Assignment #02

Name: Hasan Yahya
Class: BSE-6A
Roll no: 22L-7971
Subject: ADB

Question-01

r = (5000 + xxxx), As roll-no has 7971,
r = (5000 + 7971) = 12971

(1) Record size (R) = 30 + 9 + 40 + 9 + 8 + 1 + 4 + 4 + 4 + 3 + 1 = 113 bytes — Ans

(2) bfr = ⌊ B/R ⌋ = ⌊ 512/113 ⌋
= ⌊4.53⌋ = 4 records per block — Ans

b = ⌈ 12971/4 ⌉ , as b = ⌈ r/bfr ⌉
= ⌈3242.75⌉ = 3243 blocks — Ans

(3) we search on half of file blocks = 3243/2 = 1621.5

(i) Time = s + rd + (1621.5)(btt) = 20 + 10 + (1621.5 × 1)
= 1651.5 msec — Ans

(ii) Time = 1621.5 × (s + rd + btt) = 1621.5 × (20 + 10 + 1)
= 50266.5 msec — Ans

(ADB)

(4) Time = $\lceil \log_2 b \rceil \times (s + rd + btt)$

$= \lceil \log_2 3243 \rceil \times (20 + 10 + 1)$

$= \lceil 11.66 \rceil \times (31)$

$= 12 \times 31 = \boxed{372 \text{ msec}}$ — Ans

⇒ **Question-02 :** $r = (20000 + XXXX) = 20000 + 7971$

$r = 27971$ employee records.

(1) $bfr = \lfloor \frac{B}{R} \rfloor = \lfloor \frac{512}{115} \rfloor = \lfloor 4.45 \rfloor = \boxed{4 \text{ records/block}}$ — Ans

$b = \lceil \frac{27971}{4} \rceil = \lceil 6992.75 \rceil = \boxed{6993 \text{ blocks}}$ — Ans

(2) ⇒ (a) Key (SSN) — 9 bytes, Pointer size — 6 bytes

Record pointer — 7 bytes

~~So, index entry size is = 9+7 = 16 bytes.~~

Index record-size (R_i) = (9+6) = 15 bytes.

So, index blocking factor is,

$bfr_i = \lfloor \frac{512}{15} \rfloor = \boxed{34 \text{ entries/block}}$ — Ans

⇒ (b) Number of blocks at level-01,

$= \lceil \frac{6993}{34} \rceil =$

$= \lceil 205.67 \rceil = \boxed{206 \text{ blocks}}$

Ans

⇒ (c) At level 1, we have 206 block as computed in part (b).

Ans

At level 2, — one entry per level-1 block,

$\lceil \frac{206}{34} \rceil = \lceil 6.05 \rceil = \boxed{7 \text{ blocks}}$

Ans

At level-3, one entry per level-2 block,

$$\left\lceil \frac{7}{364} \right\rceil = \boxed{1 \text{ block}} \text{ — Ans}$$

Thus, the multi-level index has 3 levels (Level 1, level 2, Level 3).

$$\boxed{\text{Total levels } (x) = 3} \text{ — Ans}$$

→ (d) Total blocks required,

$= 219 \text{ (level 1)} + 7 \text{ (level 2)} + 1 \text{ (level 3)}$

$= 206 + 7 + 1 =$

$= \boxed{214 \text{ blocks}}$ — Ans

→ (e) → Access one block at top-level ie: 3.

→ Then one at level 2.

→ Then one at level 1.

→ Access the file block.

So, total block access = 3 (index level) + 1 (data block)

$= x + 1 = 3 + 1 = \boxed{4 \text{ block accesses}}$ Ans

(3) $R = 27971$

Index entry size $= 9 + 6 = 15$ bytes

→ (a) $bfi = \left\lceil \frac{512}{15} \right\rceil = \boxed{34 \text{ entries per index block}}$ Ans

→ (b) Blocks required – level 1,

$= \left\lceil \frac{27971}{34} \right\rceil = \lceil 822.67 \rceil = \boxed{823 \text{ blocks}}$

→ (c) level-01 : $\boxed{823 \text{ blocks}}$ — Ans Ans

level-02 : $\left\lceil \frac{875}{34} \right\rceil$

$= \lceil 24.20 \rceil = \boxed{25 \text{ blocks}}$ — Ans

level-03: $\lceil \frac{285}{304} \rceil$ = 1 block — Ans

So, secondary index also has 3 levels. (X = 3)

(d) Total blocks required, (Multi-level secondary index)

= 823 + 285 + 1

= 849 blocks — Ans

(e) Total block accesses

= 3 (index) + 1 (data block) = 4 accesses — Ans

→ X + 1 = 3 + 1

(4) Distinct PROGRAMCODE values = 1000

Total records = 27971

Records per PROGRAMCODE =

$\frac{27971}{1000}$ = 27.971

↳ = 28 records per PROGRAMCODE

bfr = 4

Blocks per PROGRAMCODE = $\lceil \frac{28}{4} \rceil = \lceil 7 \rceil = 7$

→ Multilevel clustering index:

→ level 1 = $\lceil \frac{1000}{32} \rceil = \lceil 31.25 \rceil = 32$ blocks

→ level 2 = $\lceil \frac{32}{32} \rceil$ = 1 block.

Thus, clustering index has 2-levels,

→ Block accesses to locate cluster

= 2 index block accesses.

→ Retrival of records, as stored in 7 contigeous blocks, so, requires 7 block accesses.

Total block accesses = 2 (index) + 7 (data)

= 9 block accesses — Ans

(ADB)

(4) ⇒ (a) index Record size $(R_i) = (V_{PROGRAMCODE} + P)$
$= (4+6) = 10$ bytes

$bfr_i = \left\lfloor \frac{512}{10} \right\rfloor = \lfloor 51.2 \rfloor =$ **51 bytes** — Ans

⇒ (b) $r1$ = distinct PROGRAMCODE values = **1000 entries**

First-level block $(b1) = \left\lceil \frac{1000}{51} \right\rceil = \lceil 19.6 \rceil$

= **20 blocks** — Ans

⇒ (c) $r2 = b1 = 20$ entries

→ Level-01 = **20 blocks**

→ Level-02 = $\left\lceil \frac{20}{51} \right\rceil$ = **1 block**

Hence, index has 2 levels,

$x = 2$ — Ans

⇒ (d) Total no of blocks,

$= b1 + b2 = 20 + 1 =$ **21 blocks** — Ans

⇒ (e) Number of block accesses to search for first block,

$= x + 1 = 2 + 1 =$ **3 accesses**

So, the 20 records are clustered in

$= \left\lceil \frac{20}{4} \right\rceil =$ **5 blocks**

Hence, total access-time needed to retrieve all records with a given PROGRAM_CODE $= x + 5 = 2 + 5 = 7$.

