Recovery systems.

Database recovery management:
is the process of restoring the db to the most recent consistent state that existed just before the failure.

=> 3 states db recovery.

- Pre condition - in a consistent state.
- Condition - occurs some kind sys failure.
- Post condition - Restore db to the comistent state the existed before failure

Types of failure:

① Transaction failure: incorrect i/p., deadlock

② system " : OS fault, RAM failure.

③ Media " : Disk head crash, power problem.

Data base basic update stategy

Types of Failures

① Transaction failure :

(A) Logical errors : Bad i/p, Data not found, overflow, resource limit exceeded, ← coding errors

(B) system : Deadlock.

② System Crash :
H/W malfunction, Bug in, database s/w or OS.

③ Disk failure :
A disk block losses its content during data transfer operations. So copies of data are stored / back up.

UNDO & REDO

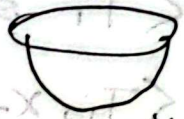
Undo : It restores the value of data items which are updated by any transaction T_i to the old values.

Redo: It sets the value of all data items updated by transaction T_i to the new value.

DO: successfully done now move forward.

T_i

R(A)
R(B)
R(C)
A = A + B
B = B + C → failure — undo
C = A + C
W(A)
commit → failure. — Redo
W(B)
W(C) → failure. — Redo
commit. failure. — Redo



There are two types of techniques.

1. log based recovery.
2. shadow paging.

Log based recovery:

The log is a sequence of log records recording all the update activities in the data base.

log contains complete records of DB activity.

various log records are:

- $\langle T_i \text{ start} \rangle$ when transaction T_i starts.
- $\langle T_i, X_j, V_1, V_2 \rangle$ Transaction T_i has performed a write operation on X_j. X_j had value V_1 before write & V_2 updated value.
- $\langle T_i \text{ commit} \rangle$ T_i has committed.
- $\langle T_i \text{ abort} \rangle$ T_i aborted.

T_1	
	$\langle T_1 \text{ start} \rangle$
R(A)	
R(B)	
A = A + B;	$\langle T_1, A, 5, 12 \rangle$
W(A)	
	$\langle T_1 \text{ commit} \rangle$

Deferred database modification:

This technique ensures transaction atomicity by recording all db modification in log, but deferring the execution of write operation until transaction partially commit:

T_0	T_1
R(A)	R(C)
A = A - 50	C = C - 100
W(A)	W(C)
R(B)	
B = B + 50	
W(B)	

A = 600
B = 900
C = 1200

T_0

< T_0 start >

< T_0, A, 550 >

< T_0, B, 950 >

< T_0 commit >

F → Redo

F undo →

T_1

< T_1 start >

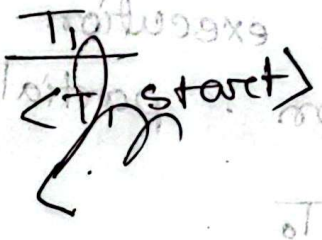
< T_1, C, 1100 > → F → Redo T_0 T_1 undo

< T_1 commit >.

→ F → T_0 T_1 Redo

Immediate database modification:

T_1	T_2
R(A)	R(C)
R(B)	R(D)
A = A + B	C = C + D
B = B + 200	W(C)
W(A)	
W(B)	



A = 200	A = 800
B = 600	B = 800
C = 900	C = 2500
D = 1600	

T_1	T_2
<T_1 start>	<T_2 start>
<T_1, A, 200, 800>	<T_2, C, 900, 2500>
<T_1, B, 600, 800>	<T_2 commit>
<T_1 commit>	

① ——— F undo ③ ———

② ———

④ ———

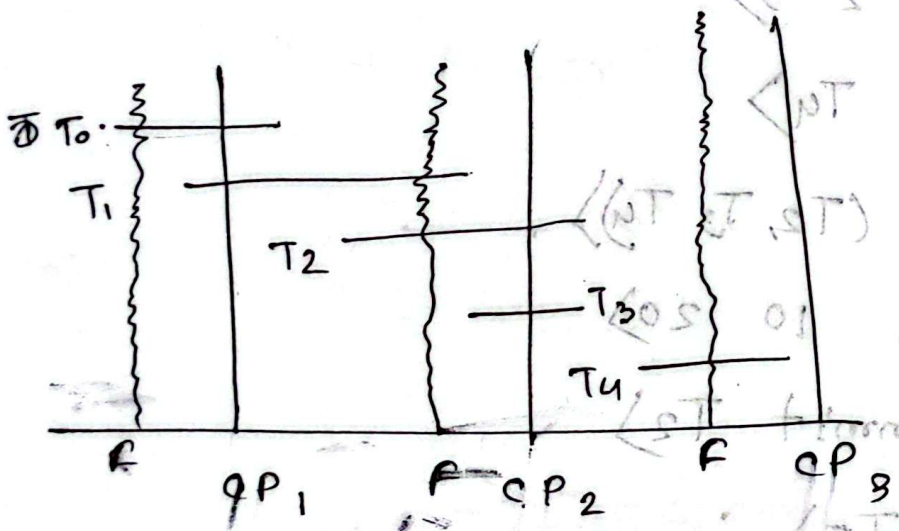
① T_1 undo $A = 200$

② T_1 Redo

③ T_2 undo T_2 + T_1 Redo . $C = 900$

④ Redo T_1 , T_2

CHeck Points!



