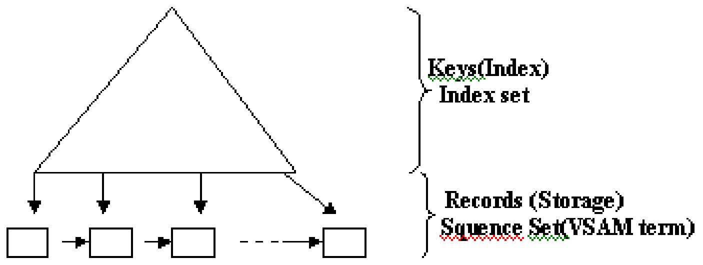
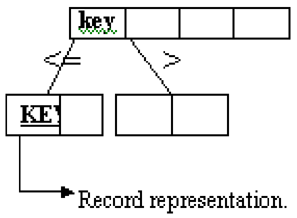
## B+ TREES

A B+ Tree is a B tree variation. Its terminal nodes or leaves are used to store records and nonterminals. Nonterminal nodes are used to store key values.



- Key size <= Record size, so the B+ Trees are shallower (shorter ) than B trees.
- In B+ trees all records require same disk accesses.
- All records are stored at the same level.
- Terminal nodes are conneced each other.

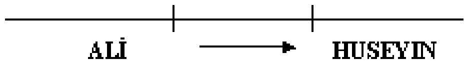


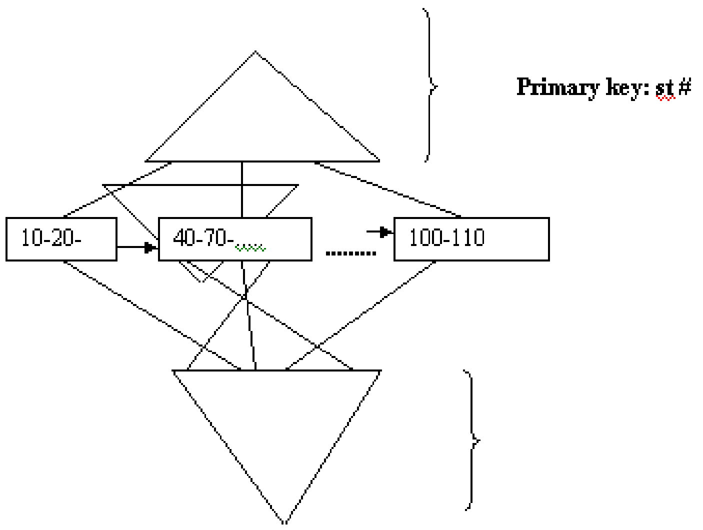
A node stores many keys.

Record representation.

**B+ Tree Characteristics**

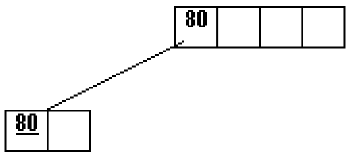
- Dynamic: self organizing
- Balanced at all times.
- Supports an ordering on a large file
- Provides direct access based on a key value.
- Provides efficient range query processing
- Display st major when
  St_name >Ali and st_name> HUSEYIN



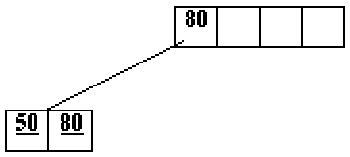


**B+ TREE EXAMPLE**
Records to be inserted to the B+ tree;
80,50,100,90,60,65,70,75,55,64,51,76,77,78,200,300,150
Index node degree =2 (4 key values)
Data nodes degree =1 (2 records)

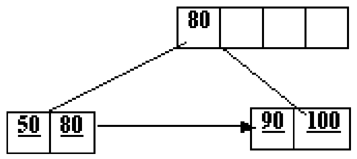
**INSERT 80**



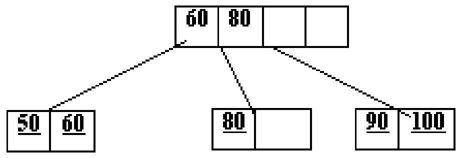
**INSERT 50**



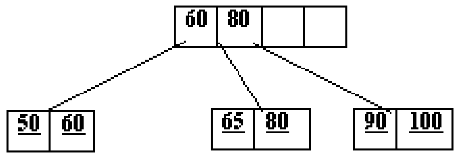
**INSERT 100, 90**



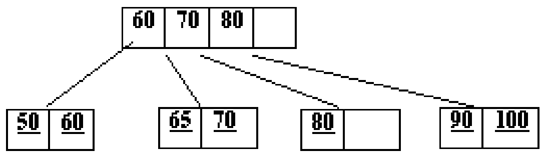
**INSERT 60=>** first node becomes 50, 60, 80. So the middle one 60 goes up



