Qatar University
College of Engineering
Computer Science and Engineering Department

Advanced Database Systems

Homework 1- Solution

Exercise 1

(a)

Record Size(R) = 9+20+20+1+10+35+12+4+8+1=120 bytes

Bfr=floor (B/R)=2.4*1024bytes/120bytes=20records/block

Number of disk blocks(b)=ceiling(# records/bfr)=30,000/20=1500 blocks

(b)

Waste Space=Block size-(bfr*record size)=2.4*1024-(20*120)=57.6 bytes

(c)

(i) Average number of blocks accessed in linear search=half the file blocks

accessed blocks=1500/2=750 blocks

(ii) Average number of accessed blocks (ordered blocks) in binary search =ceiling(log2(#b))

#accessed blocks=ceiling(log2(1500))=11 blocks

Exercise 2

For extensible hashing we always start with 2 buckets and a Hash Function = mod 2.

After rehash insert 1, 16, 20,7, 27

Initially hash function=mod 2

1	
0	→ B_0
1	→ B_1

1		
16	20	B_0

1		
1	7	B_1

Inserting **27** will cause B_1 to overflow. Since GD=LD →we need to (1) Double the directory (2) Split B_1 (3) then rehash B_1 values using a hash function = mod 4

Insert 29, 18

2	
00	→ B0
01	→ B1
10	→ B0
11	→ B3

1		
16	20	B0

2		
1	29	B1

2		
7	27	B3

B_0 overflow=18

B_0 LD is less than GD →split B_0 and rehash

Insert 11

2	
00	→ B0
01	→ B1
10	→ B2
11	→ B3

B0 (LD 2)	
16	20

B1 (LD 2)	
1	29

B2 (LD 2)	
18	

B3 (LD 2)	
7	27

B_3 overflow=11

B_3 LD is equal to GD → double directory, split B_3 then rehash B_3 using mod 8

Insert 22, 28

3	
000	→ B0
001	→ B1
010	→ B2
011	→ B3
100	→ B0
101	→ B1
110	→ B2
111	→ B7

B0 (LD 2)	
16	20

B1 (LD 2)	
1	29

B2 (LD 2)	
18	22

B3 (LD 3)	
27	11

B7 (LD 3)	
7	

B_0 overflow=28

B_0 LD is less than GD → split B_0 and rehash

Insert 9

3	
000	→ B0
001	→ B1
010	→ B2
011	→ B3
100	→ B4
101	→ B1
110	→ B2
111	→ B7

3			
16			B0
2			
1	29		B1
2			
18	22		B2
3			
27	11		B3
3			
20	28		B4
3			
7			B7

B_1 overflow=9

B_1 LD is less than GD → split B_1 and rehash

Insert 14

3	
000	→ B0
001	→ B1
010	→ B2
011	→ B3
100	→ B4
101	→ B5
110	→ B2
111	→ B7

Local depth			Bucket
3	16		B0
3	1	9	B1
2	18	22	B2
3	27	11	B3
3	20	28	B4
3	29		B5
3	7		B7

B_2 overflow =14

B_2 LD is less than GD →split B_2 and rehash

3	
000	→ B0
001	→ B1
010	→ B2
011	→ B3
100	→ B4
101	→ B5
110	→ B6
111	→ B7

Local depth			Bucket
3	16		B0
3	1	9	B1
2	18		B2
3	27	11	B3
3	20	28	B4
3	29		B5
3	22	14	B6
3	7		B7

Exercise 3

For linear hashing we always start with 2 buckets and a Hash Function = mod 2. Also we should show the bucket address in binary format (e.g., 00, 01, 10, 11).

Note that linear hashing use a family of hash functions H_0, H_1, H_2, ...