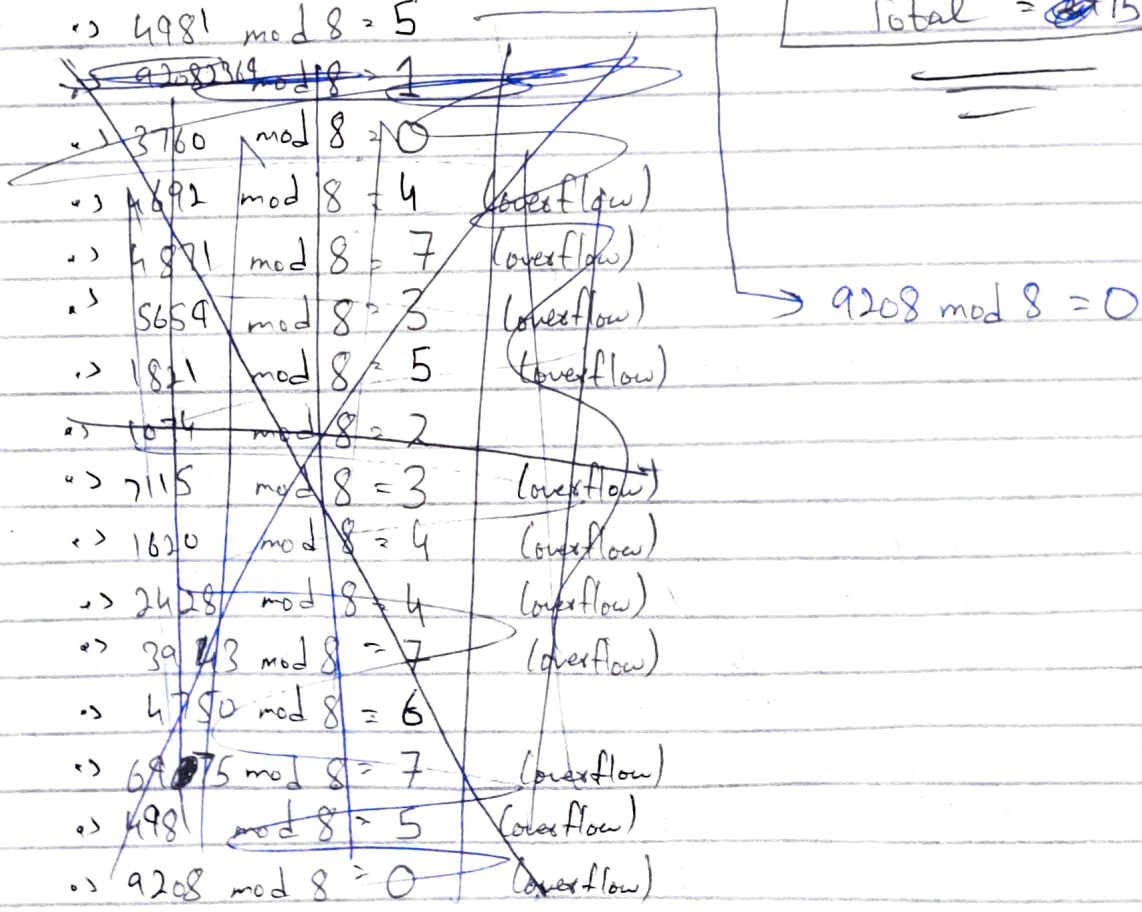
= **7 block accesses** — Ans

Question-03 : $h(k) = k \bmod 8$

(1)

- 2369 mod 8 = 1
- 3760 mod 8 = 0
- 4692 mod 8 = 4
- 4871 mod 8 = 7
- 5659 mod 8 = 3
- 1821 mod 8 = 5
- 1074 mod 8 = 2
- 7115 mod 8 = 3
- 1620 mod 8 = 4
- 2428 mod 8 = 4 (overflow)
- 3943 mod 8 = 7
- 4750 mod 8 = 6
- 6975 mod 8 = 7 (overflow)
- 4981 mod 8 = 5

Overflow = 2
Normal = 13
Total = 15



Bucket no #	index 1	index 2	overflow values
0	3760	9208	
1	2369		
2	1674		
3	5659	7115	
4	4692	1620	2428 (overflow)
5	1821	4981	
6	4750		
7	4871	3943	6975 (overflow)

(2) → Total overflow records = 2
→ Total records = 15
→ Not overflowing = 15 - 2 = 13

(3) So, average no of block accesses.

$$= 1\left(\frac{13}{15}\right) + 2\left(\frac{2}{15}\right)$$

$$= \left(\frac{13}{15}\right) + \left(\frac{4}{15}\right)$$

$$= \left(\frac{17}{15}\right)$$

$$= 1.13333\ldots$$

•> **Question 04**

Number of records (N) = 50000 + 7971 = 57971
Block size (B) = 4 KB (4069 bytes) records
SSN key size = 9 bytes
Pointer size = 6 bytes
Record size = 120 bytes

(1) For a B^+ tree of order p, the following inequality must be satisfied (for each internal tree node),

$$\boxed{(p \times P) + ((p-1) \times V_{SSN}) < B}$$

$$(p \times 6) + ((p-1) \times 9) < 4096$$
$$6p + 9p - 9 < 4096$$
$$15p - 9 < 4096$$
$$15p < 4096 + 9$$
$$15p < 4105$$
$$p < \frac{4105}{15}$$
$$p < 273.6666.....$$
$$p < 273.67$$

So, taking floor of 273.67, $\lfloor 273.67 \rfloor = 273$.

$$\boxed{p = 273}$$ — Ans

For P_{leaf},

$$\boxed{(P_{leaf} \times (V_{SSN} + P_R)) + P < B}$$

$$(P_{leaf} \times (9+6)) + 6 < 4096$$
$$15P_{leaf} + 6 < 4096$$
$$15p_{leaf} < 4090$$
$$P_{leaf} < 4090/15$$
$$P_{leaf} < 272.6666....$$
$$P_{leaf} < 272.67$$

Taking $\lfloor 272.67 \rfloor = 272$, we get,

$P_{leaf} = 272$ — Ans

(2) Average no. of key-values in a leaf-node,

$= 0.49 \times P_{leaf}$

$= 0.49 \times 272$

$= 133.28$

or we round-up to 134 for convenience,

So we get 134 key-values.

$$\text{Leaf-blocks} = \left\lceil \frac{57971}{134} \right\rceil = \lceil 432.619 \rceil$$

= 433 leaf blocks — Ans

(3) Average fan-out for internal nodes is,

$$f_o = \lceil 0.49 \times p \rceil = \lceil 0.49 \times 273 \rceil = \lceil 133.77 \rceil$$

$= 134.$

→ Level-01 = 433 nodes

→ Level-02 = $\left\lceil \frac{433}{134} \right\rceil = \lceil 3.23 \rceil = 4$ nodes.

→ Level-03 = $\left\lceil \frac{4}{134} \right\rceil = \lceil 0.029 \rceil = 1$ node.

Thus B+- tree has 3 levels (counting the leaf-level). To verify,

$$x = \left\lceil \log_{134} 433 \right\rceil + 1 = \lceil 1.239 \rceil + 1 = 2 + 1$$

$x = 3$ — Ans — Hence Verified

(4) Total no of blocks required,

$= b1 + b2 + b3$

$= 433 + 4 + 1 =$ **438 blocks** — Ans

(5) Total no of block accesses needed,

$= x + 1 = 3 + 1 =$ **4 block accesses** — Ans

→ **Question-05:**

(1) As, $p \times P + ((p-1) \times (V_{SSN} + P_R)) \leq B$ must be satisfied for B-tree,

$(p \times 6) + ((p-1) \times (9+6)) \leq 4096$

$6p + ((p-1) \times 15) \leq 4096$

$6p + 15p - 15 \leq 4096$

$21p \leq 4096 + 15$

$p \leq 4111/21$

$p \leq 195.76....$

So, donot have a **$p = 195$** for B-trees. B-trees separate **P_{leaf}** value.

$p = 195$ — Ans

(2) Average no of key values,

$= 0.49 \times 195$

$=$ **$95.55 \cong 96.$**

we round to 96 for convinience.

no of leaf-level blocks (b_1) = $\lceil \frac{57971}{96} \rceil = \lceil 603.86 \rceil$

= **604 blocks** — Ans

(3)

$f_o = \lceil 0.49 \times 195 \rceil = 96$.

→ Level 1 = 604 blocks.
→ Level 2 = $\lceil 604/96 \rceil = \lceil 6.29 \rceil = 7$ blocks.
→ Level 3 = $\lceil 7/96 \rceil = \lceil 0.07 \rceil = 1$ block.

OR

$x = \lceil \log_{96}(604) \rceil + 1 = \lceil 1.402 \rceil + 1$

$= 2 + 1 =$ **3 levels** — Ans

(4) No of blocks required by B-tree,

= 604 + 7 + 1

= **612 blocks** — Ans

(5) No of block accesses,

= x + 1 = 3 + 1 = **4 accesses** — Ans

Comparison: B vs B+ Tree

The B-tree has lower capacity per node vs B+ tree, so it requires more leaf-blocks (604 block) vs that of B+ tree (433 blocks). However searching is slightly quicker in B-tree than in B+-tree.

-> **Question - 06 :**

237, 589, 124, 763, 901, 456, 378, 812, 645,
293, 715, 847, 362, 508, 194

① Decimal ——(using value % 8)——> Hashed ——> Binary

237	→	5	→ 101
589	→	5	→ 101
124	→	4	→ 100
763	→	3	→ 011
901	→	5	→ 101
456	→	0	→ 000
378	→	2	→ 010
812	→	4	→ 100
645	→	5	→ 101
293	→	5	→ 101
715	→	3	→ 011
847	→	7	→ 111
362	→	2	→ 010
508	→	4	→ 100
194	→	2	→ 010

→ Extendible Hashing:

⑤

237
589
124

d' = 0 (2^0 = 1)
d = 0

763 (overflow)

(2^1 = 2)