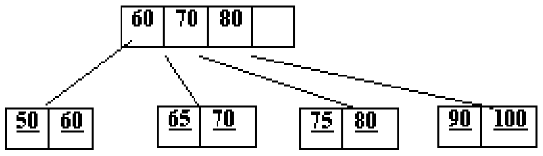
**INSERT 65**



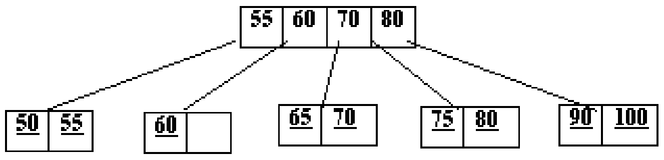
**INSERT 70=>**second node becomes 65,70,80. So the middle one 70 goes up



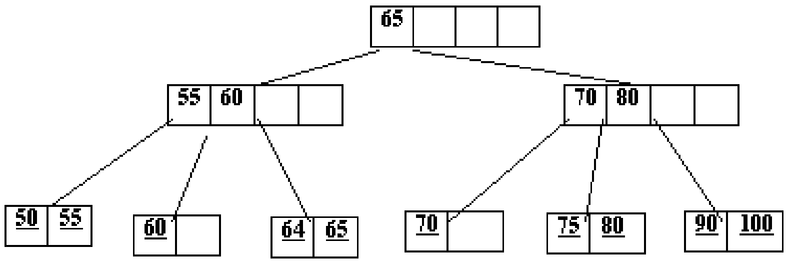
**INSERT 75**



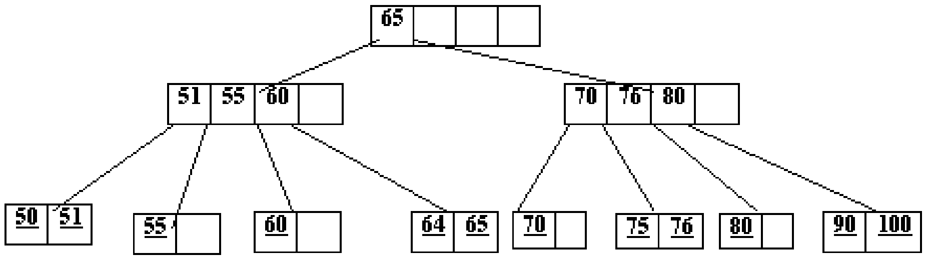
**INSERT 55=>** First node becomes 50,55,60. so The middle one 55 goes up.



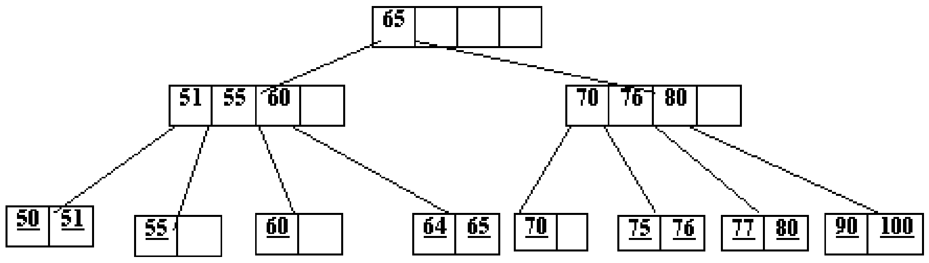
**INSERT 64=>**third node becomes 64, 65, 70, so the middle one 65 goes up. Index node becomes 55,60,65,70,80. so the middle one 65 has to go up one level