$H_i(key)$ = *key* mod$(2^i N_o)$; N_0 = initial # buckets

N_0 in our case is = 2. Hence $H_0(key)$ = *key* mod$(2^0 N_o)$ = key mod 2

Insert 1, 16, 20, 7, 27

H_0=mod 2

Next→	0	16	20			H_0
	1	1	7	→	27	H_0

Inserting 27 causes an overflow, we split the bucket Next→ is pointing to (i.e., B_0) and we redistribute the values using a new Hash function H_1 = mod 4. We also move the Next→ pointer to the next bucket B_1. Note that B_1 is still using the Hash Function H_0.

H_1=mod 4

	00	B_0	16	20			H_1
Next→	1	B_1	1	7	→	27	H_0
	10	B_2					H_1

Insert 29

	00	B_0	16	20				
Next→	1	B_1	1	7	→	27	29	H_0
	10	B_2						

Inserting 29 an overflow, we split the bucket Next→ is pointing to (i.e., B_1) and we redistribute the values using a new Hash function H_1 = mod 4. We also move the Next→ pointer to the bucket B_0 because the Round has finish and we need to start another Round (In $Round_0$ we started with N_0 = 2 Buckets and we spitted both, now it is time to start $Round_1$ with N_1 = 4). Note that all buckets are now using the new Hash Function H_1 and that's another indicator that we need to start another round. Linear hashing splitting proceeds in `rounds'. Round ends when all N_R initial (for round R) buckets are split.

Next→	00	B_0	16	20	H_1
	01	B_1	1	29	H_1
	10	B_2			H_1
	11	B_3	7	27	H_1

Insert 18, 11

Next→	00	B_0	16	20		H_1
	01	B_1	1	29		H_1
	10	B_2	18			H_1
	11	B_3	7	27	→ 11	H_1

Insert 22, 28, and 9

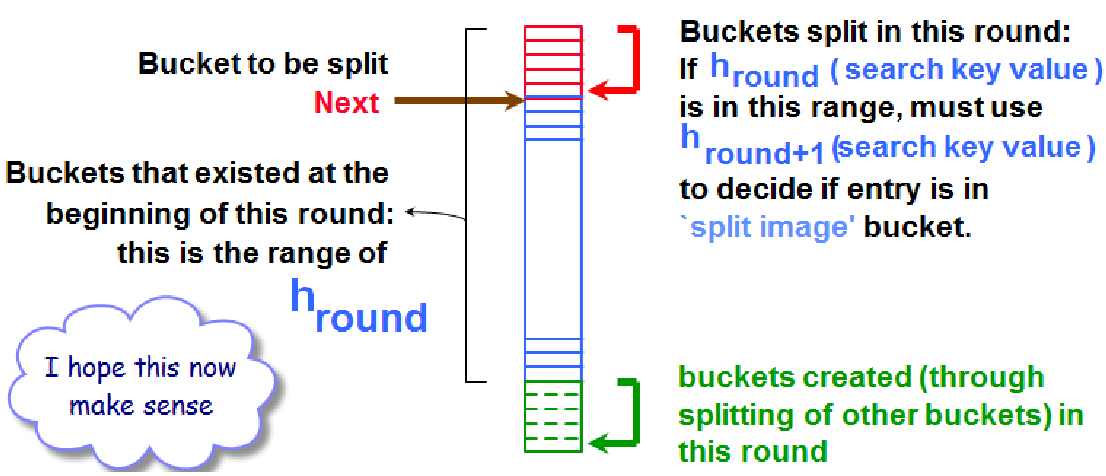
H_2=mod 8

	000	B_0	16			H_2
Next→	01	B_1	1	29	→ 9	H_1
	10	B_2	18	22		H_1
	11	B_3	7	27	→ 11	H_1
	100	B_4	20	28		H_2

Insert 14

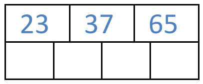
	000	B_0	16			H_2
	001	B_1	1	9		H_2
Next→	10	B_2	18	22	→ 14	H_1
	11	B_3	7	27	→ 11	H_1
	100	B_4	20	28		H_2
	101	B_5	29			H_1

	000	B_0	16			H_2
	001	B_1	1	9		H_2
	010	B_2	18			H_2
Next→	11	B_3	7	27	→ 11	H_1
	100	B_4	20	28		H_2
	101	B_5	29			H_2
	110	B_6	22	14		H_2

