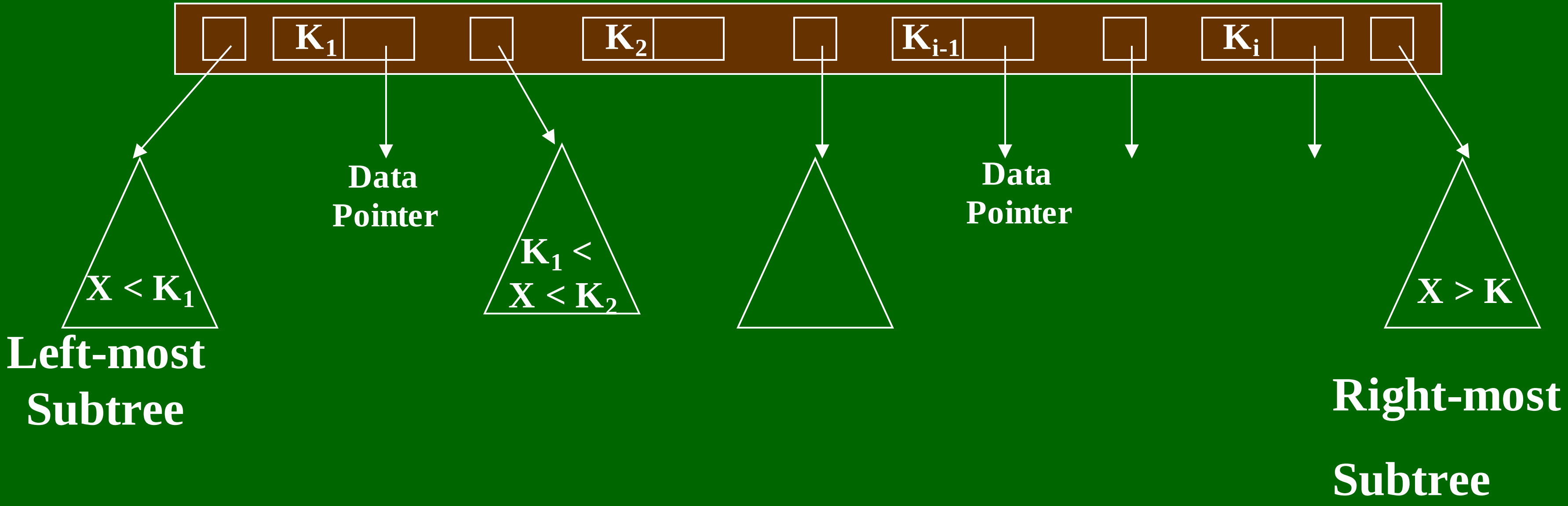
B Tree


5
8
1
3
6
7
9
12
o
Tree node pointer
Data pointer
Null tree pointer

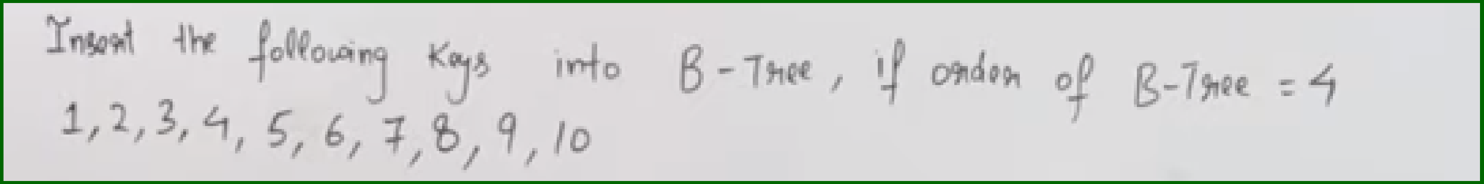
B-Trees

- A Tree data structure where each node has a predetermined maximum fan-out *p*
- Terminologies: root node, leaf nodes, internal nodes, parent, children