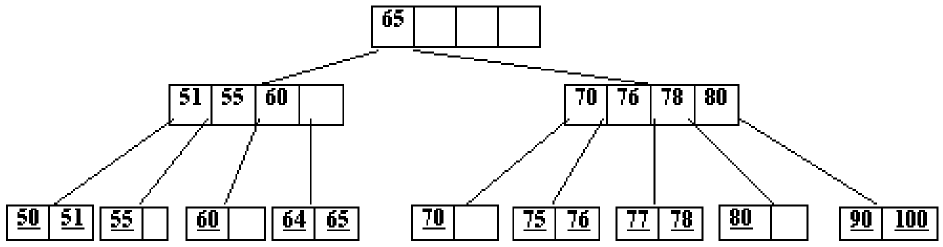
**INSERT 51, 76**



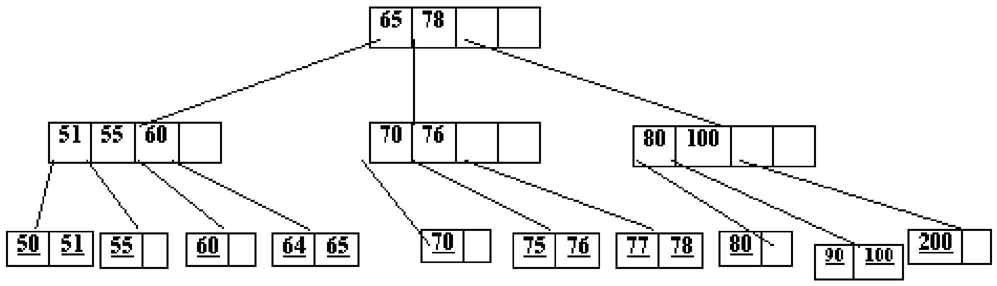
**INSERT 77**



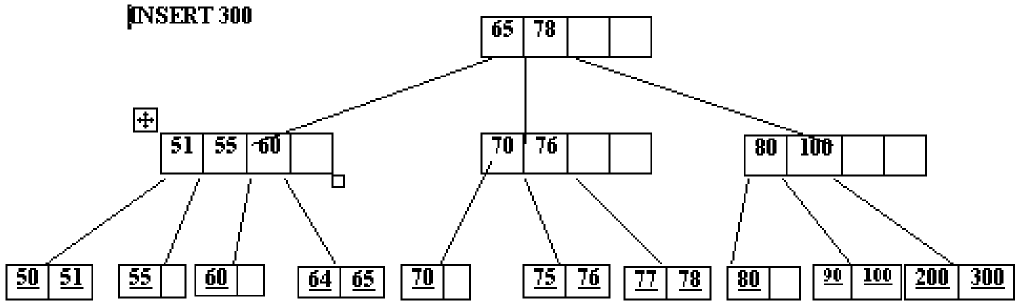
**INSERT 78**



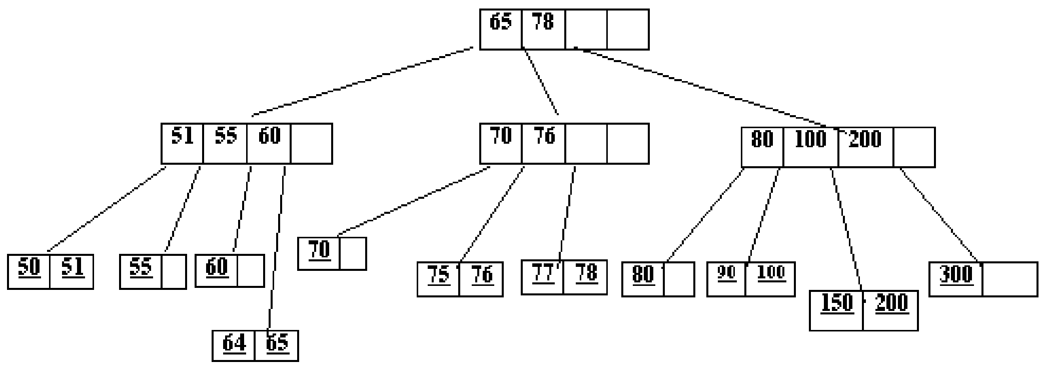
**INSERT 200**=> Last node becomes 90,100, 200. so 100 goes up. Index node becomes 70,76,78,80 and 100. so the middle one 78 has to go up



**INSERT 300**

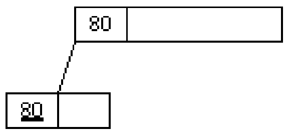


**INSERT 150**



**A B+ tree of degree d (order d) has the following features:**

1. In index nodes no of keys >= d and =< 2d
(root is a special case, see the previous example)
2. All internal nodes with k keys has (k+1) children.
(root may not obey to this rule according to our example)



In the B+ tree two disk accesses are enough

It is shown that each node is ln2 ( 0.70, 70% full). Shown by Yoo, 1978.