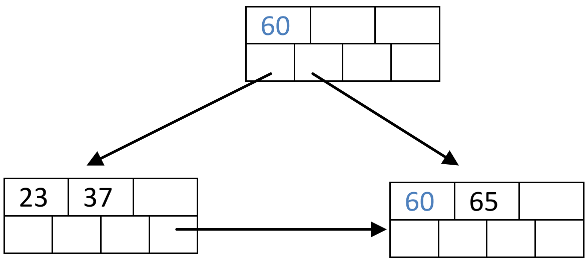


Exercise 4

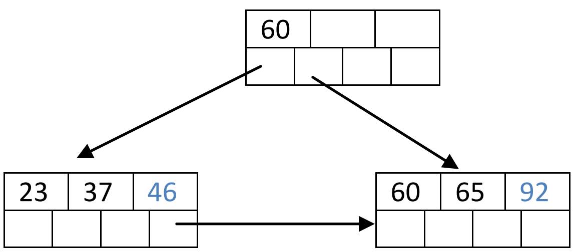
Insert 23, 65, and 37



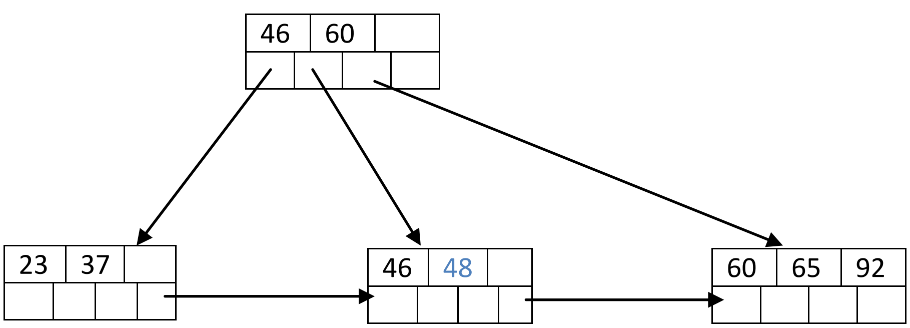
Insert 60



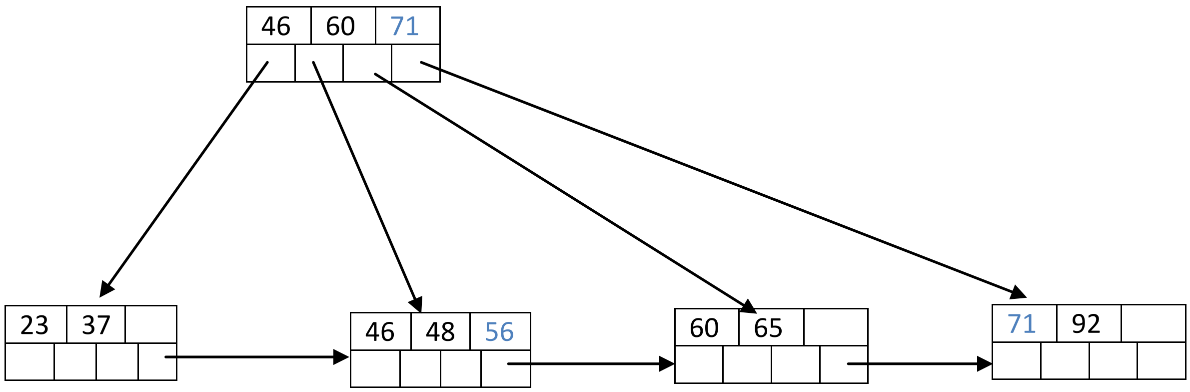
Insert 46, 92



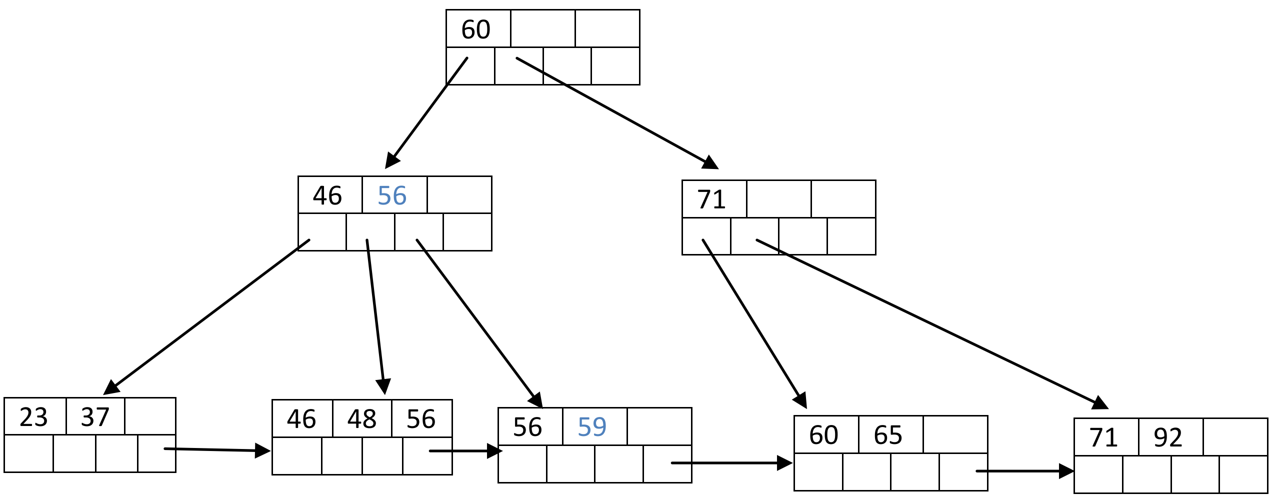
Insert 48