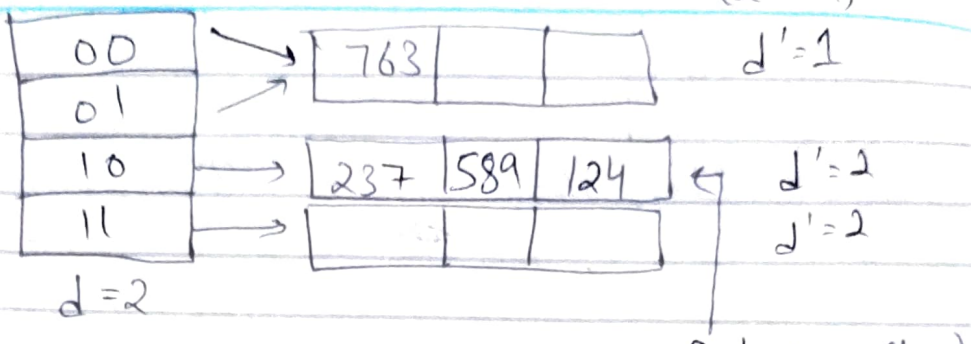
⑤

0 →	763			d' = 1
1 →	237	589	124	d' = 1

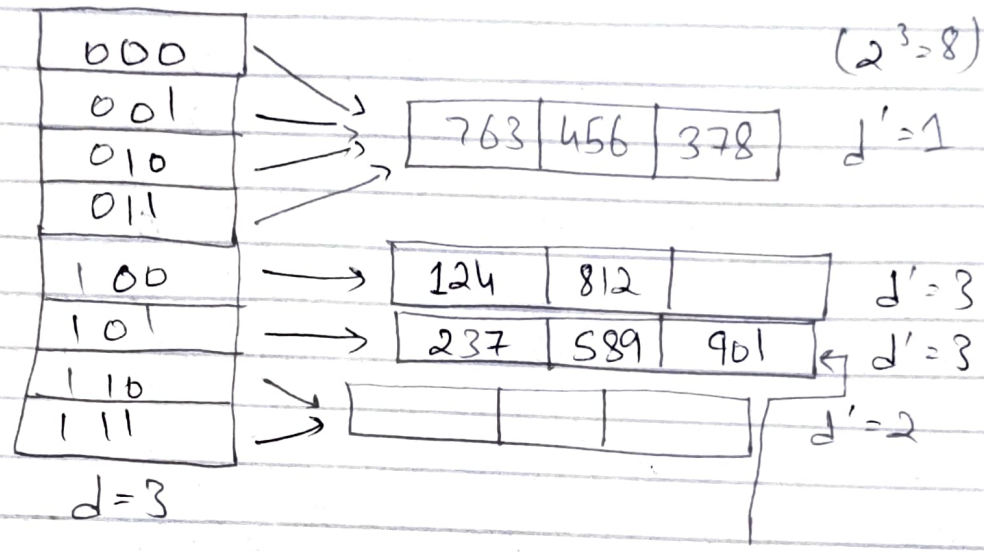
901 (overflow)

d = 1

(5)



(5)



which is not possible, so doubling the hash function,

$$h(k) = k \bmod 16$$

Decimal	→ (using value % 16) Hashed	→ Binary
237	→ 13	→ 1101
589	→ 13	→ 1101
124	→ 12	→ 1100
763	→ 11	→ 1011
901	→ 5	→ 0101
456	→ 8	→ 1000
378	→ 10	→ 1010
812	→ 12	→ 1100

645 → 5 → 0101
293 → 5 → 0101
715 → 11 → 1011
847 → 15 → 1111
362 → 10 → 1010
508 → 12 → 1100
194 → 2 → 0010