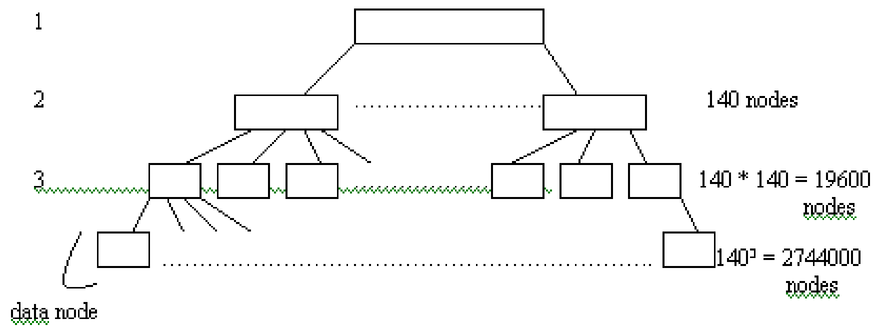
pointer size = 4 bytes
key size = 8 bytes

12 bytes

index node size = 2400 bytes

2400 / 12 = 200 d = 100

A full node can contain 200 key values.
A typical contains 0,7 * 200 = 140 keys



| level | no of nodes | total size for all nodes |
|---|---|---|
| 1 | 1 | 1 * 2400 = 2400 bytes |
| 2 | 140 | 140 * 2400 = 326000 |
| 3 | 19600 | 19600 * 2400 ≅ 47000000 bytes = 47 MB |

key top two levels in main memory

first disk access

2nd disk access $\longrightarrow$ data node

no of records in data nodes

node size / record size = 2400/ 240 = 10

$$\underbrace{(2\,744\,000)}_{\text{no of data nodes}} * \underbrace{0.70 * 10}_{\text{expected failures}} \cong 19\,000\,000 \text{ records}$$

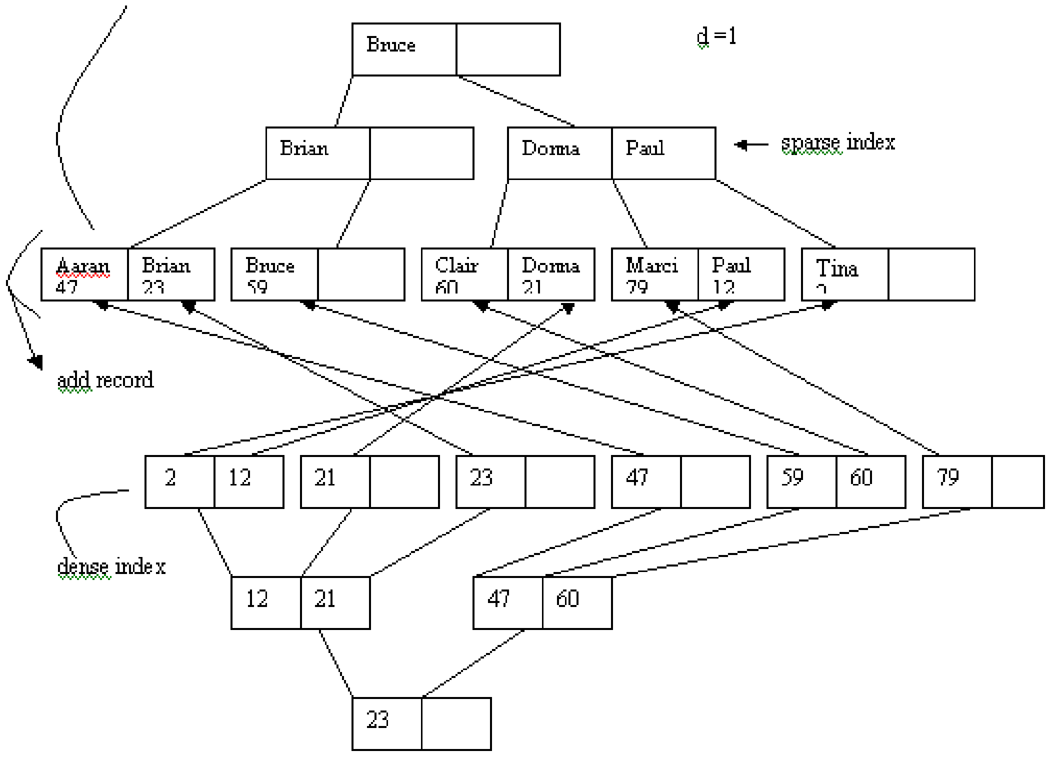
Primary and Secondary Key Based Retrieval