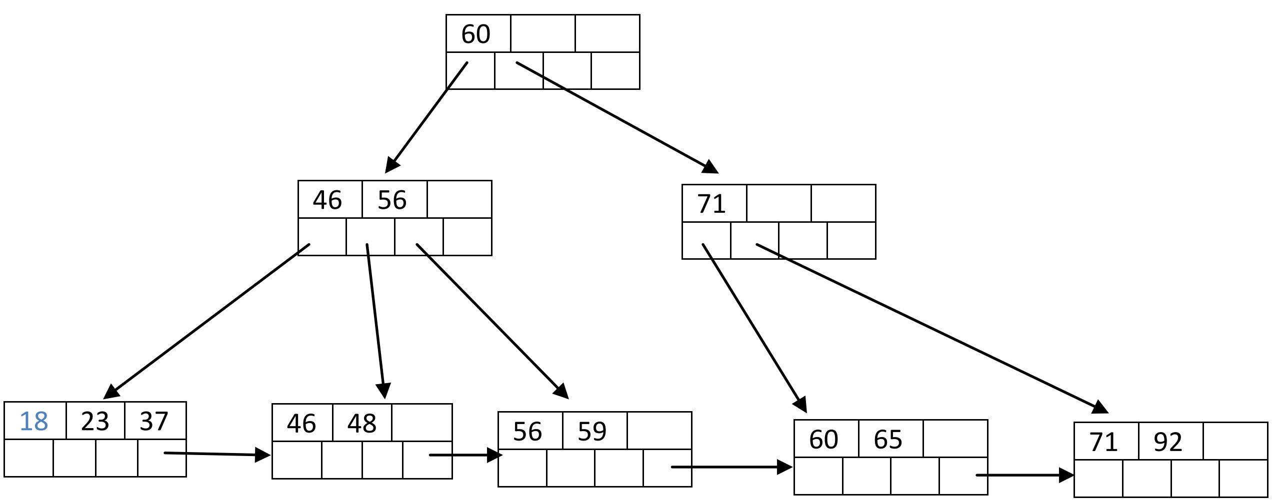
Insert 71,56



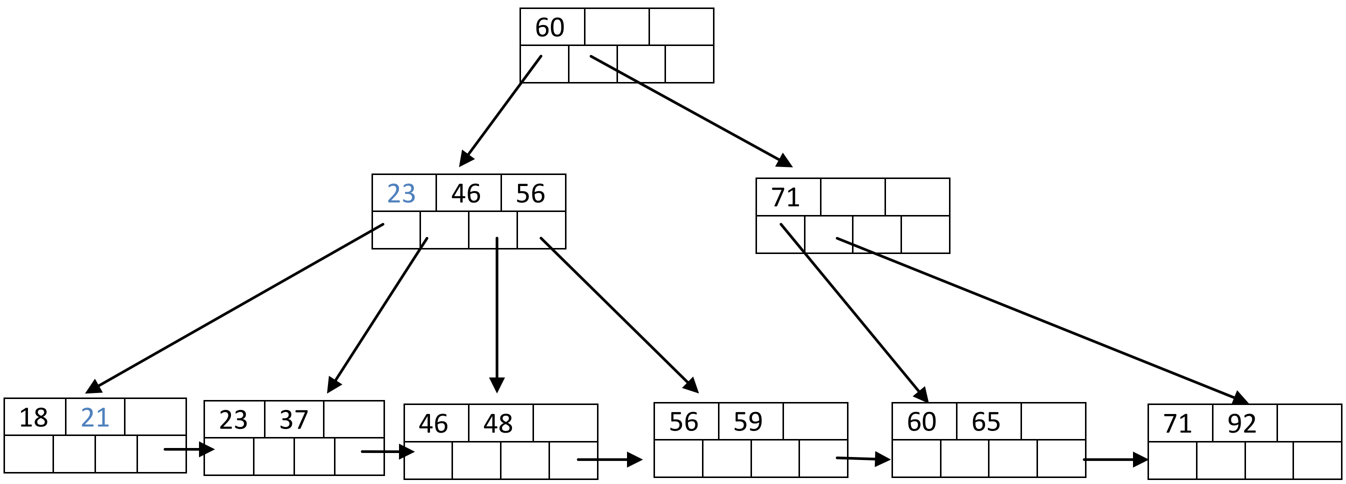
Insert 59



Insert 18

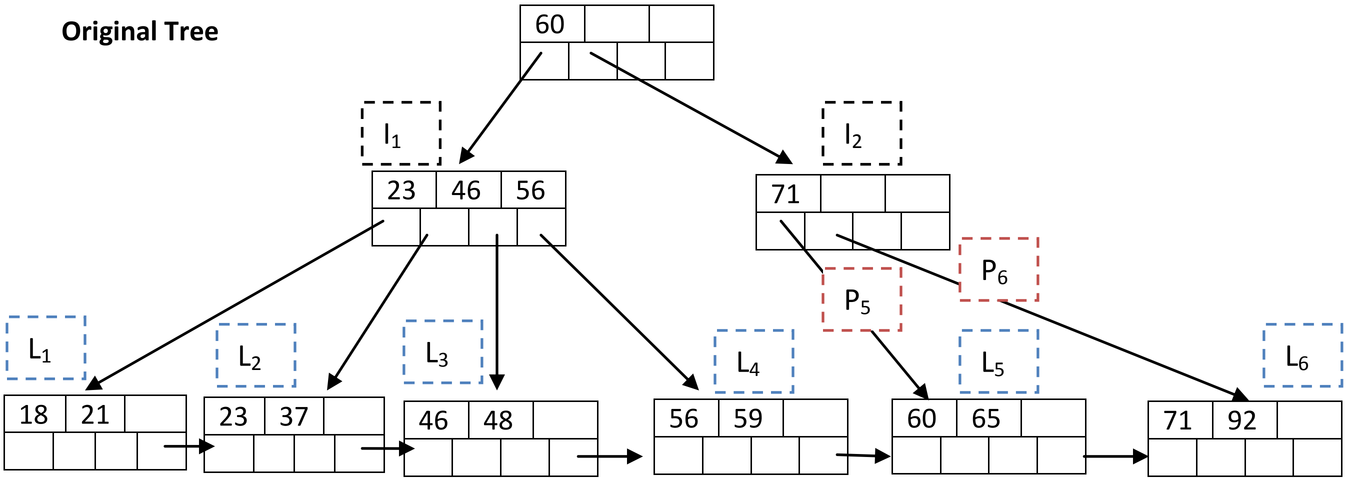


Insert 21



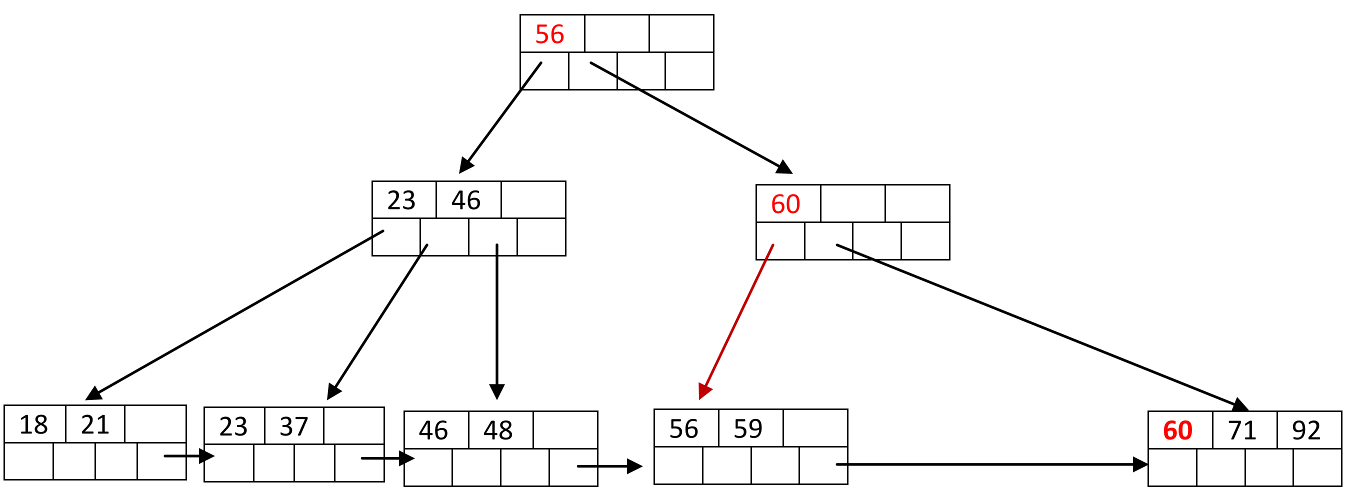
Exercise 4

Original Tree

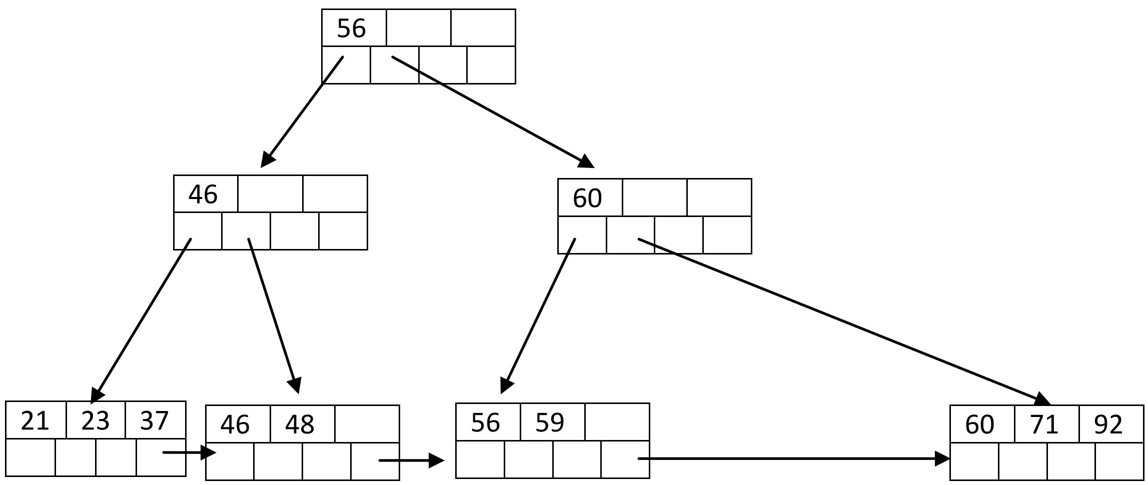


Deleting 65