Redo List

T_0, T_1

<checkpoint 2>

Undo List

<start T_1>

<T_1 A, 1, 2>

<start T_2>

<commit T_1>

<start T_3>

<T_3 A 2 3>

<start T_4>

<CKPT (T_2, T_3 T_4)>

<T_2 B 10 20>

<commit T_2>

<start T_5>

<T_5 D 1000 2000>

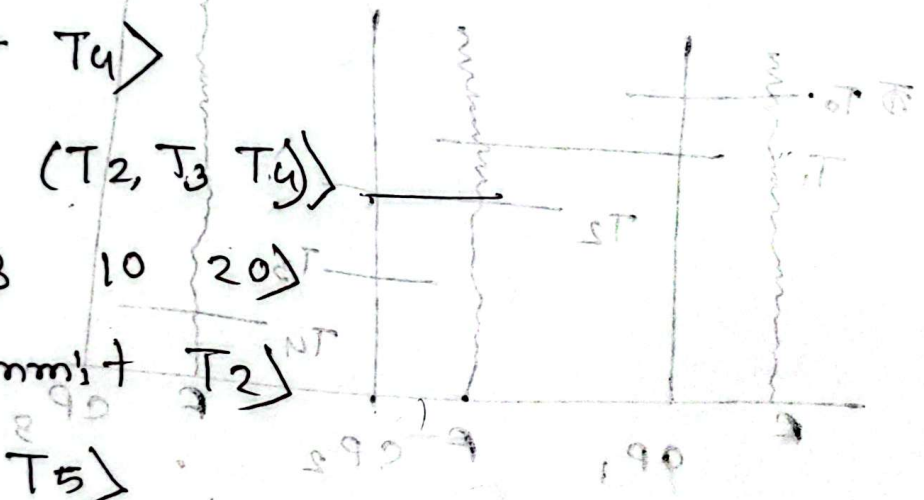
<T_4 C 100 200>

<commit T_5>

<start T_6>

<END CKPT>

<T_6 D 2000 3000>



B+ tree

(*) Order মতো দেওয়া থাকবে Max children হবে same order number.

Order, $m = 4$

max children $= 4$

min " $= \lceil \frac{m}{2} \rceil = 2$

Max keys $= m - 1 = 4 - 1 = 3$

min " $\lceil \frac{m}{2} - 1 \rceil = \lceil \frac{4}{2} - 1 \rceil = 1$

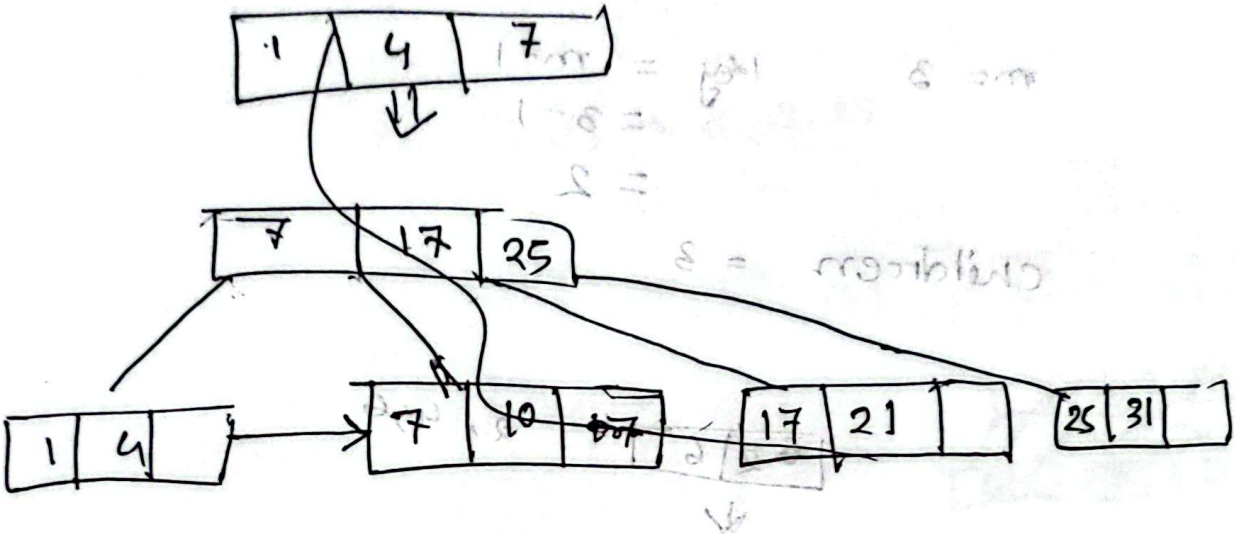
Root node এর জন্য এটা হবে না।

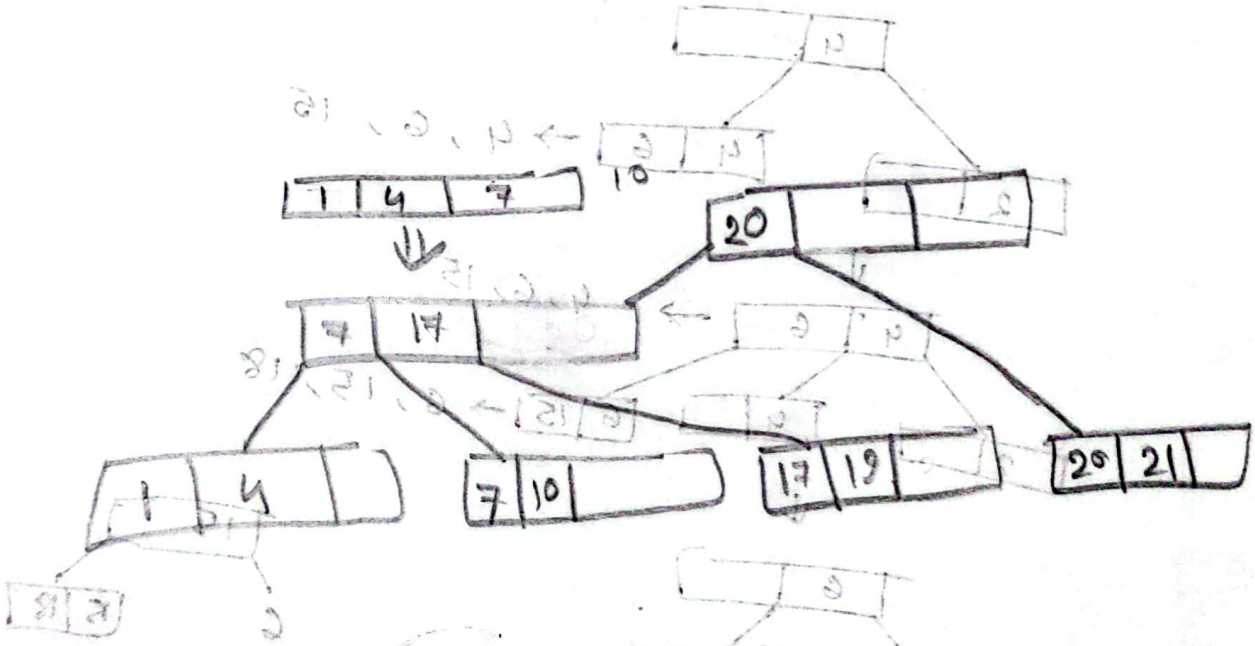
(*) B+ tree তে data গুলো শুধুই leaf node এ থাকে।