<u>Using B+ trees</u>

| | |
|---|---|
| Bruce, 59 | Clair, 60 |
| Aaran, 47 | Tina, 2 |
| Donna, 21 | Paul, 12 |
| Brian, 23 | Marcia, 79 |

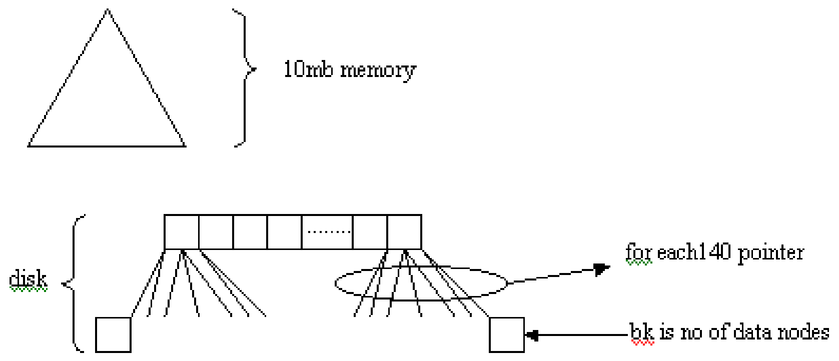
primary key secondary key

The order of records matches the order of index elements. ( index is clustered)



-**With 2 disk accesses how large a file can be handled by a B+ tree.**

Assumption: 10 MB of memory is available for the index tree structure.



bk / 140 = number of index blocks at the parent of leaf nodes

10 MB → 10.000.000 / 2400 = 4167 buckets
bucketsize in bytes

$bk / (140)^2 + bk / (140)^3 + 1 = 4167$

$bk = 140^2 \times \underset{\uparrow}{4167}$ number of available buckets in memory

|

fo: fan out : (expected number of pointers) / (index node)

$bk = fo^2 \times mem \longrightarrow mem = bk / (fo)^2$

**EXERCISE**
Find the expected file size

From the calculations bk= $bk = 140^2 \times 4167$ we obtain 80 million buckets. If each data bucket is one block, 80 million buckets , file size is $2 \times 10^5$ =200000MB.
One disk drive for IBM 3380 is 2400 MB so we get 80 disks as the expected size of the file in IBM 3380.

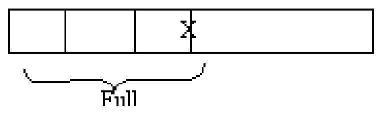
## SIMPLE FILE OPERATIONS USING B+ TREES

**n:** number of nodes
**Bkfr:** number of records/bucket

**Bk:** No of data buckets= n/ ln2 * Bkfr (ln2=0.7)

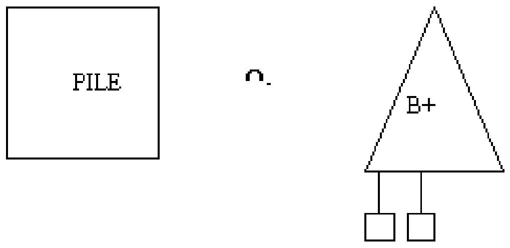
- Time to read the whole file
  Tx (primary) = bk* (s+r+dtt)

- Finding the next record
  TN(primary) = (1/ln2*Bkfr)*(s+r+dtt)
  This is the expected time
  Min time needed=0;
  Max time needed=(s+r+dtt)



**FILE INTERSECTION**

**F1** **F2**



**= RESULTANT FILE**
**(70000 records)**

**n=100.000** **n=100.000**

70% common records.
Record size=400 bytes
File size = 40MB
To create the intersection file look pile, compare in F2 for each two disk accesses.
Time needed to read F1 (Assume that we have 10MB memory ) = b*ebt
Block size =2400 bytes

Bfr= 6
b=100.000/6=16667 => b*ebt=16667*0.84=14 sec
Writing the resultant file
70.000/6=11667 blocks.
11667*0.84=10 sec
Searching the file
2 disk accesses /record
=2* n*(s+r+btt)
=2*100.000*(s+r+btt)=84 min