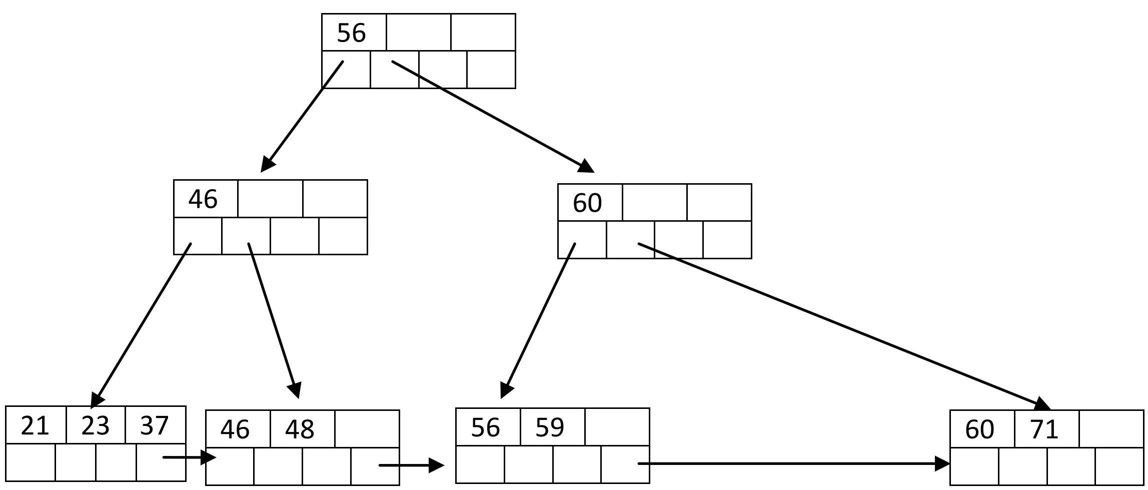
- Deleting 65 will cause L_5 to merge with L_6. L_5 block will be deleted and the associated Pointer P_5 will also be deleted. This will cause the element the pointer P_5 is linked to (i.e., the first element of I_2 (72)) to also be deleted. Just one Pointer is left (P_6)
- To keep the tree balance, I_2 will borrow a value from its sibling (i.e., borrow 56 from its sibling I_1). This is done by pushing 56 up to the root and 60 comes down to I_2. This move will cause the **pointers originally linked to 56 to be move to 60 in I_2**. (when we moved a value from one Intermediate Node to the next, the associated Pointers are also moved).
- Important note: **you can only borrow from sibling (adjacent node with same parent as the node being merged)**.



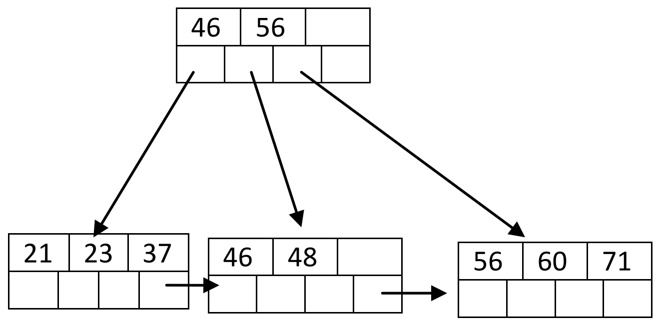
After deleting 18



After deleting 92



After deleting 59



After deleting 37