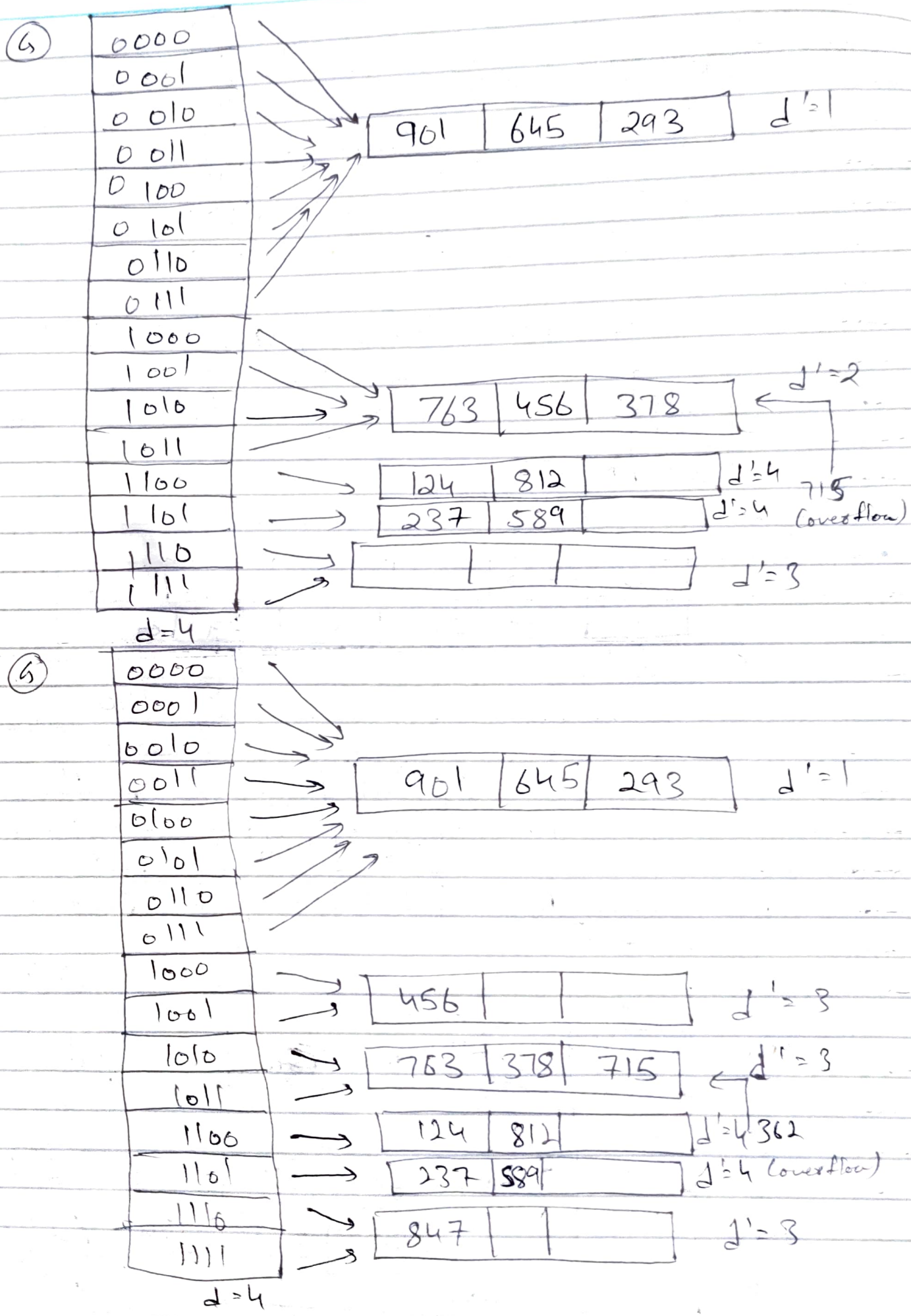
using extendable hashing again

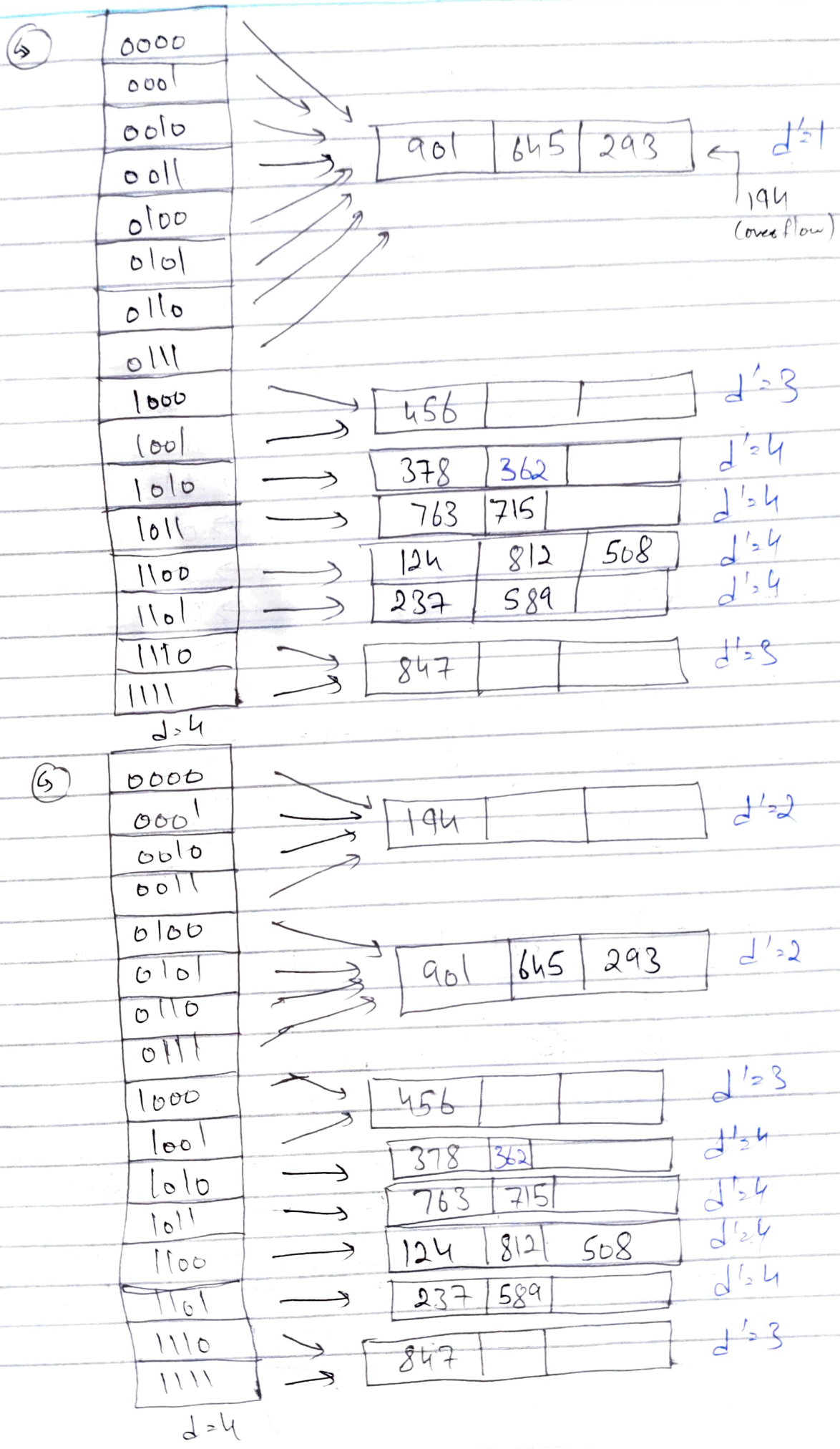
(4) 237 | 589 | 124 ← 763 (overflow)
d' = 0 (2^0 = 1)
d = 0

(5)
0 → [ ] [ ] [ ] d' = 1
1 → 237 | 589 | 124 d' = 1
d = 1
763 (overflow)

(6)
00 → 901 d' = 1
01
10 → 763 | 456 | 378 d' = 2
11 → 237 | 589 | 124 d' = 2
d = 2
812 (overflow)

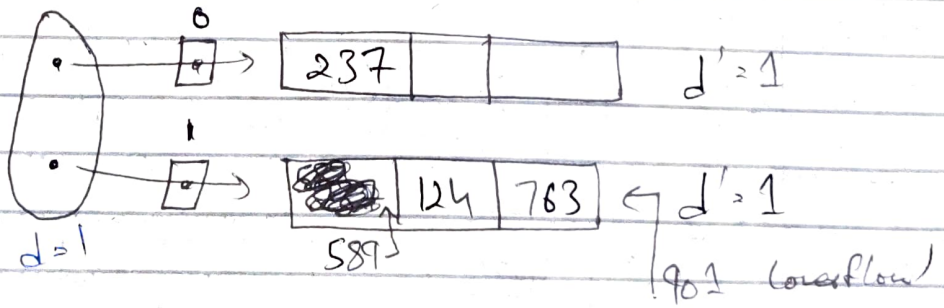
(5)
000 → 901 d' = 1
001
010 → 763 | 456 | 378 d' = 2
011
100 → 237 | 589 | 124 d' = 3
101
110 → [ ] [ ] [ ] d' = 3
111
d = 3
812 (overflow)

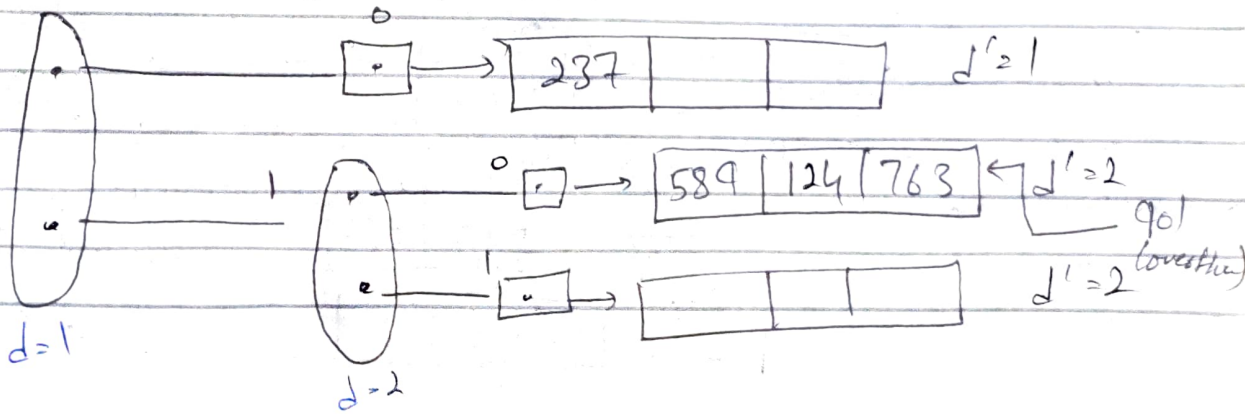

(4)
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
d=4
901
645
293
d'=1
763
456
378
d'=2
124
812
d'=4
237
589
d'=4
715
(overflow)
d'=3
(5)
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
d=4
901
645
293
d'=1
456
d'=3
763
378
715
d'=3
124
812
d'=4
362
237
589
d'=4 (overflow)
847
d'=3


(4)
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
d=4
901
645
293
194
(overflow)
d'=1
456
d'=3
378
362
d'=4
763
715
d'=4
124
812
508
d'=4
237
589
d'=4
847
d'=3
(5)
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
d=4
194
d'=2
901
645
293
d'=2
456
d'=3
378
362
d'=4
763
715
d'=4
124
812
508
d'=4
237
589
d'=4
847
d'=3

(2) Dynamic Hashing

Decimal	(value mod 10) mod 3	Hashed	Binary
237	→	1	→ 01
589	→	2	→ 10
124	→	2	→ 10
763	→	2	→ 10
901	→	2	→ 10
456	→	1	→ 01
378	→	1	→ 01
812	→	0	→ 00
645	→	1	→ 01
293	→	0	→ 00
715	→	2	→ 10
847	→	2	→ 10
362	→	0	→ 00
508	→	2	→ 10
194	→	0	→ 00



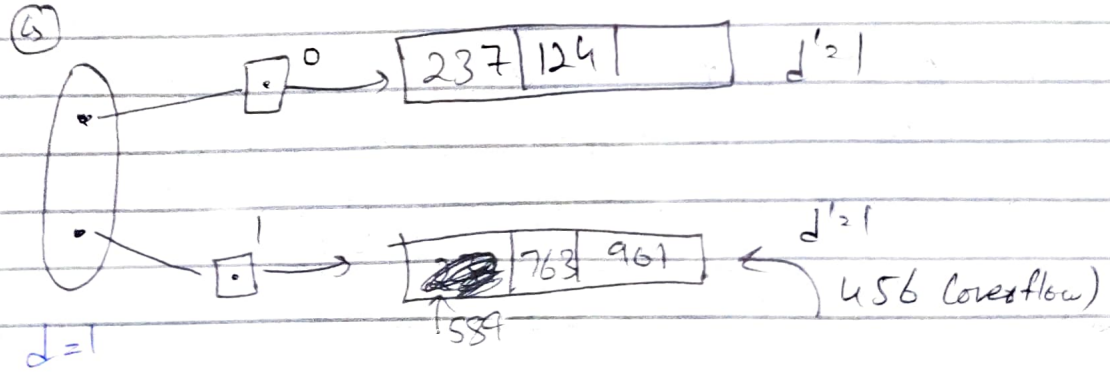


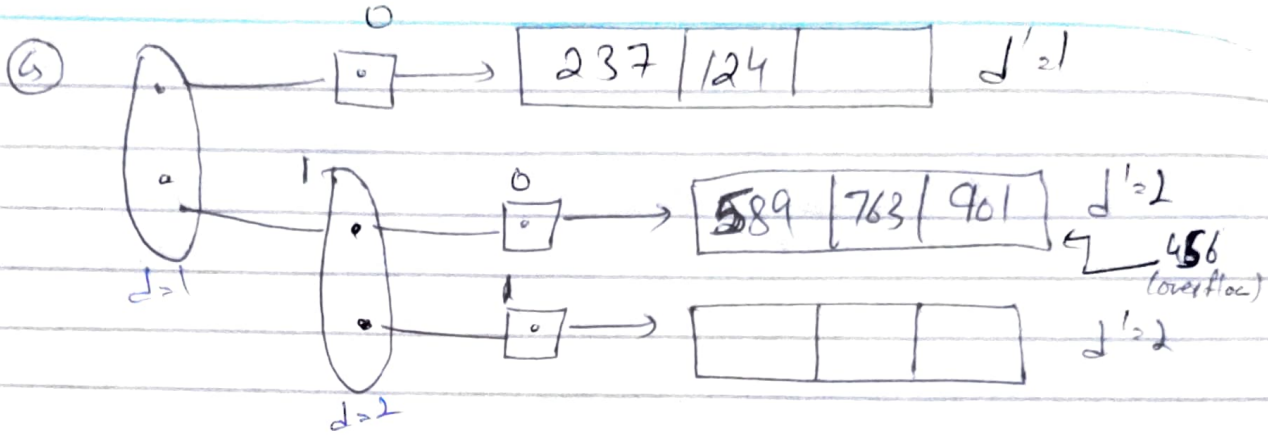
dividing again creates a problem that the binary representation is in 2 bit values & doubling will create 3 bit values that cannot be compared, so we double the hash-function.