- Using two sorted files is much shorter= 28 sec

**SECONDARY KEY BASED RETRIEVAL**

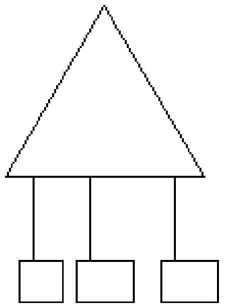
Display employ-name when
Dept_TOY and Age =25

$\underbrace{\text{Emp\_no}}_{\text{Primary atribute}}$ $\underbrace{\text{emp name} \quad \text{dept} \quad \text{age}}_{\text{multiattribute query(secondary attributes-keys)}}$

**Conjunctive query:** ......and....and.....
**Disjunctive query:**..........or.....or.......
**Mixed query:**(and)..or....

**INVERTED FILE STRUCTURE**

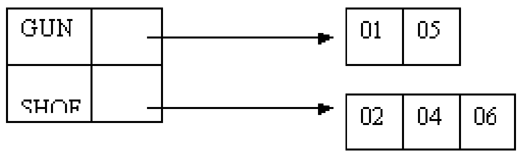
| **Emp no** | **Emp name** | **dept** | **No of dependants** | **Salary** |
|---|---|---|---|---|
| 01 | Jane | Gun | 1 | |
| 02 | George | Shoe | 2 | |
| 04 | Dane | Shoe | 3 | |
| 05 | Mary | Gun | 2 | |
| 06 | Mike | Shoe | 1 | |



**B+ tree on emp no**

**Inverted indexes for secondary attributes**

**Dept** **Posting list**



**EXAMPLES ABOUT THIS WEEK LECTURE NOTES**

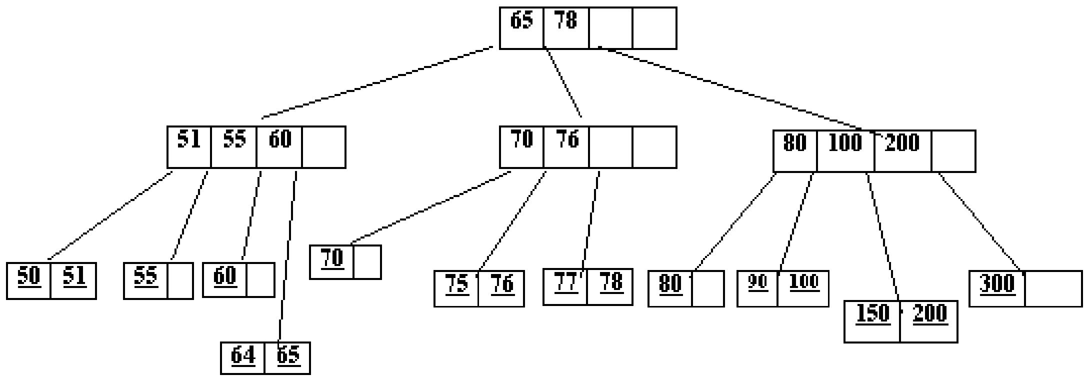
1. There are 8 million records, 400 bytes each records. These files are organized in a B+ tree.
   - How much space does this file takes if ln2 of the space is used in the tree.

     8 million * 400 bytes= 3200 MB is the file size. As only ln2 of the space is used 4500 MB of space is needed.

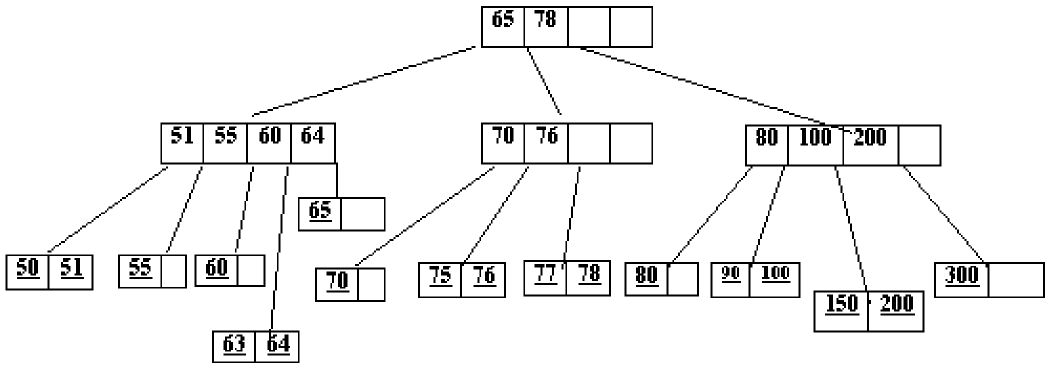
   - Assume that the leaves are 2400 bytes. How many bytes is the parent of leaf level. (totaly a bucket is 12 byte for an address and pointer )