B+ tree

1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42

1, 4, 7, 10, 17, 19, 20, 21, 25, 28, 31, 42

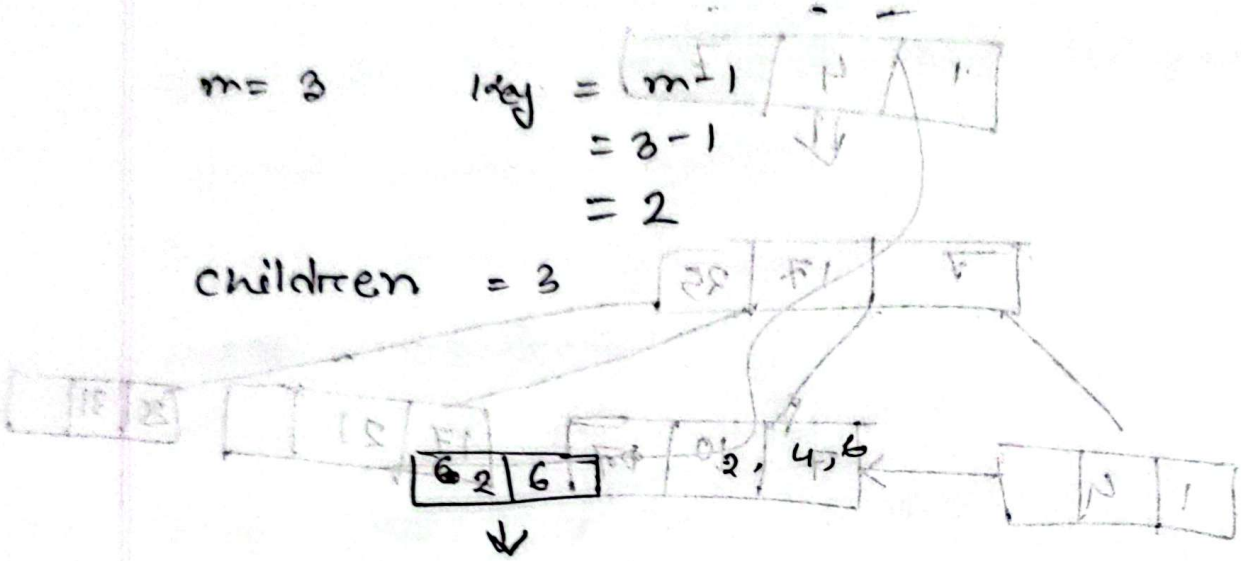

1
4
7
7
17
25
1
4
7
10
17
21
25
31

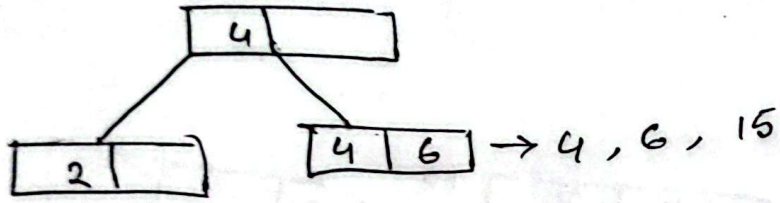

1
4
7
20
17
1
4
7
10
17
19
20
21

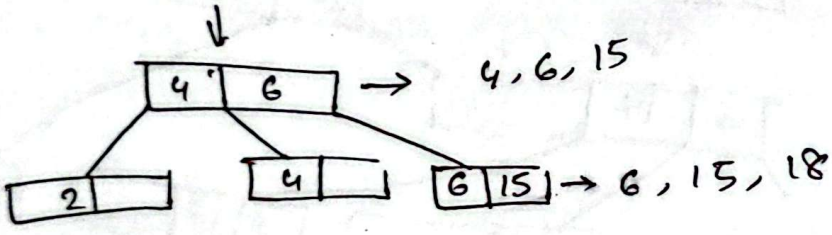
6, 2, 4, 15, 18, 8, 27, 9, 10, 13, 12, 17

