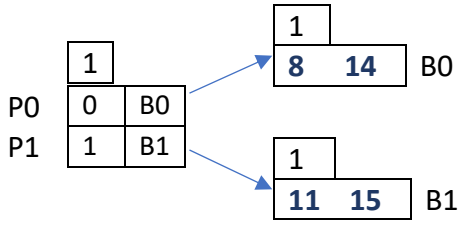
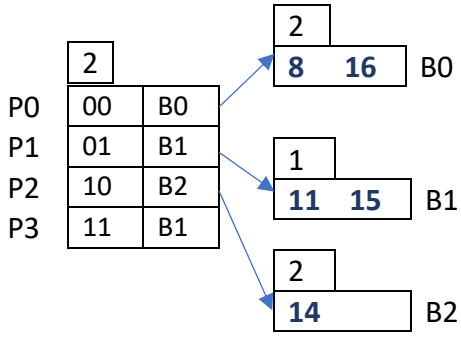
Q1[25 pts]. Insert values: **8, 11, 14, 15, 16, 17, 19, 20, 33, 43, 48** into an expandable hash file using extensible hashing. Assume a Bucket size of 2. Show the structure of the directory each time its structure changes; also show the global and local depths.



Key % 2
8 % 2 = 0 → insert 8 in B0
11 % 2 = 1 → insert 11 in B1
14 % 2 = 0 → insert 14 in B0
15 % 2 = 1 → insert 15 in B1
16 % 2 = 0 → insert 16 in B0 → overflow B0
Local depth (1) = global depth (1):

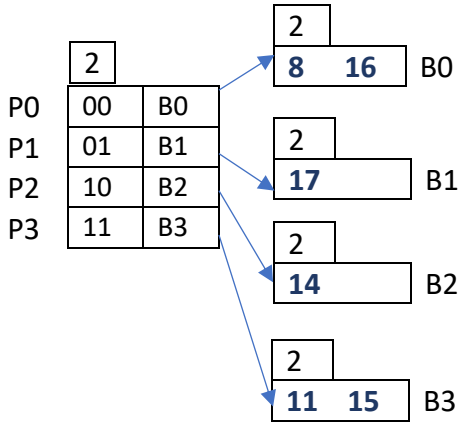
1. Double directory → size = 4
2. Split B0 → create new bucket B2
3. Increase global depth = 2
4. Increase local depths for B0 and B2 = 2
5. Change hash function → **key % 4**



6. Redistribute keys in B0 using new mod function
   8 % 4 = 0 → keep 8 in B0
   14 % 4 = 2 → move 14 to B2
   16 % 4 = 0 → keep 16 in B0

17 % 4 = 1 → insert 17 in B1 → overflow B1
Local depth (1) < global depth (1):

1. Split B1 → create new bucket B3
2. Increase local depths for B1 and B3 = 2

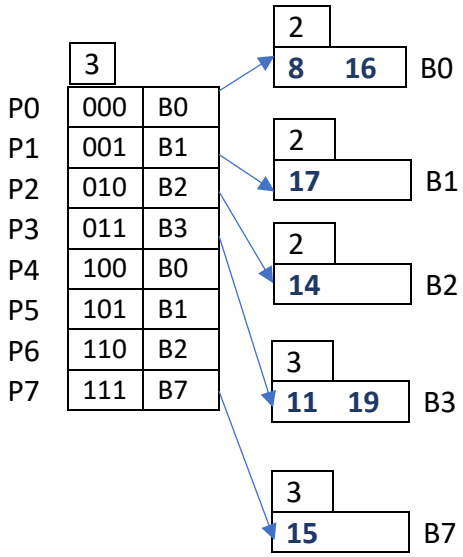


Key % 4

3. Redistribute keys in B1 using new mod function
   11 % 4 = 3 → move 11 to B3
   15 % 4 = 3 → move 15 to B3
   17 % 4 = 1 → keep 17 in B1

19 % 4 = 3 → insert 19 in B3 → overflow B3
Local depth (2) = global depth (2):

1. Double directory → size = 8
2. Split B3 → create new bucket B7
3. Increase global depth = 3
4. Increase local depths for B3 and B7 = 3
5. Change hash function → **key % 8**

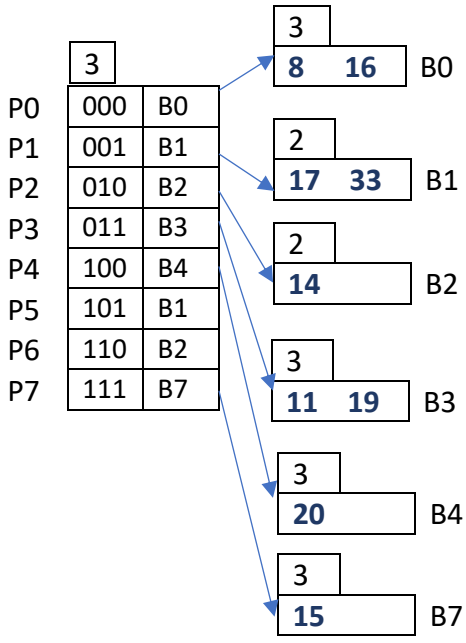


Key % 8

6. Redistribute keys in B3 using new mod function
   11 % 8 = 3 → keep 11 in B3
   15 % 8 = 7 → move 15 to B7
   19 % 8 = 3 → keep 19 in B3