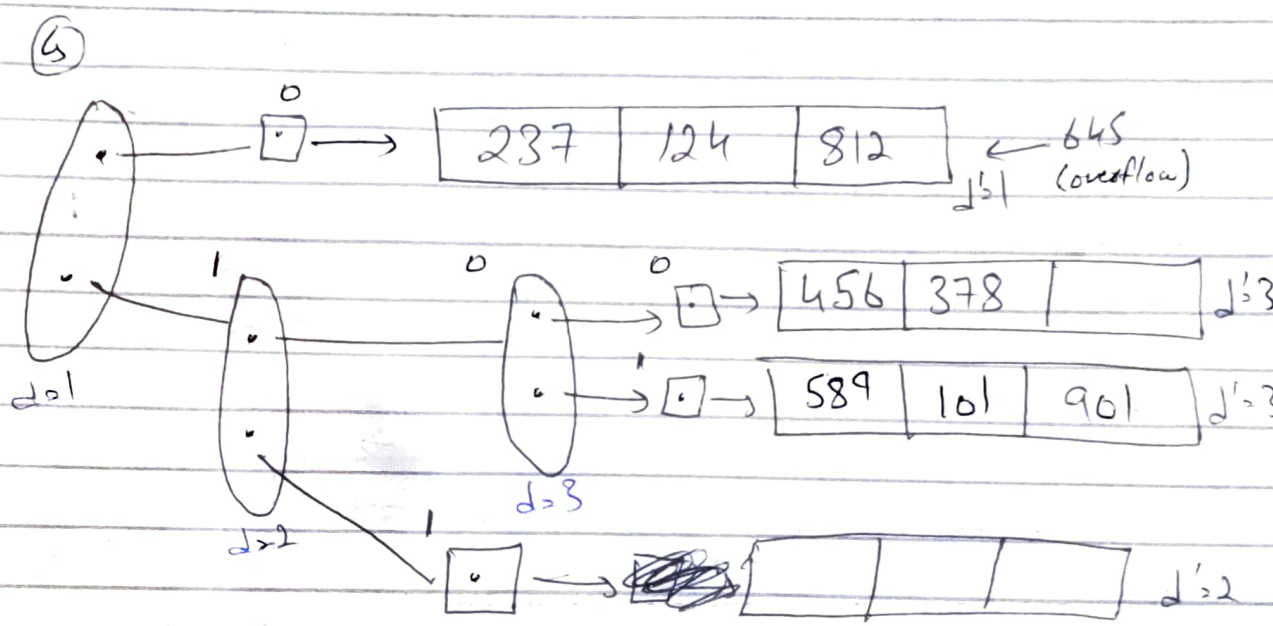
$$h(k) = (k+10) \% 6$$

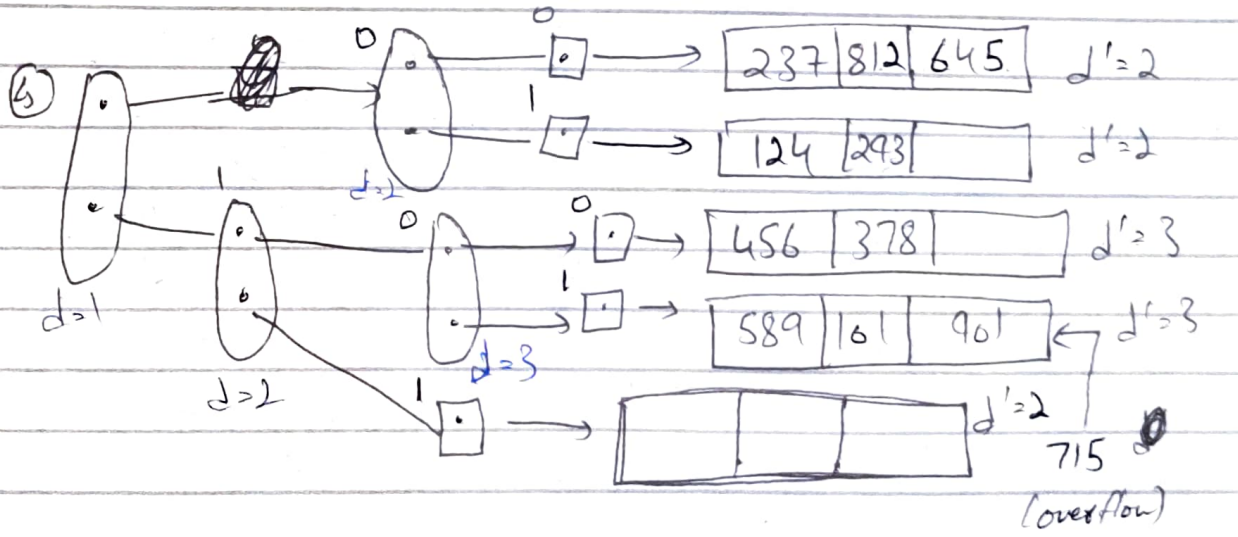
Decimal	→ (rel+10)%6	Hashed	→	Binary
237	→	1	→	001
589	→	5	→	101
124	→	2	→	010
763	→	5	→	101
901	→	5	→	101
456	→	4	→	100
378	→	4	→	100
812	→	0	→	000
645	→	1	→	001
293	→	3	→	011
715	→	5	→	101
847	→	5	→	101
362	→	0	→	000
508	→	2	→	010
194	→	0	→	000

(4)