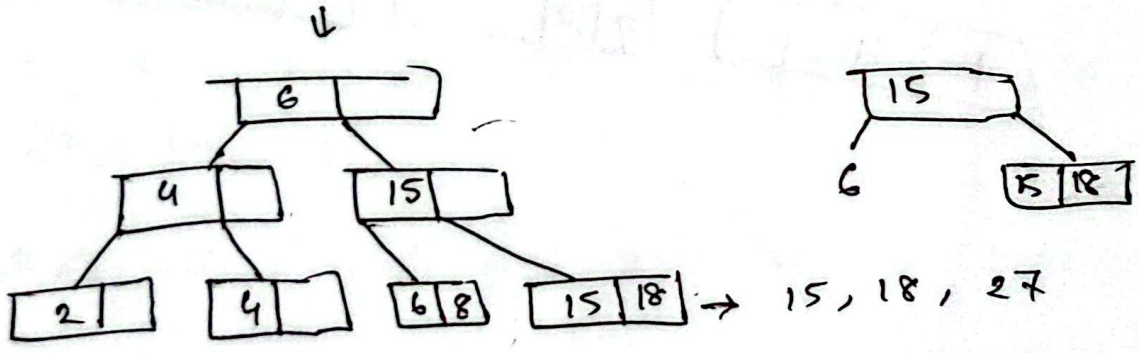
m = 3 key = m-1
= 3-1
= 2

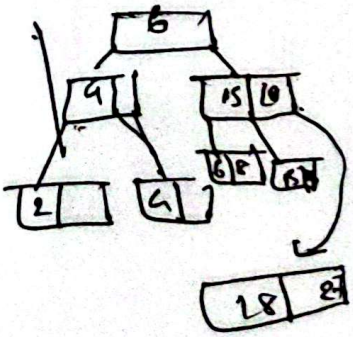
children = 3

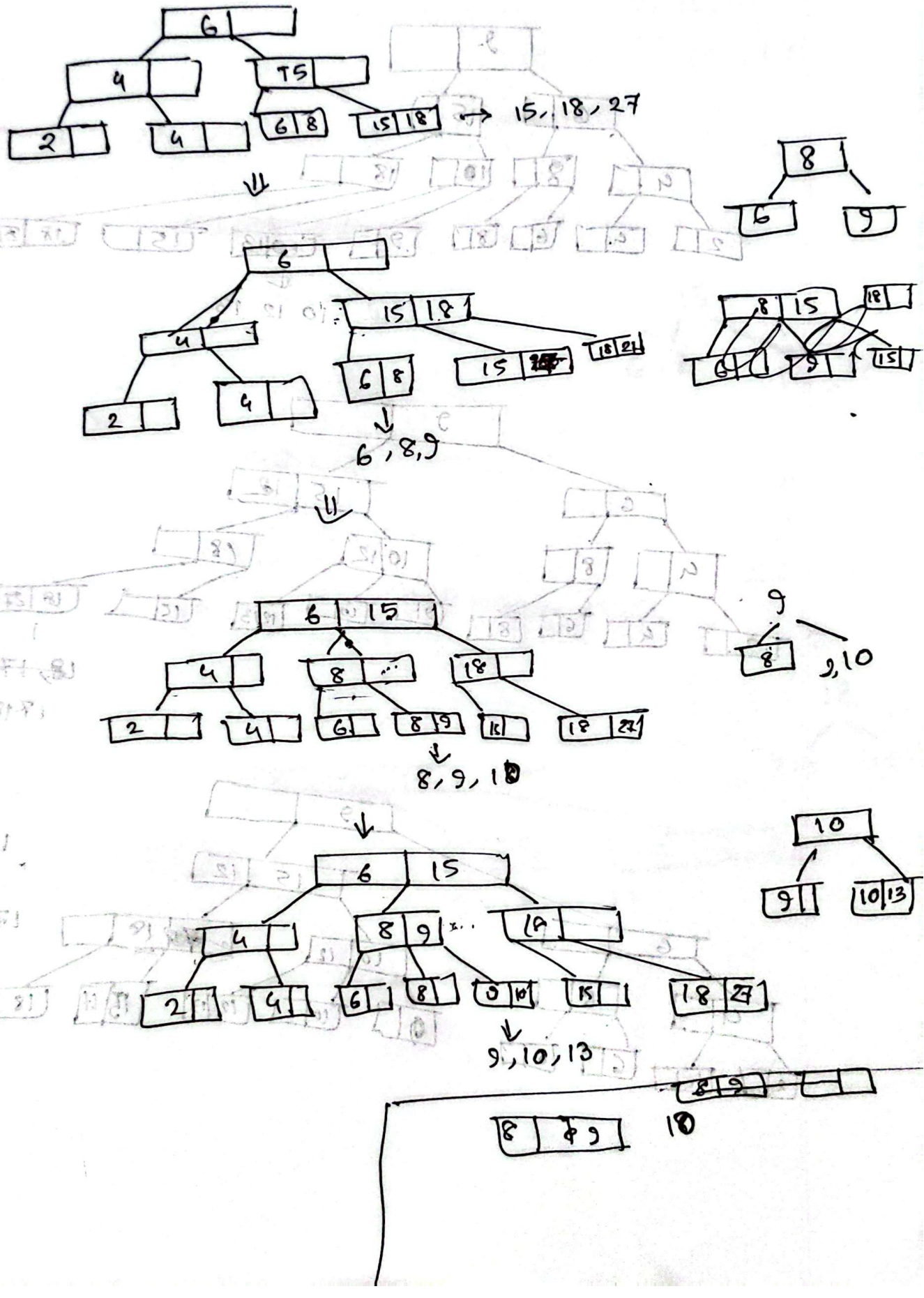


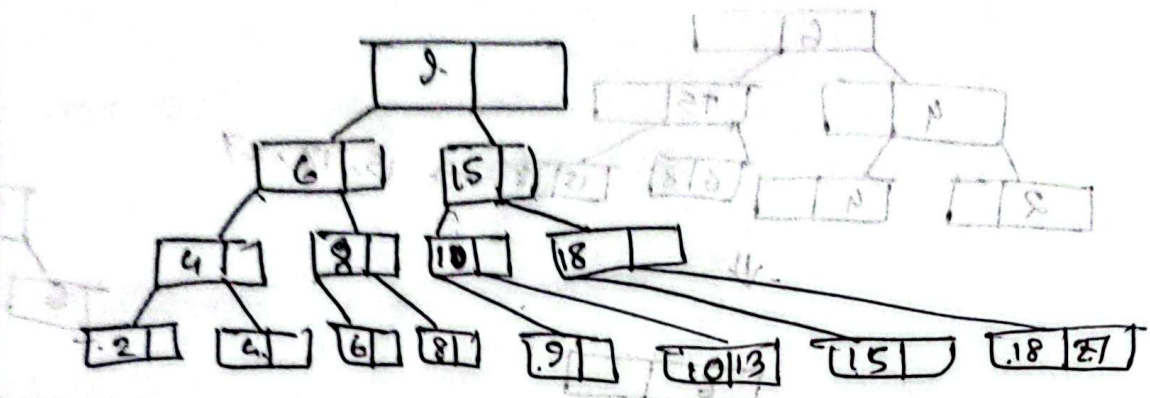




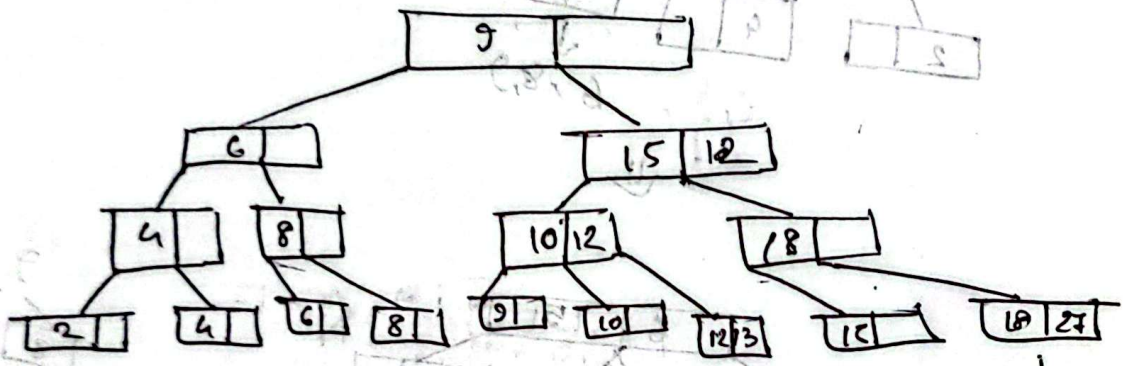





15, 18, 27
6, 8, 9
8, 9, 10
9, 10, 13

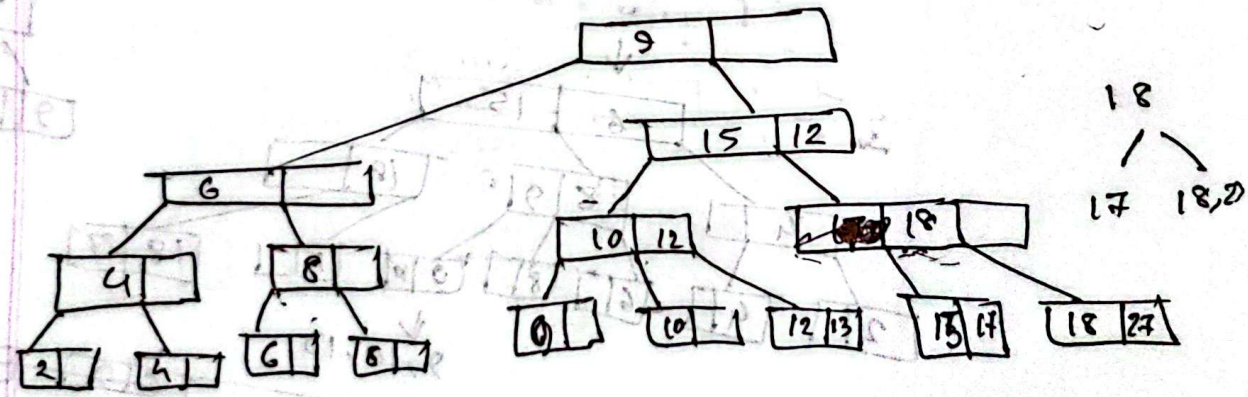


10 12 13



~~18, 17, 27~~

17 18, 27



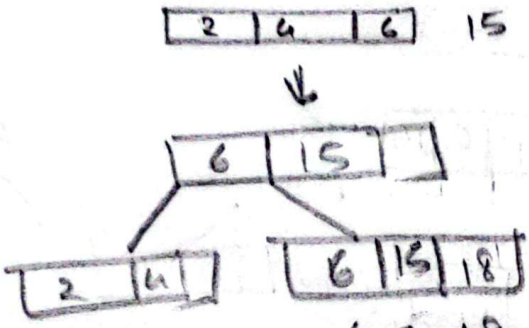
18

17 18,20

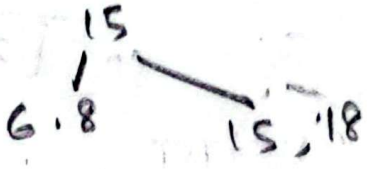
B+ tree insertion of order 4

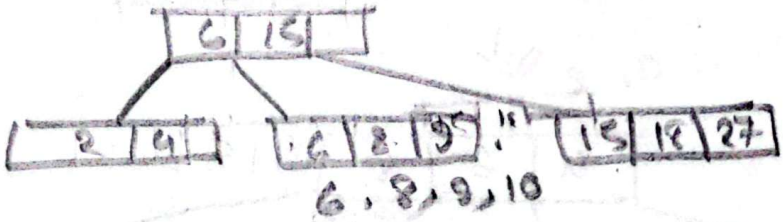
6, 2, 4, 15, 18, 8, 27, 9, 10, 13, 12, 17