20 % 8 = 4 → P4 points to B0 → overflow B0
Local depth (2) < global depth (3):

1. Split B0 → create new bucket B4
2. Increase local depths for B0 and B4 = 3

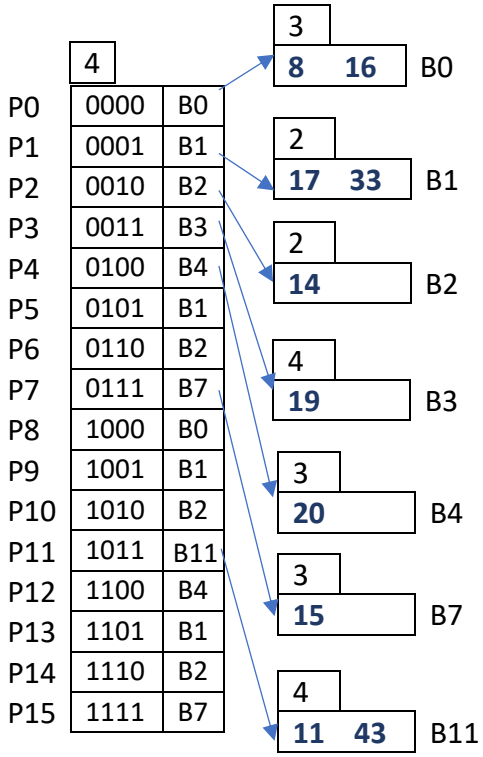


Key % 8

3. Redistribute keys in B0 using new mod function
   8 % 8 = 0 → keep 8 in B0
   16 % 8 = 0 → keep 16 in B0
   20 % 8 = 4 → move 20 to B4

33 % 8 = 1 → insert 33 in B1
43 % 8 = 3 → insert 43 in B3 → overflow B3
Local depth (3) = global depth (3):

1. Double directory → size = 16
2. Split B3 → create new bucket B11
3. Increase global depth = 4
4. Increase local depths for B3 and B11 = 4
5. Change hash function → **key % 16**



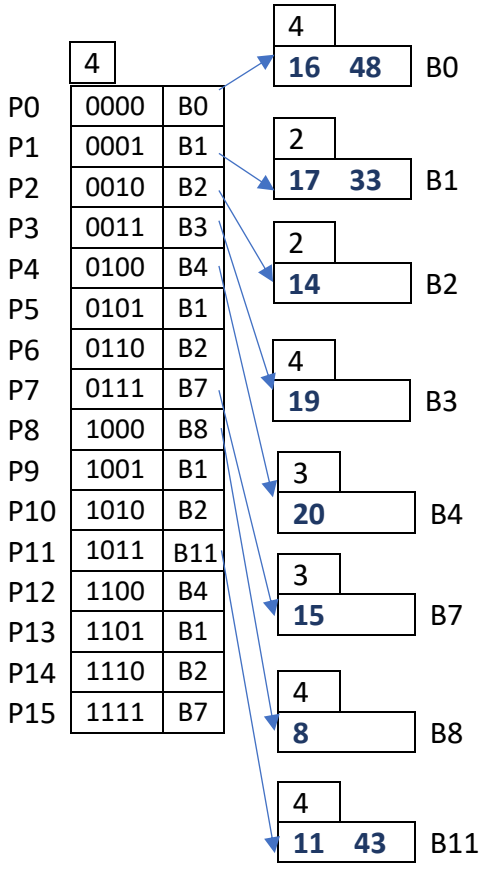
Key % 16

6. Redistribute keys in B3 using new mod function
   11 % 16 = 11 → move 11 to B11
   19 % 16 = 3 → keep 19 in B3
   43 % 16 = 11 → move 43 to B11

48 % 16 = 0 → insert 48 in B0 → overflow B0

Local depth (3) < global depth (4):

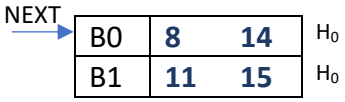
1. Split B0 → create new bucket B8
2. Increase local depths for B0 and B8 = 4

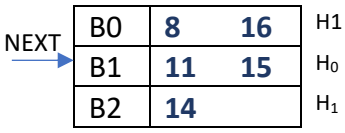


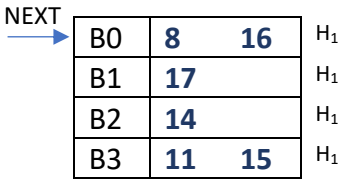
Key % 16

3. Redistribute keys in B0
   8 % 16 = 8 → move 8 to B8
   16 % 16 = 0 → keep 16 in B0
   48 % 16 = 0 → keep 48 in B0

Q2[25 pts]. Insert the records of Q1into an expandable hash file using linear hashing. Assume a Bucket size of 2. Show how the file grows and how the hash functions change as the records are inserted. Assume that blocks are split whenever an overflow occurs.