     The number of buckets is 192 000 000 ((8 million * 400/2400)/ln2)
     Each of these buckets needs 12 bytes for an address and pointer. ,
     1 920 000 * 12=22.8 MB is needed for the total buckets.
     But only ln2 space is used so we have to divide this number with ln2 in order to get the actual space needed at parent of leaf level. 22.8/ln2=33MB is the parent of leaf level

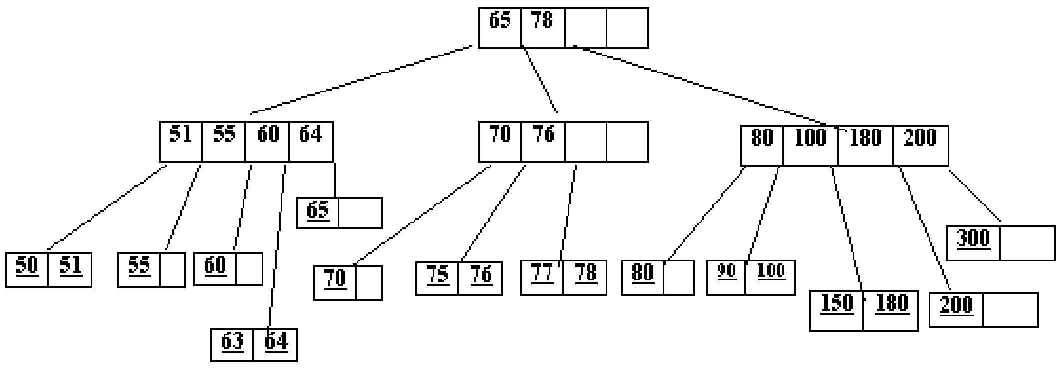
**2.** A Final B+ tree structure is given below. Insert new records with given key values **63, 180, 95.**



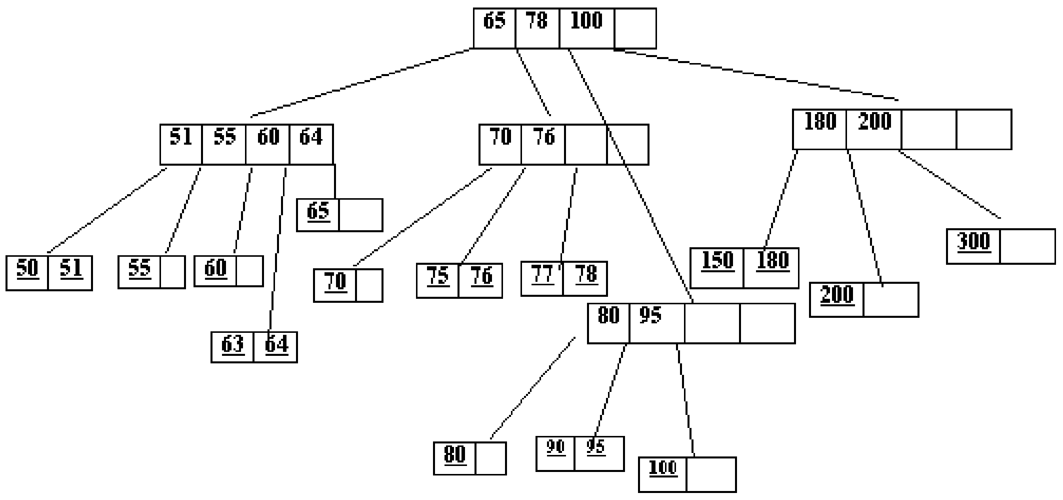
**Insert 63=>** In data field 4th node becomes 63,64,65. so the middle one 64 goes up and tree is structured again.



**Insert 180=>**In the data field 11th node becomes 150, 180, 200. so the middle one 180 goes up. Tree is restructured



**Insert 95**=> In the data field 10th node becomes 90, 95,100. So the middle one 95 goes up. But in the index field 3rd node is full so the middle index 100 goes up one level.


65
78
100
51
55
60
64
70
76
180
200
65
50
51
55
60
63
64
70
75
76
77
78
80
95
150
180
200
300
80
90
95
100