m = 4 children = 4 key = 3



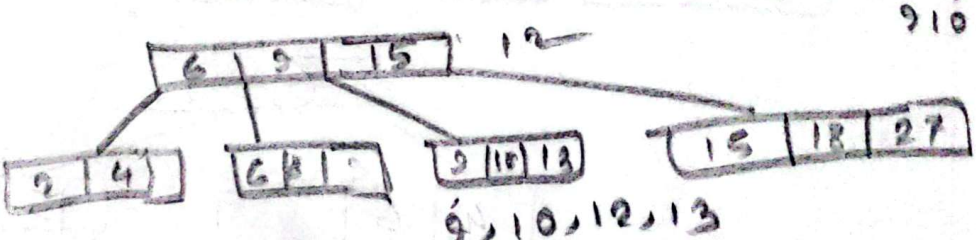
6,8 | 15, 18





6, 8, 9, 10

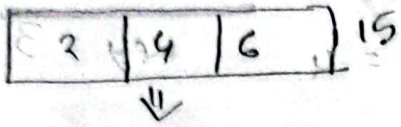
6, 9

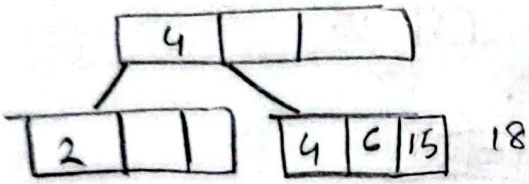


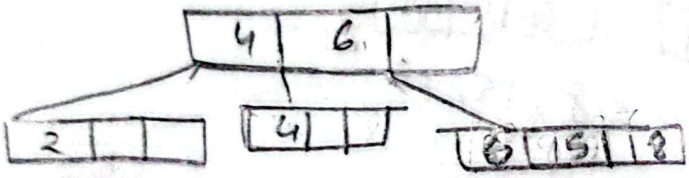
9, 10, 12, 13

11

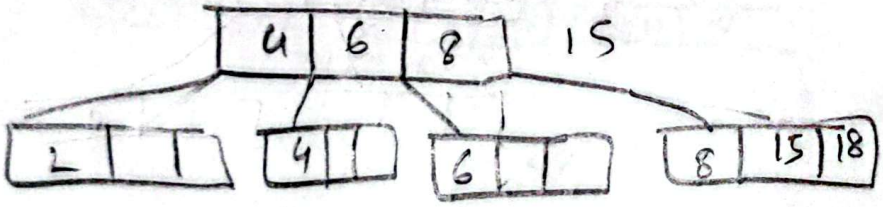
2, 6, 4, 15, 18, 8, 27, 9, 10, 13, 12, 17