R0 [N0 = 2, H0 = Key % 2]

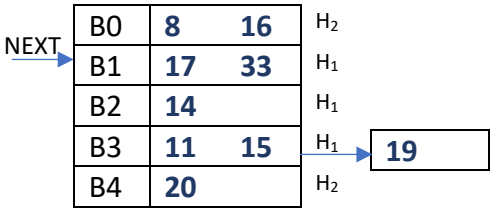
8 % 2 = 0 → insert 8 in B0
11 % 2 = 1 → insert 11 in B1
14 % 2 = 0 → insert 14 in B0
15 % 2 = 1 → insert 15 in B1
16 % 2 = 0 → insert 16 in B0 → overflow:

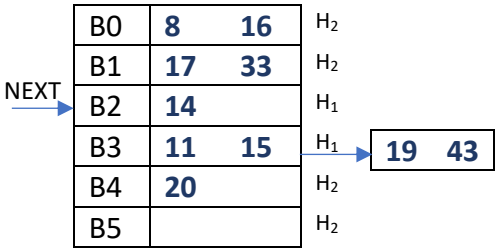
1. Split bucket with NEXT pointer → B0
2. Create new bucket → B2
3. Create new hash function → **H1 = Key % 4**
4. Both B0 and B2 will use H1
5. Rehash splitted bucket B0 using H1:
   a. 8 % 4 = 0 → keep 8 in B0
   b. 14 % 4 = 2 → move 14 to B2
   c. 16 % 4 = 0 → keep 16 in B0
6. Move the NEXT pointer to B1

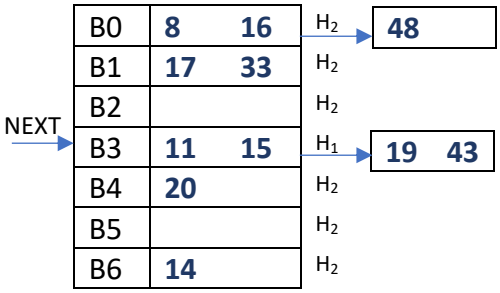
---

17 % 2 = 1 → insert 17 in B1 → overflow:

1. Split bucket with NEXT pointer → B1
2. Create new bucket → B3
3. Both B0 and B2 will use H1
4. Rehash splitted bucket B1 using H1:
   a. 11 % 4 = 3 → move 11 to B3
   b. 15 % 4 = 3 → move 15 to B3
   c. 17 % 4 = 1 → keep 17 in B1
5. Round R0 is finished → set NEXT pointer to B0







R1 [N1 = 4, H1 = Key % 4]

19 % 4 = 3 → insert 19 in B3 → overflow:

1. Split bucket with NEXT pointer → B0
2. Create new bucket → B4
3. Create new hash function → **H2 = Key % 8**
4. Both B0 and B4 will use H2
5. Rehash splitted bucket B0 using H2:
   a. 8 % 8 = 0 → keep 8 in B0
   b. 16 % 8 = 0 → keep 16 in B0
6. Move the NEXT pointer to B1

20 % 4 = 0 → B0 is before the NEXT pointer and it uses H2 not H1, so we have to rehash 20 using H2:
20 % 8 = 4 → insert 20 in B4
33 % 4 = 1 → insert 33 in B1

---

43 % 4 = 3 → insert 43 in B3 → overflow:

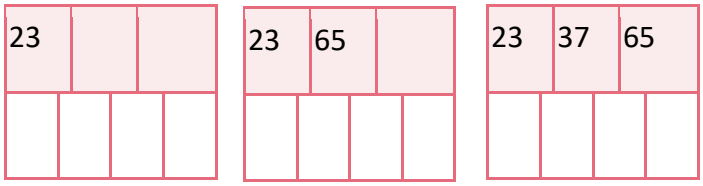
1. Split bucket with NEXT pointer → B1
2. Create new bucket → B5
3. Both B1 and B5 will use H2
4. Rehash splitted bucket B1 using H2:
   a. 17 % 8 = 1 → keep 8 in B1
   b. 33 % 8 = 1 → keep 33 in B1
5. Move the NEXT pointer to B2

---

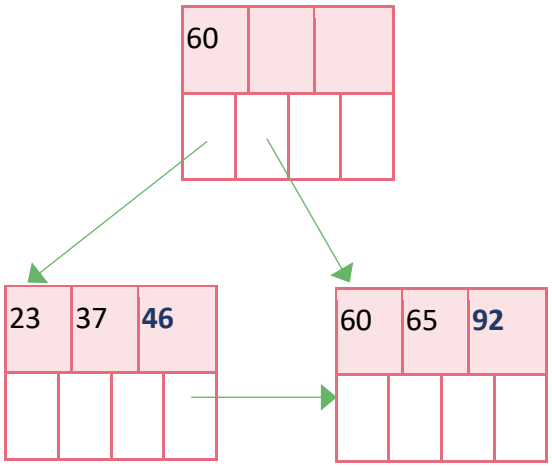
48 % 4 = 0 → B0 is before the NEXT pointer and it uses H2 not H1, so we have to rehash 48 using H2:
48 % 8= 0 → insert 48 in B0 → overflow:

1. Split bucket with NEXT pointer → B2
2. Create new bucket → B6
3. Both B2 and B6 will use H2
4. Rehash splitted bucket B2 using H2:
   a. 14 % 8 = 6 → move 14 to B6
5. Move the NEXT pointer to B3

Q3[25 pts]. Insert values: **23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21** in the given order in a B+ Tree. Suppose that the maximum entries per node is n = 3; show how the tree will expand and what the final tree will look like.