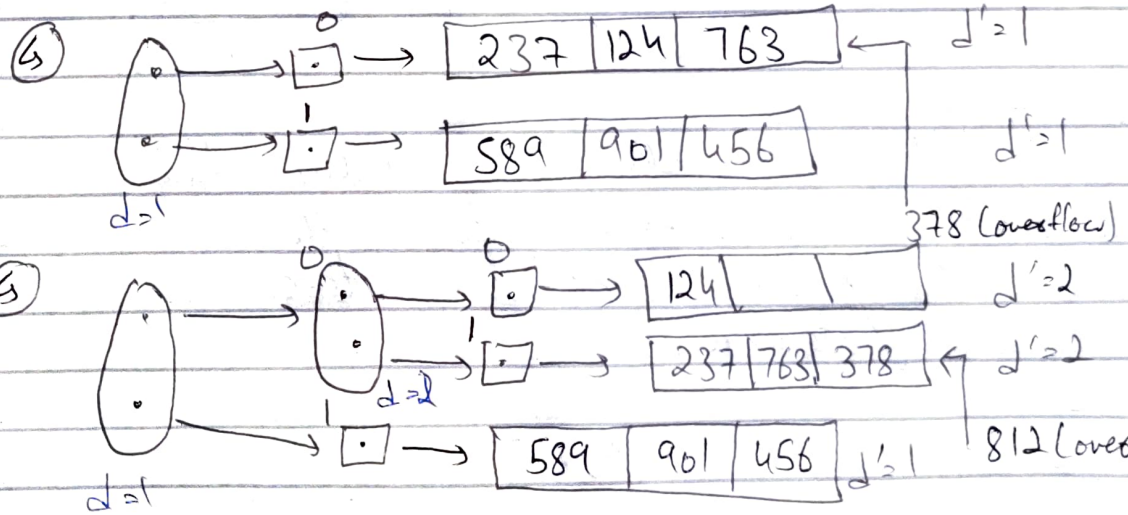


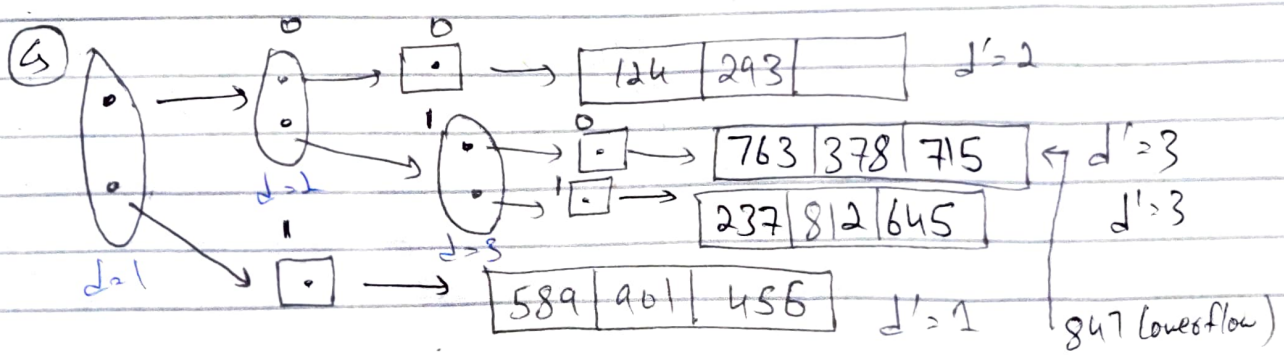


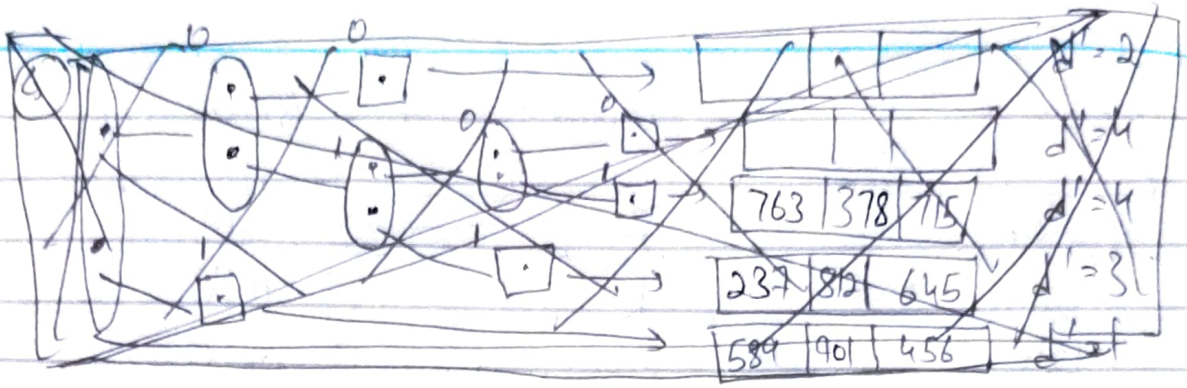
We can't split, so we double the hash-function again.

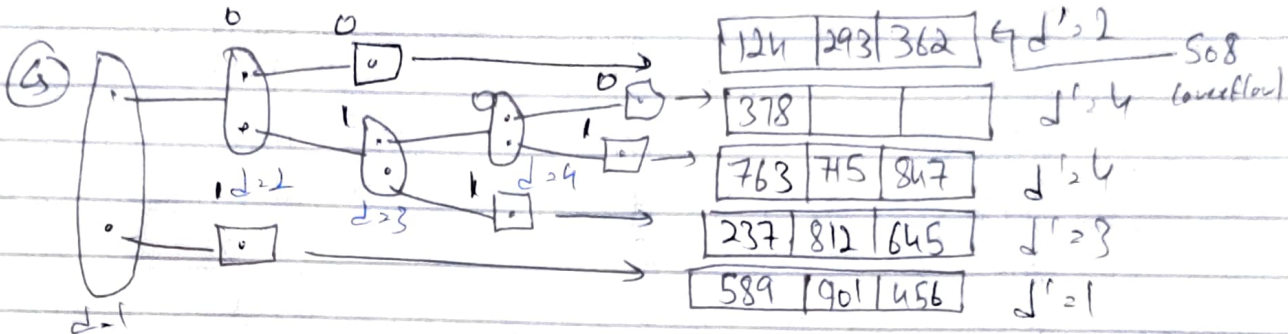
$$h(k) = (k+16) \bmod 12$$

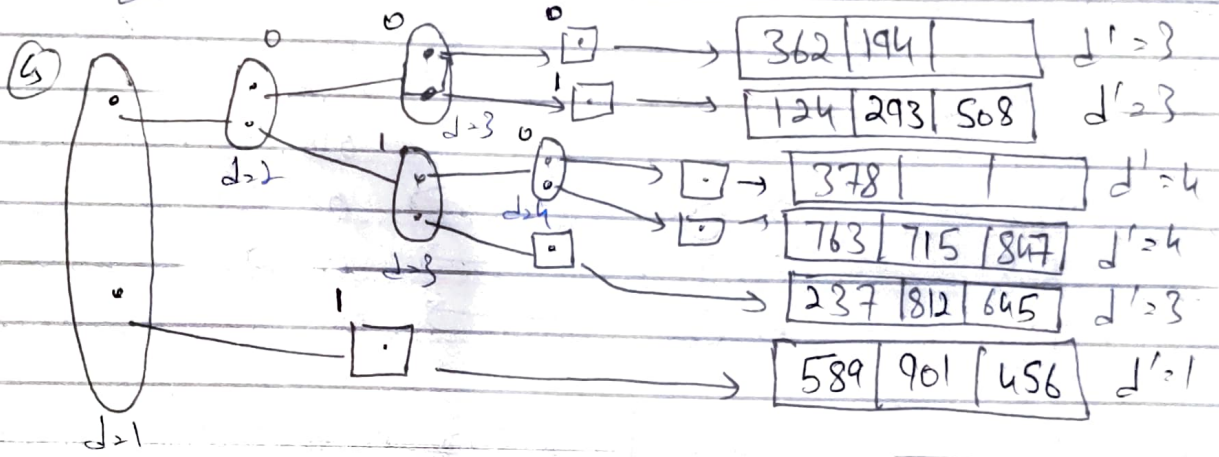
Decimal	(value + 10) * 1.12 → Hashed	Binary
237	7	0111
589	11	1011
124	2	0010
763	5	0101
901	11	1011
456	10	1010
378	4	0100
812	6	0110
645	7	0111
293	3	0011
715	5	0101
847	5	0101
362	0	0000
508	2	0010
194	0	0000











This way all 15 values have been added to the dynamic hashed tree structure

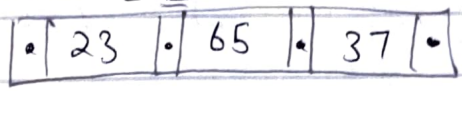
Next Page

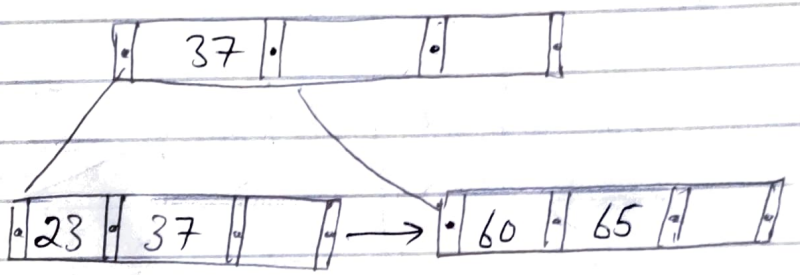
→ **Question-07:**

23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21, 10, 74, 78, 15, 16, 20, 24, 28, 39, 43, 47, 50, 69, 75, 8, 49, 33, 38

(a) Insert 23, 65, 37, 60:

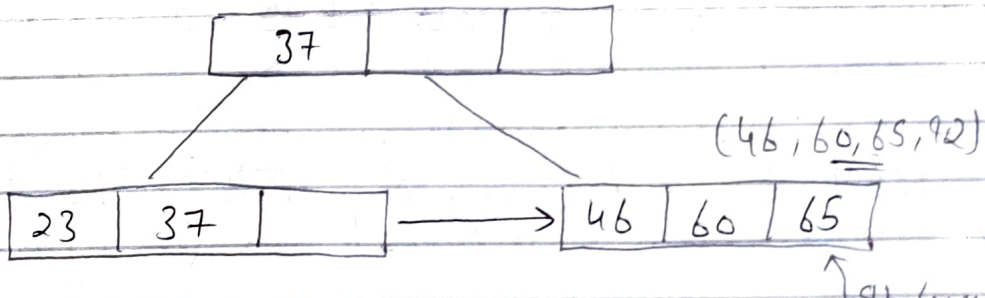
(23, 37, 60, 65)