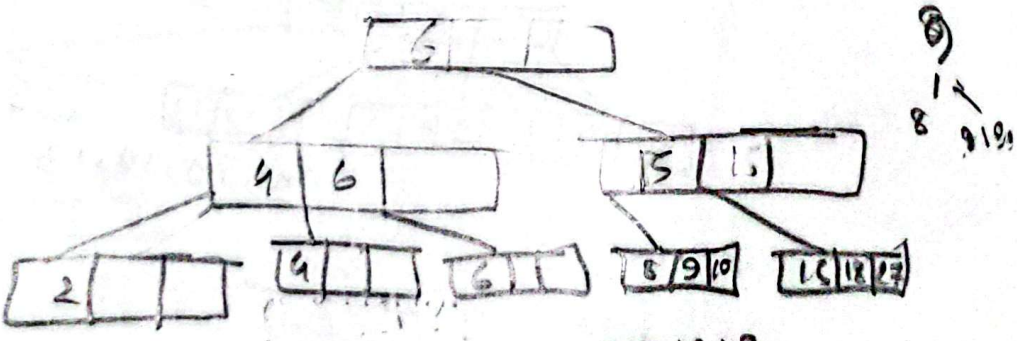




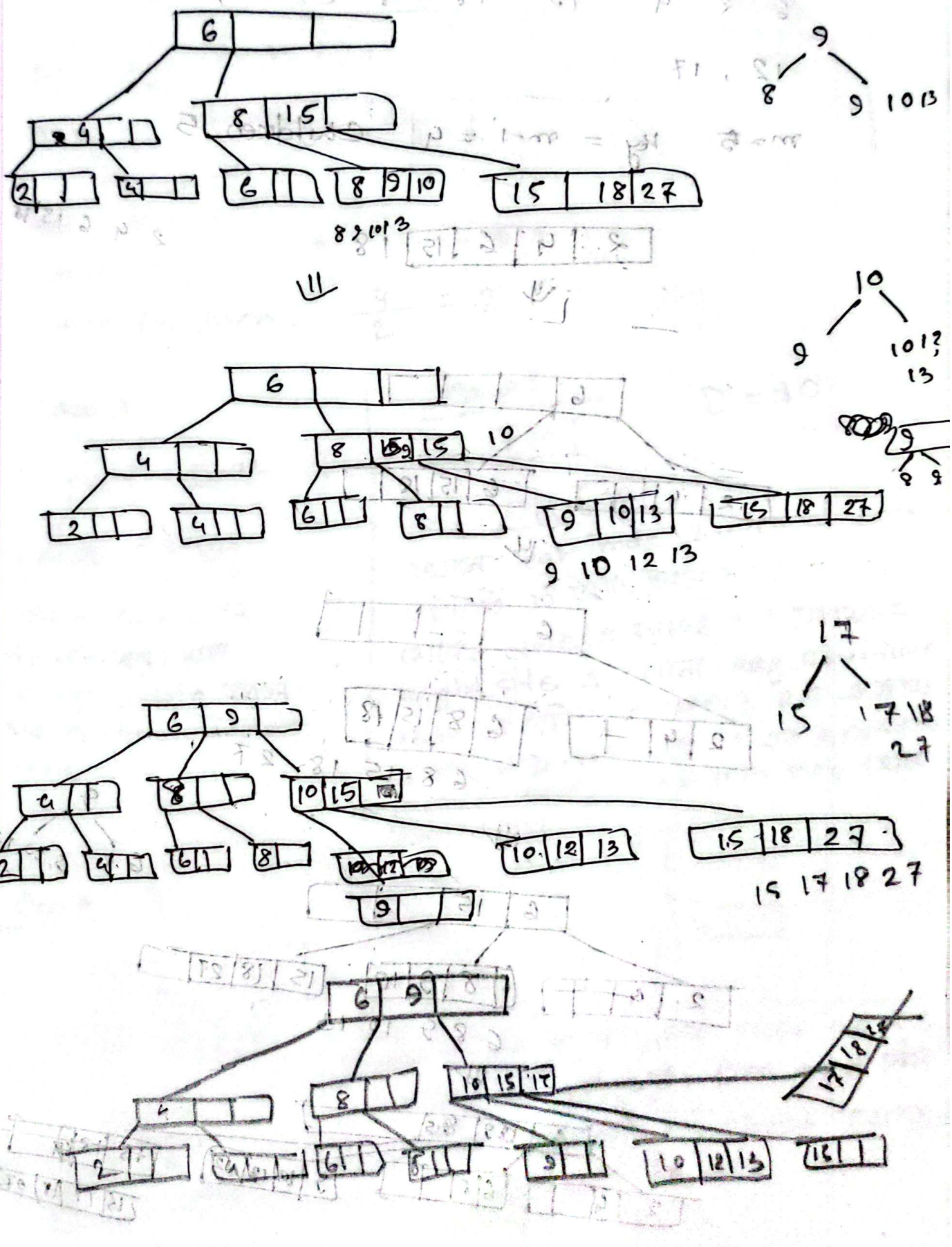
6,8, 15,18



27



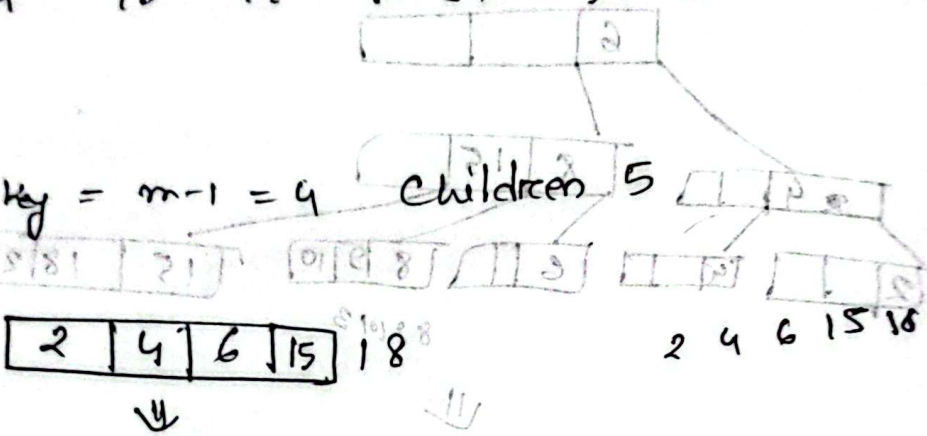
9 10 13

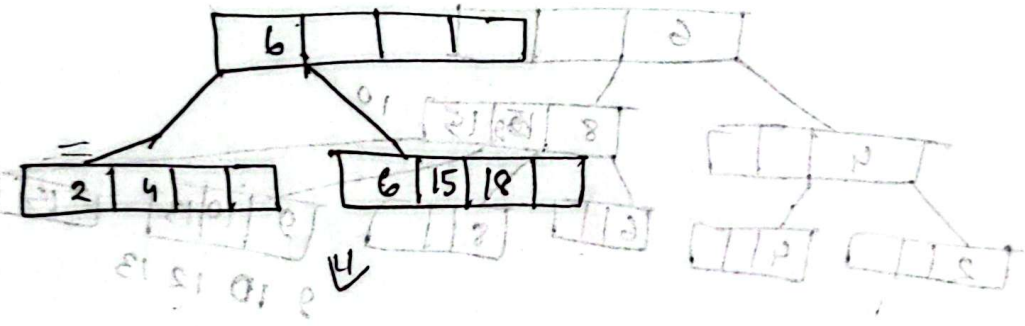

6
4
8 15
2
4
6
8 9 10
15 18 27
8 9 10 13
9
8
9 10 13
10
9
10 13
6
4
8 9 15
10
2
4
6
8
9 10 13
15 18 27
9 10 12 13
17
15
17 18 27
6 9
4
8
10 15
2
4
6
8
10 12 13
15 18 27
15 17 18 27