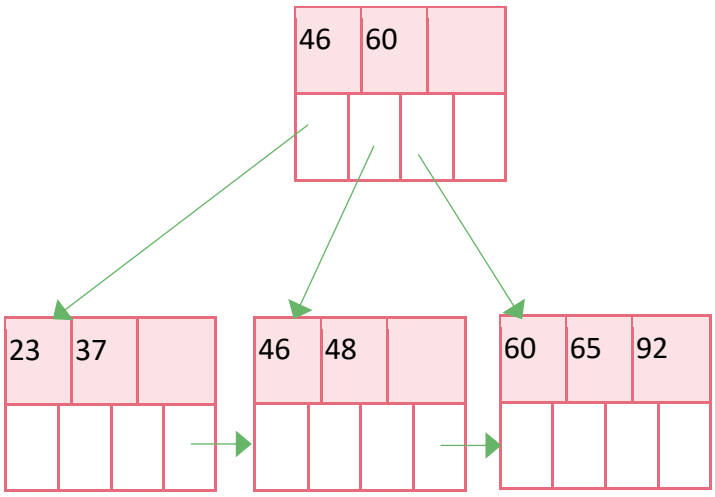


Insert 23, 65, 37…
Insert 60 → node full:



1. Split
2. Sort the numbers → 23, 37, 60, 65
3. Create new node & redistribute the values
4. Copy 60 to the root
5. Rearrange the pointers

Insert 46, 92 …
Insert 48 → left node full:

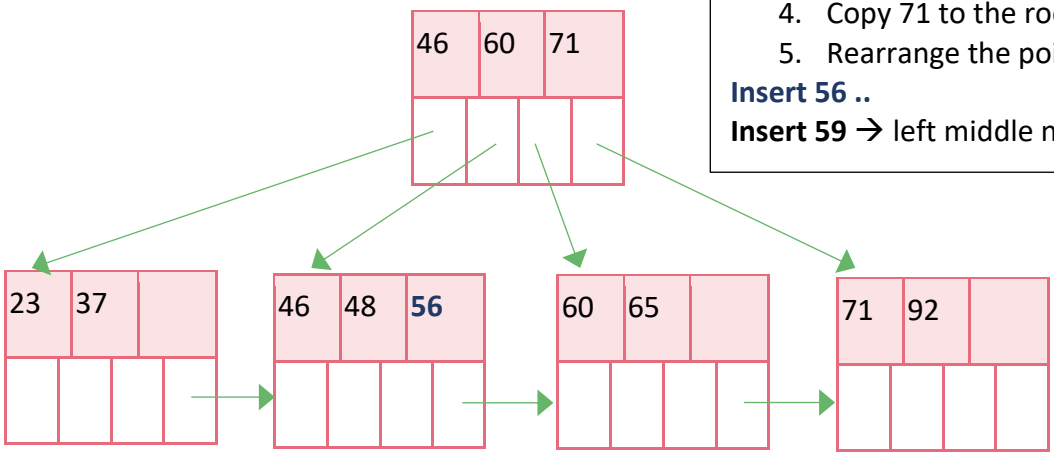


1. Split
2. Sort the numbers → 23, 37, 46, 48
3. Create new Node & redistribute the values
4. Copy 46 to the root
5. Rearrange the pointers

Insert 71 → right node full:

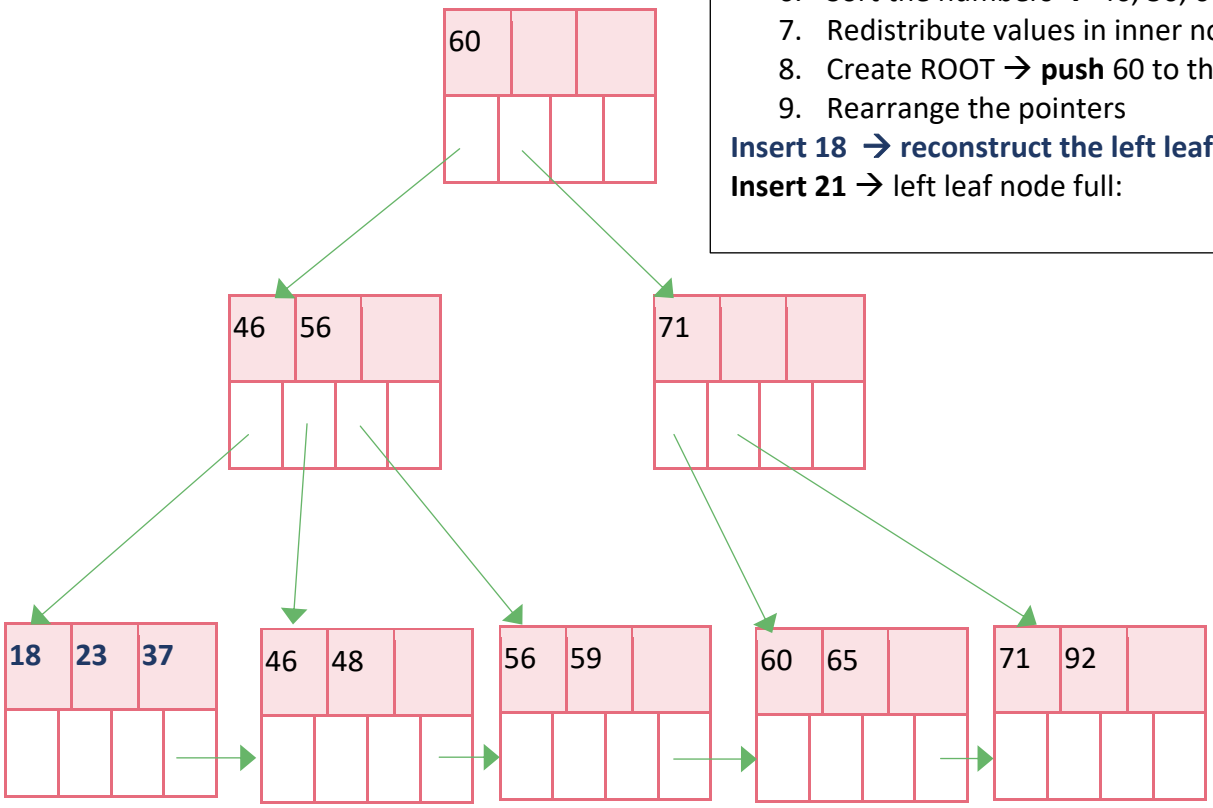
1. Split
2. Sort the numbers → 60, 65, 71, 92
3. Create new Node & redistribute the values
4. Copy 71 to the root
5. Rearrange the pointers

Insert 56 ..
Insert 59 → left middle node full:

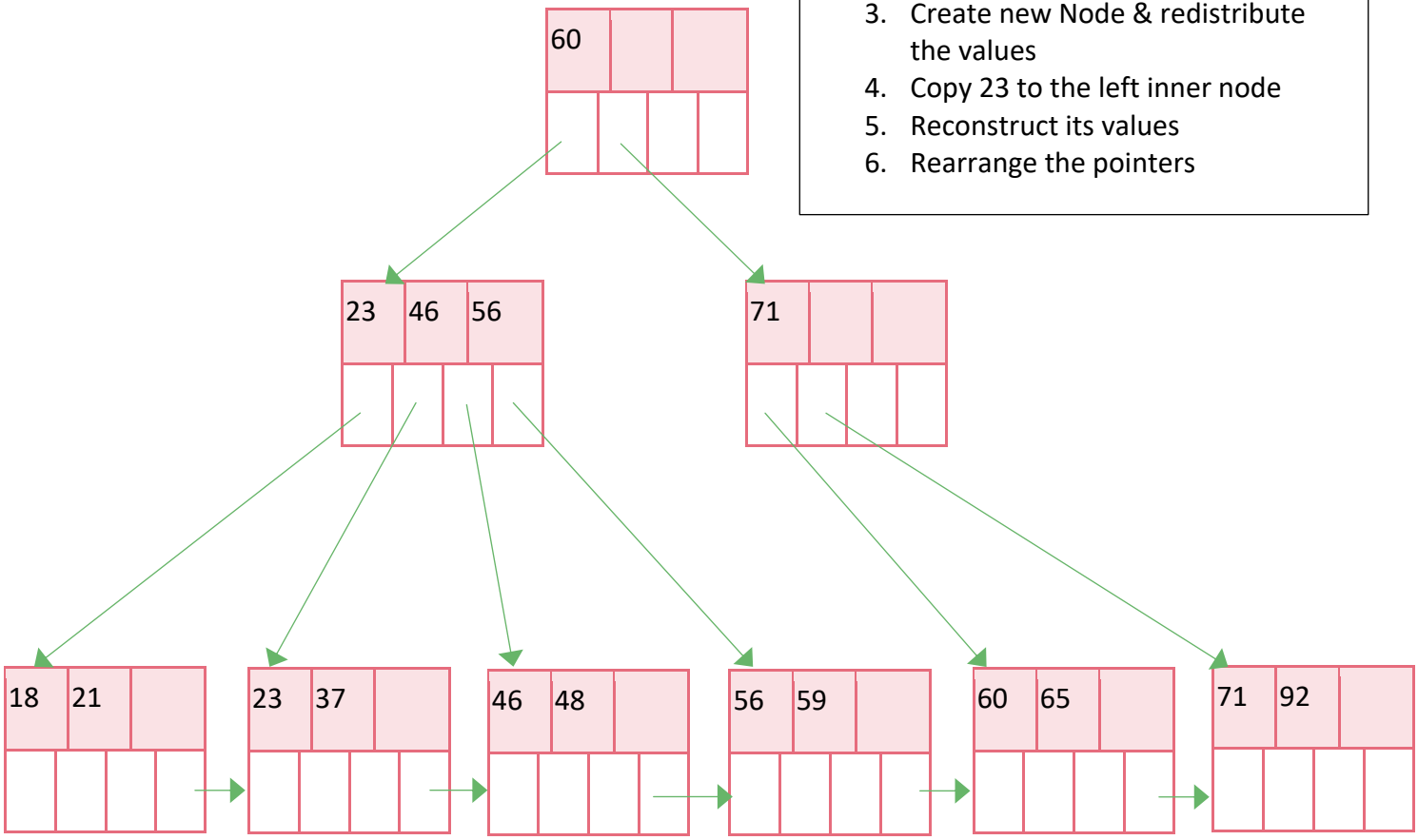


1. Split
2. Sort the numbers → 46, 48, 56, 59
3. Create new Node & redistribute the values
4. Copy 56 to the root → ROOT FULL!
5. Split the root → 2 inner nodes
6. Sort the numbers → 46, 56, 60, 71
7. Redistribute values in inner nodes
8. Create ROOT → **push** 60 to the root
9. Rearrange the pointers

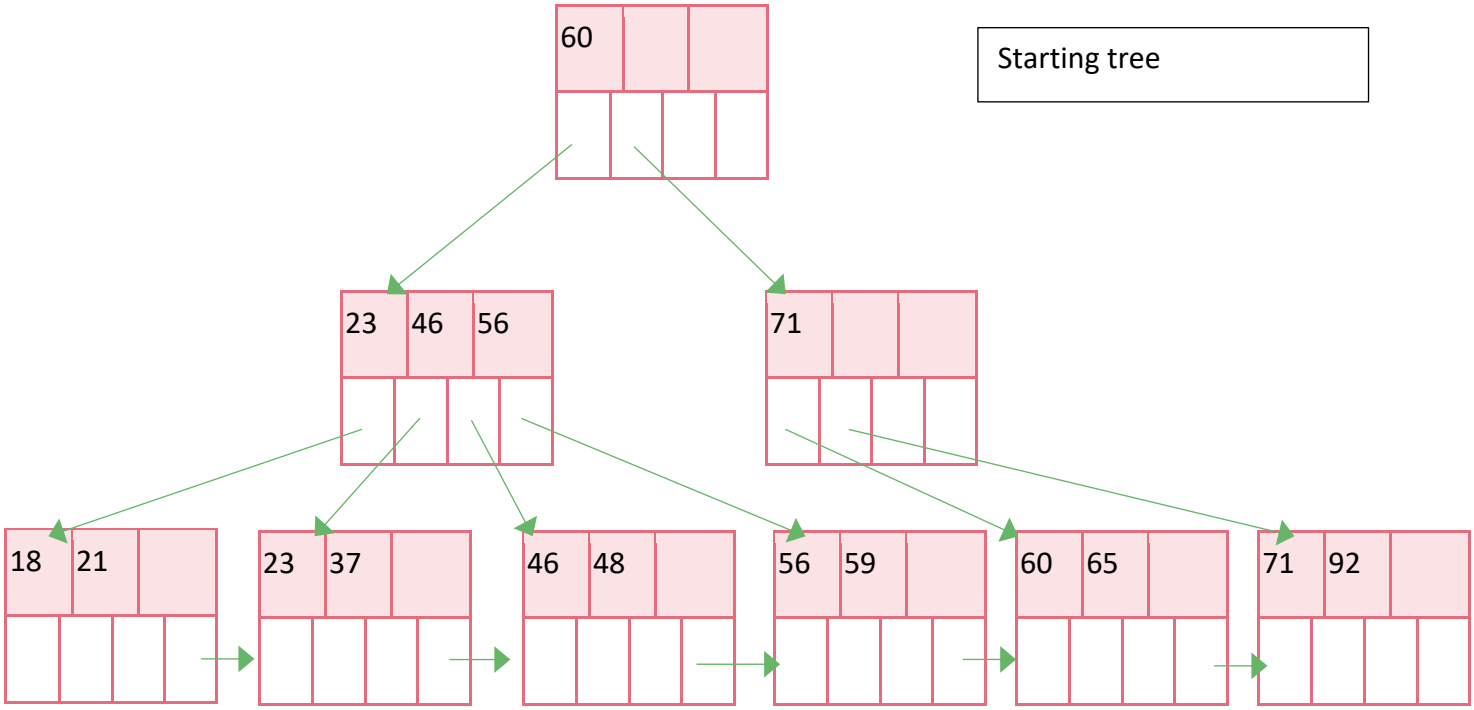
Insert 18 → reconstruct the left leaf node
Insert 21 → left leaf node full:

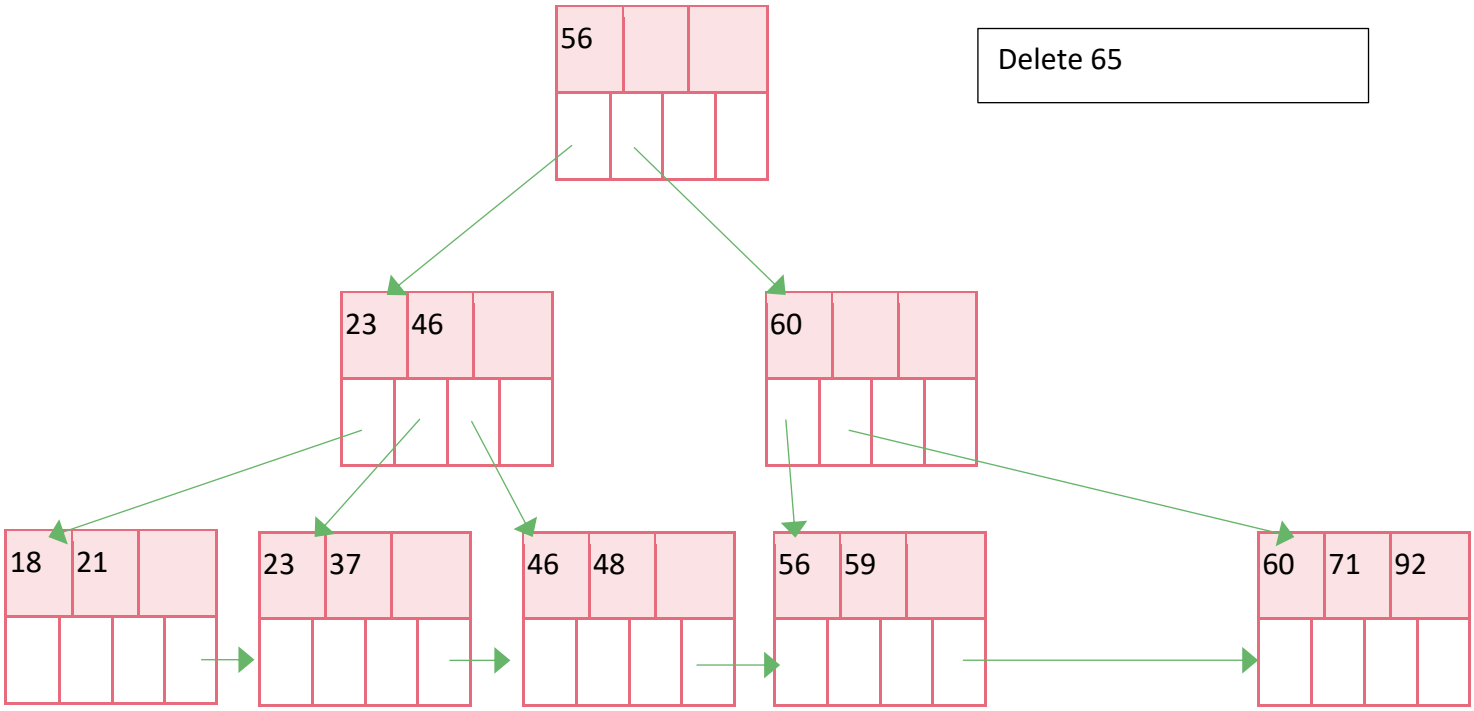


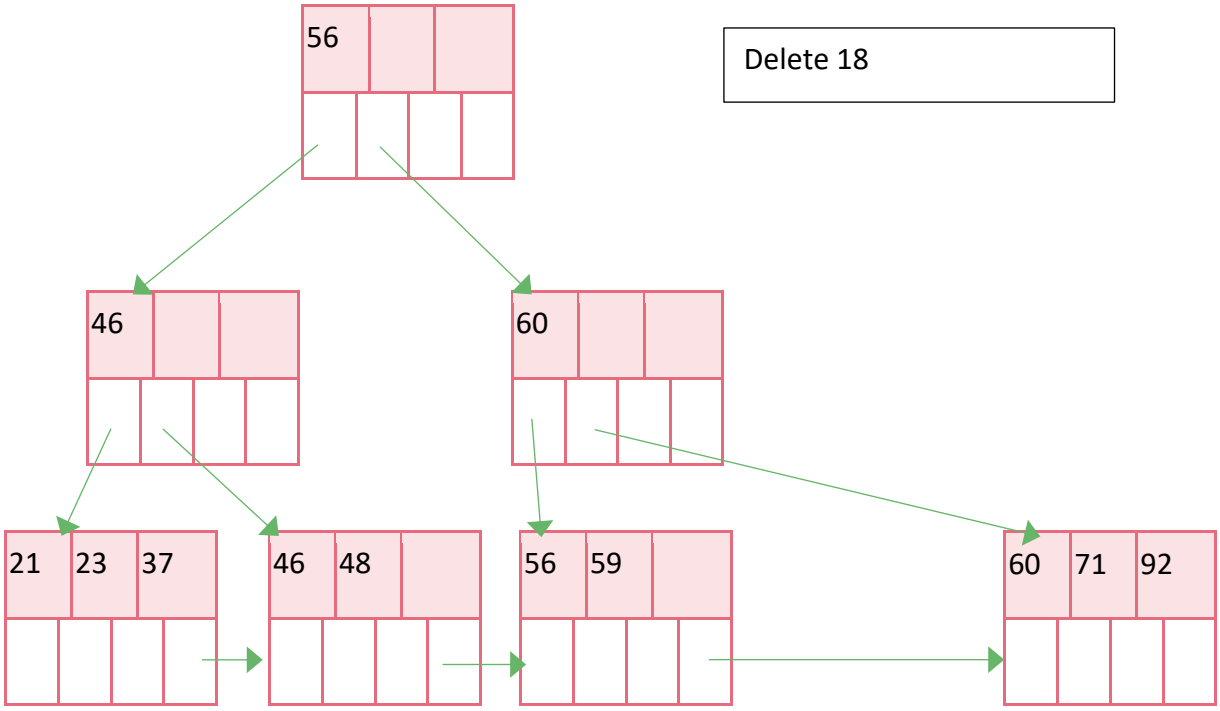
1. Split
2. Sort the numbers → 18, 21, 23, 37
3. Create new Node & redistribute the values
4. Copy 23 to the left inner node
5. Reconstruct its values
6. Rearrange the pointers