Structure of a Node



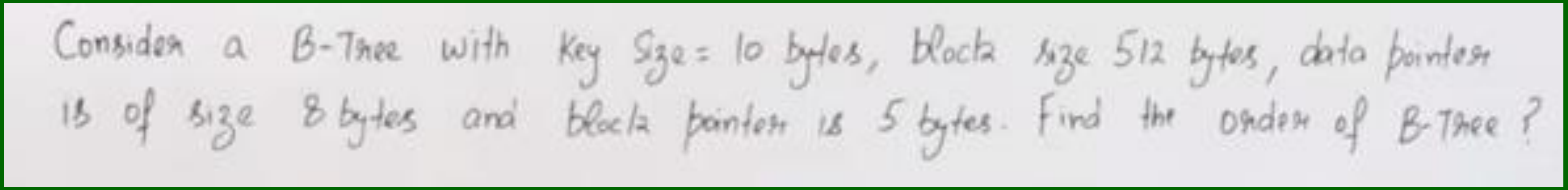
B Tree insertion



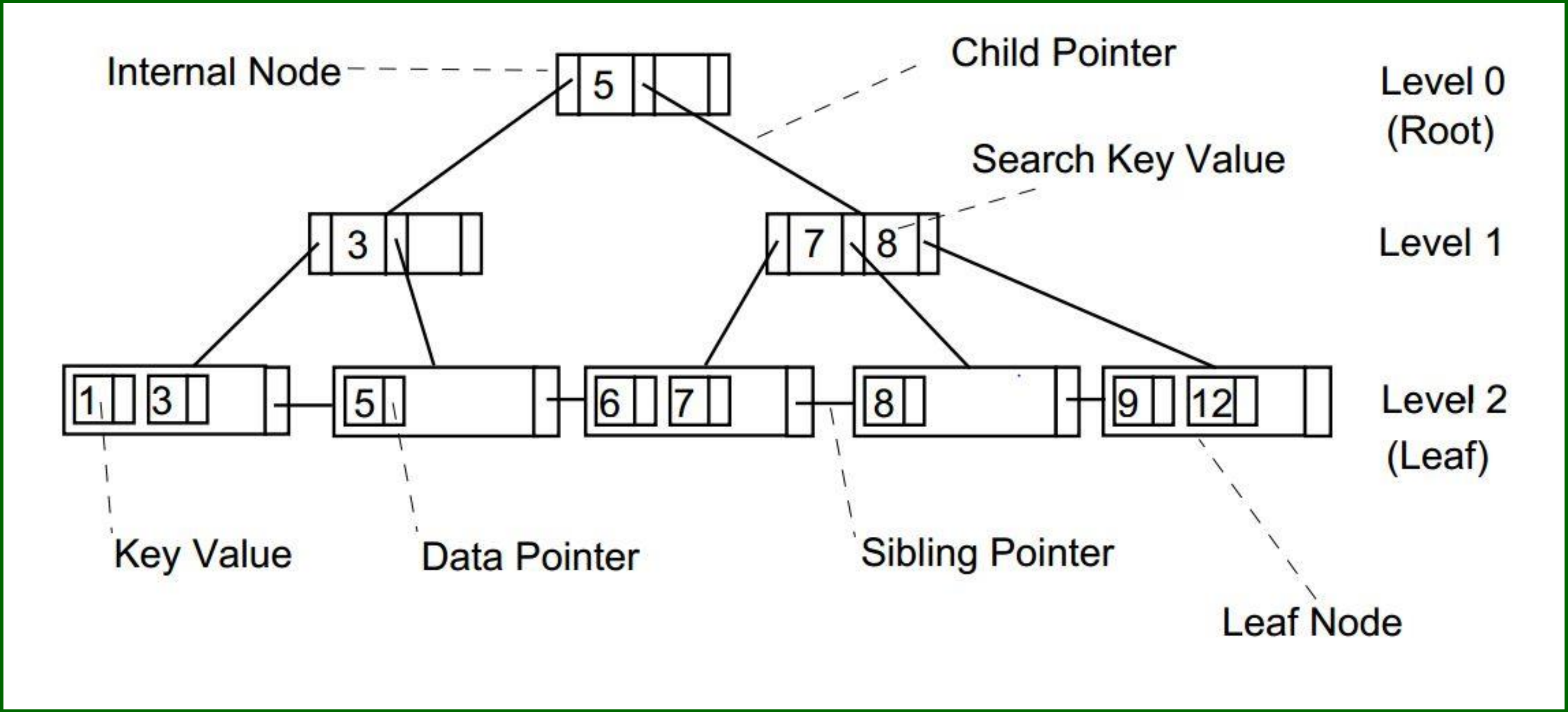
Insert the following keys into B-Tree, if order of B-Tree = 4

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

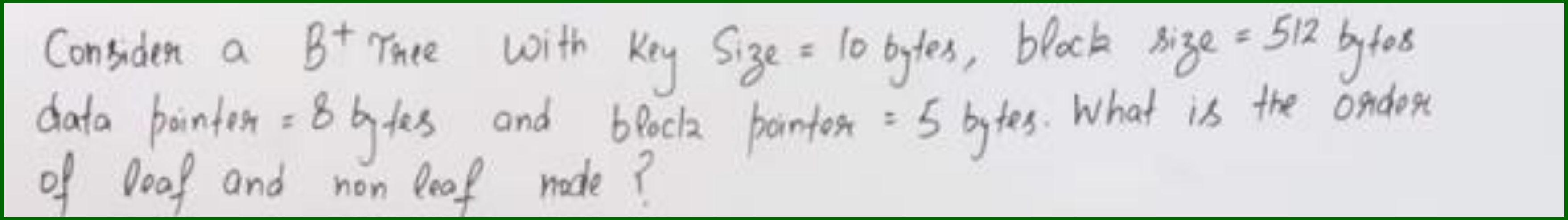
Order of B-Tree


Consider a B-Tree with Key Size = 10 bytes, block size 512 bytes, data pointer is of size 8 bytes and block pointer is 5 bytes. Find the order of B-Tree ?

Structure of B+ Tree



Order of B+ tree



Consider a B^+ Tree with Key Size = 10 bytes, block size = 512 bytes data pointer = 8 bytes and block pointer = 5 bytes. What is the order of leaf and non leaf node?