9
6 9
4
8
10 15 17
2
6
9
10 12 13

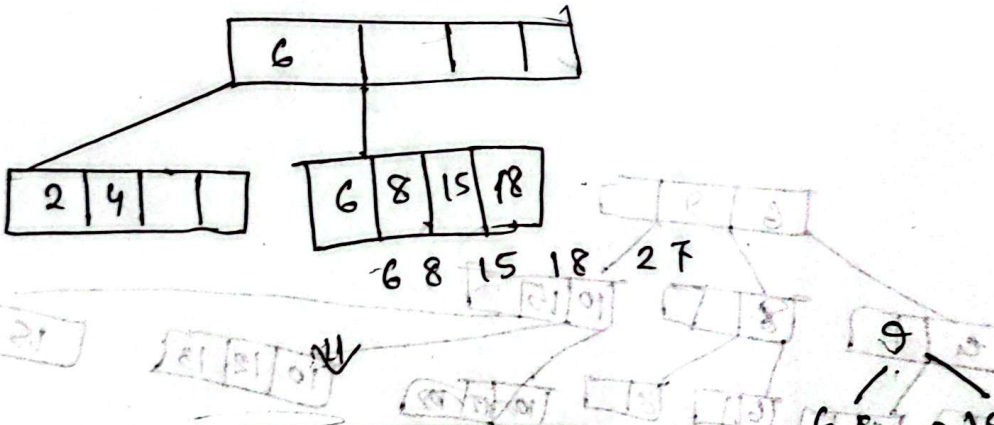
6 2 4 15 18 8 27 9 10 13

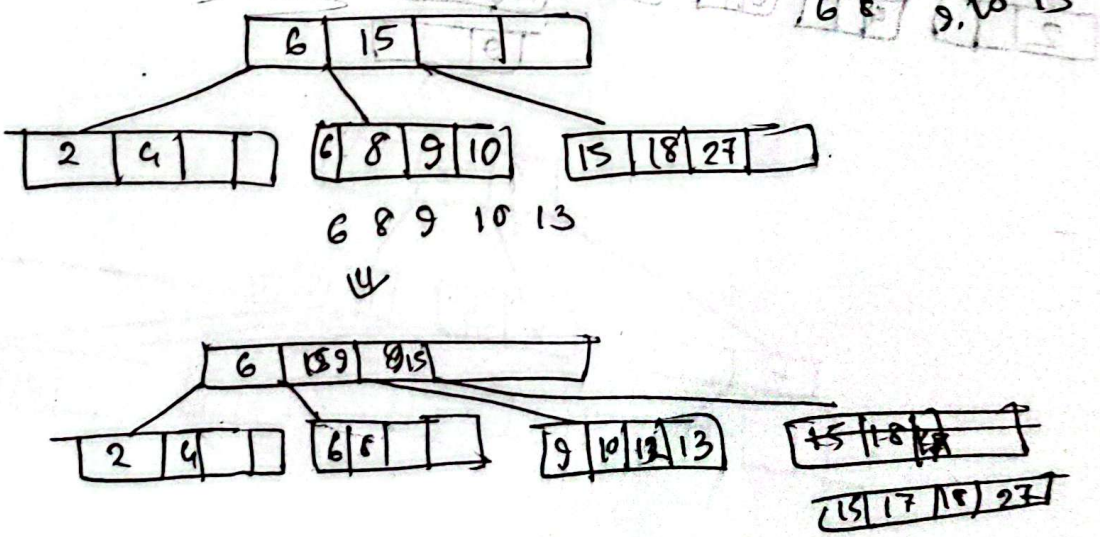
12, 17

m=5 key = m-1 = 4 children 5





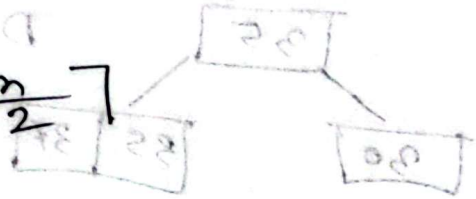




B+ tree deletion

Min key = $\lceil \frac{m}{2} \rceil - 1$

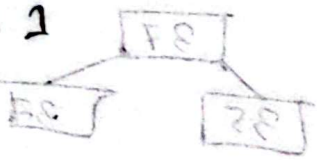
min children = $\lceil \frac{m}{2} \rceil$