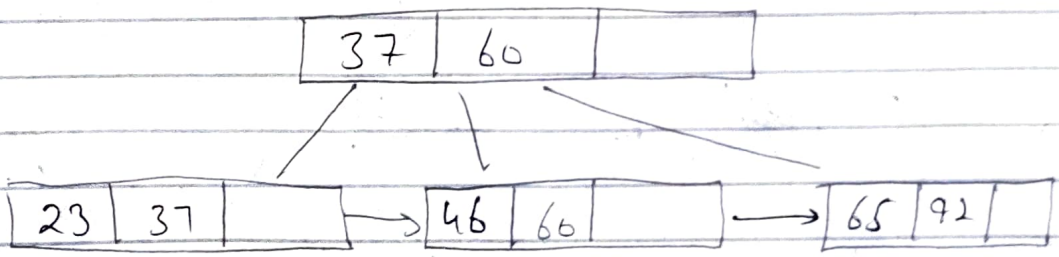
(60 inserted after splitting)

(b) Insert 46:

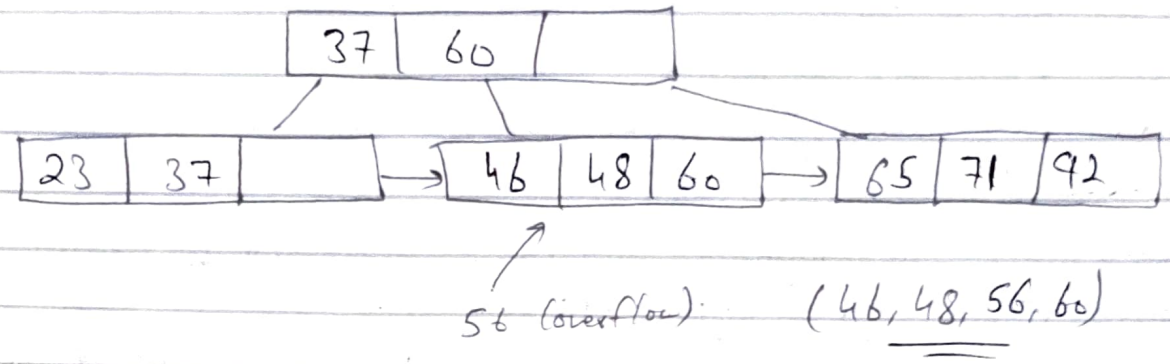


(46, 60, 65, 92)

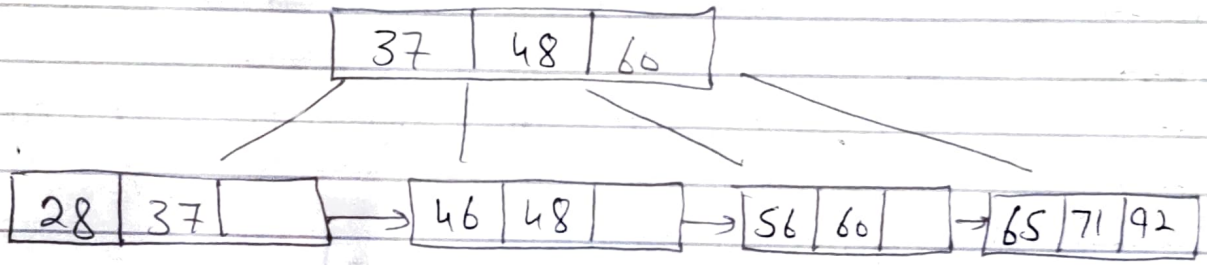
(c) Insert 92:



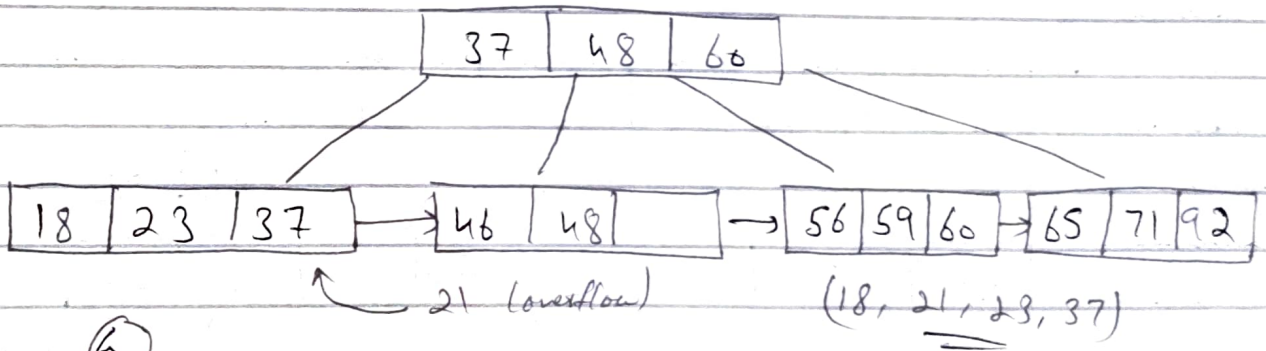
(5) Insert 48, 71:



(5) Insert 56:

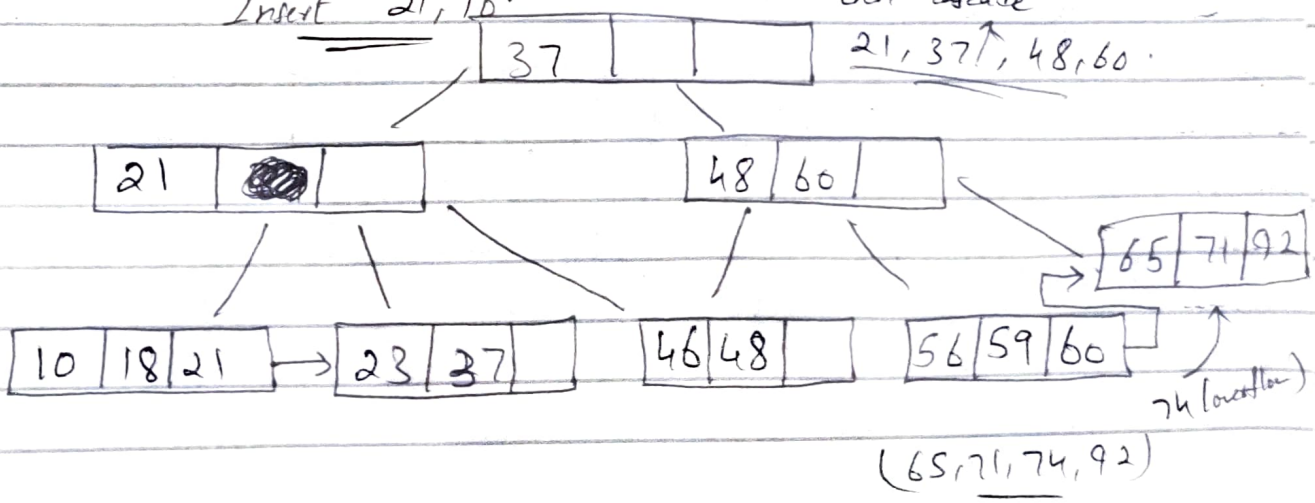


(5) Insert 59, 18:

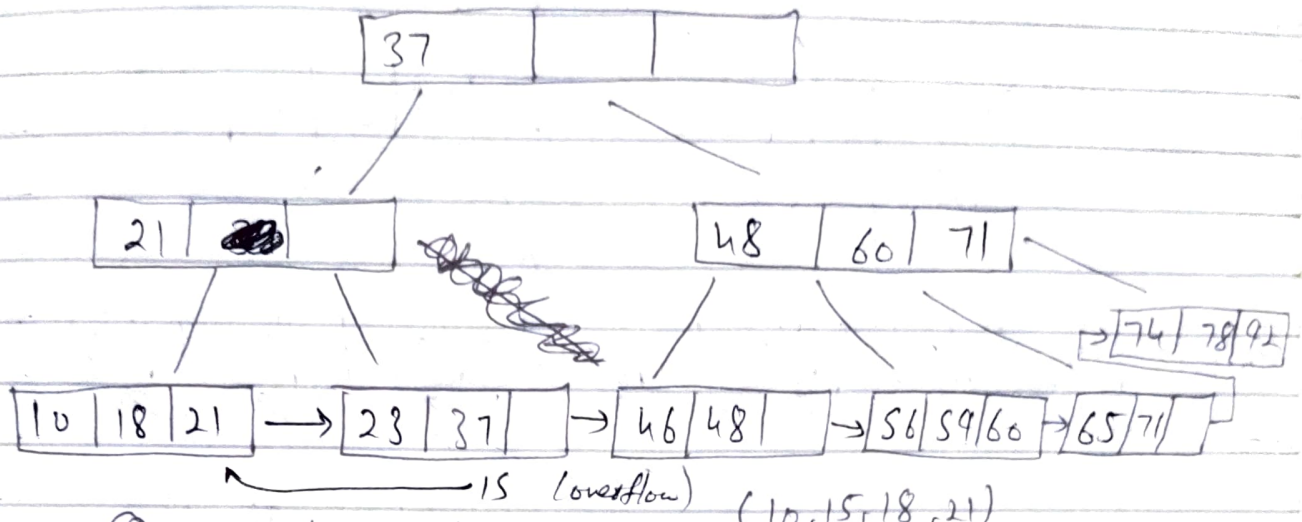


(5) Insert 21, 10:

then cascade 21, 37, 48, 60.