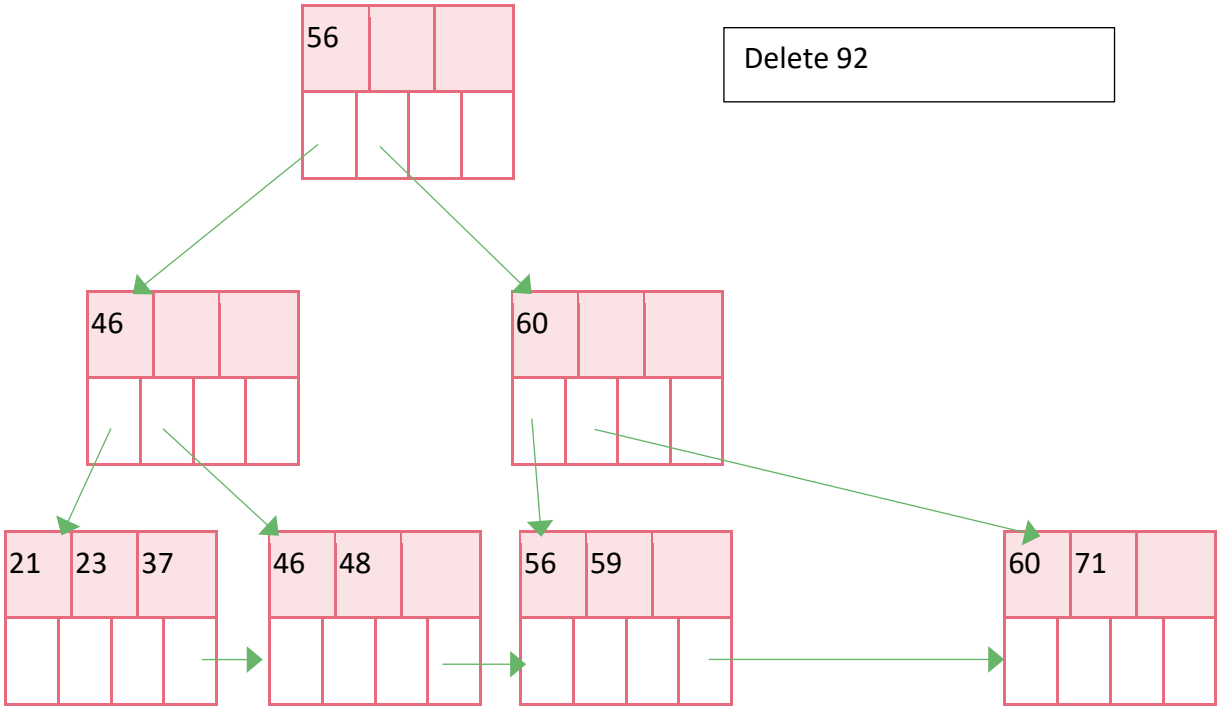


Q4[25pts]. Suppose that the following search field values are deleted, in the given order, from the B+ Tree of Q3. Show how the tree will shrink and show the final tree. The deleted values are **65, 18, 92, 59, 37.**

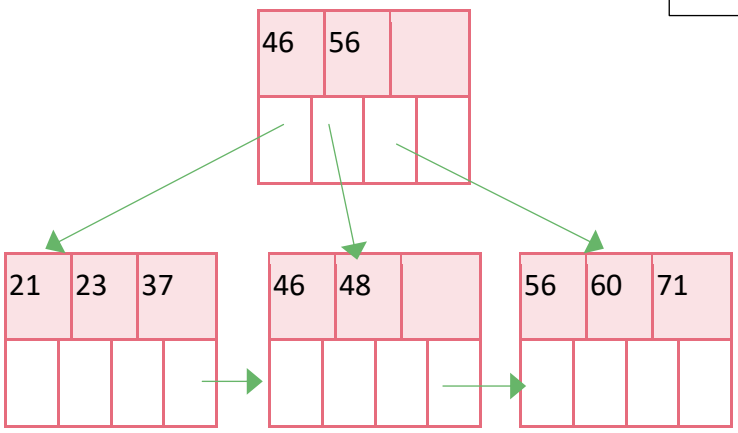





56
Delete 18
46
60
21
23
37
46
48
56
59
60
71
92


56
Delete 92
46
60
21
23
37
46
48
56
59
60
71

Delete 59



Delete 37