m = 4.

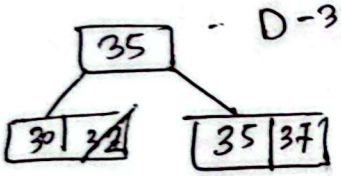
min key = $\frac{4}{2} - 1 = 1$

min children = $\frac{4}{2} = 2$



Case 1

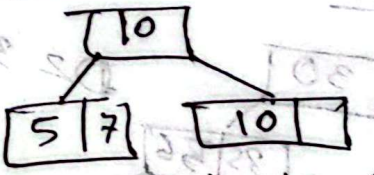
D-32



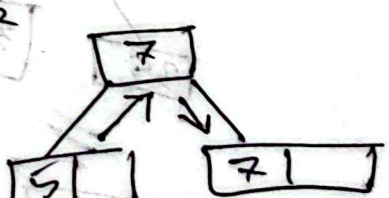
এখানে direct 32 delete করা যাবে as ওটা delete করলে সব condition maintained থাকে।

Case 2

D = 10

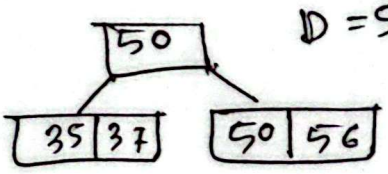


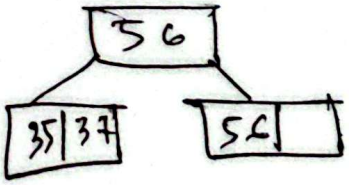
এখানে leaf node ও internal node হটাতেই 10 আছে, আমাদের 2 জায়গা থেকেই delete করতে হবে। সেক্ষেত্রে right side এ কোনো key children থাকে না তাই L.S থেকে 7 R.S এ যায় আবার ও In-node এর 10 এর জায়গায় 7 বসে। যেহেতু L.S এ min key থেকে বেশি আছে



Case - 3

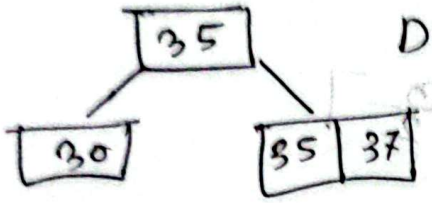
D = 50



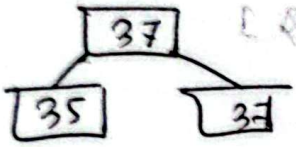


Index delete করার আগে node এর R.S এ সবচেয়ে ছোট value টা নিবো and সেটা যে উপরে পাঠাবো এবং check করবো ও min key আছে কিনা

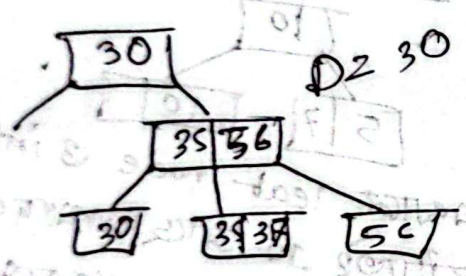
Case 4



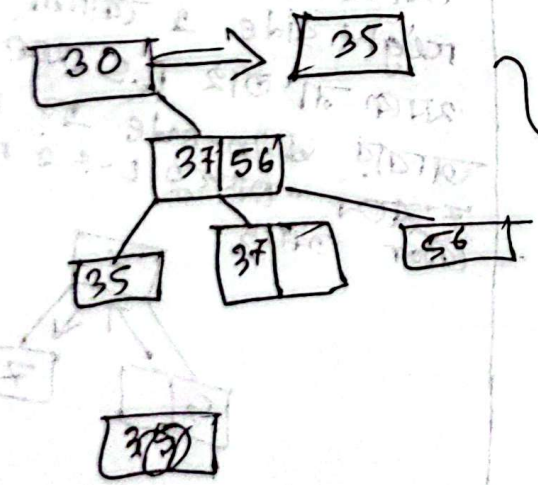
D = 30



Case 5



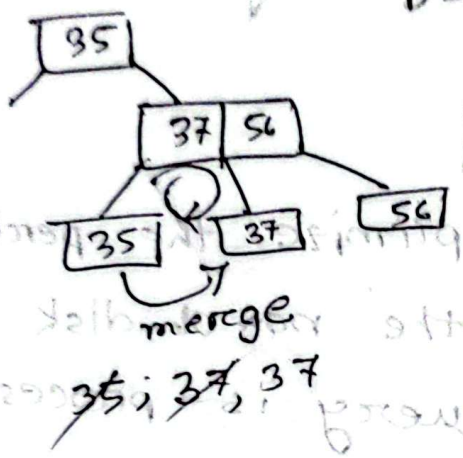
D = 30

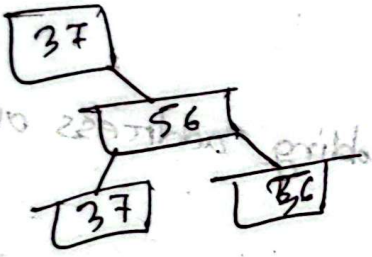


সবগুলোর min যেটা সেটা Root এ চলে যাবে।

Case 6

D = 35





Case 7

D = 4