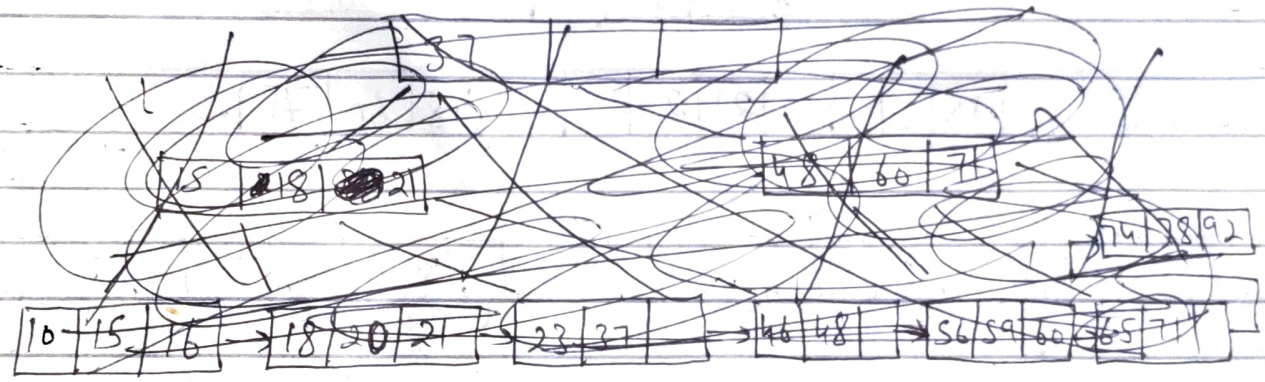


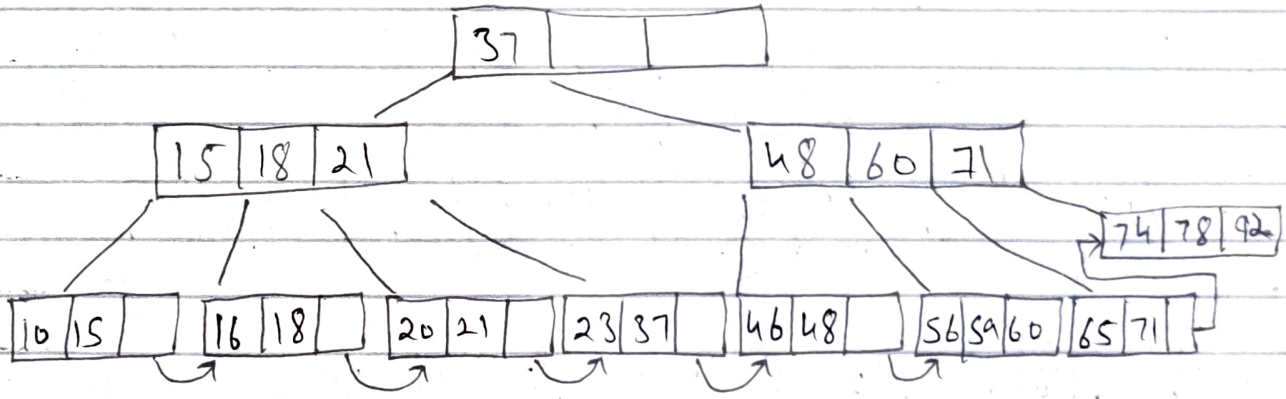
(5) Insert 74, followed by 78:



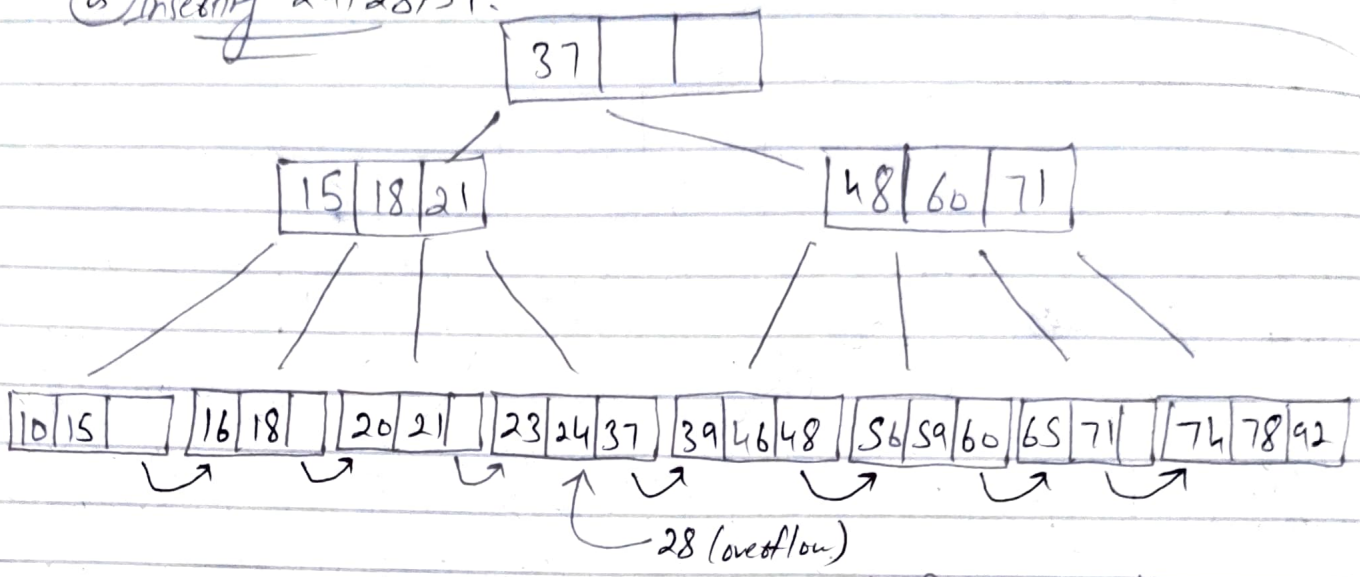
15 (overflow) (10, 15, 18, 21)

(6) Insert 15, 16, 20:

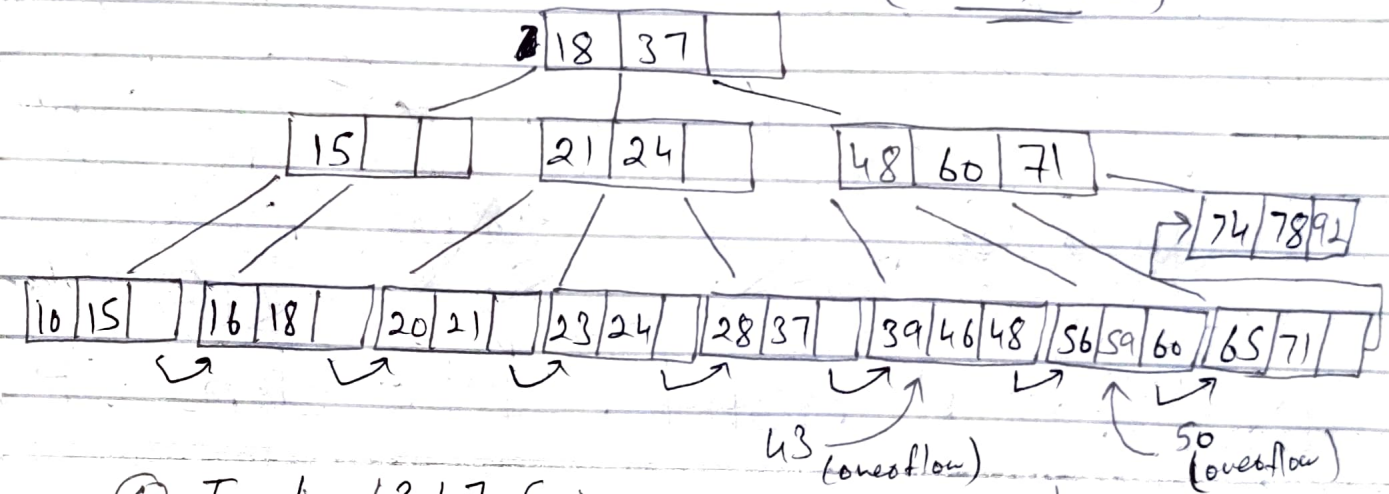




(4) Inserting 24, 28, 39:



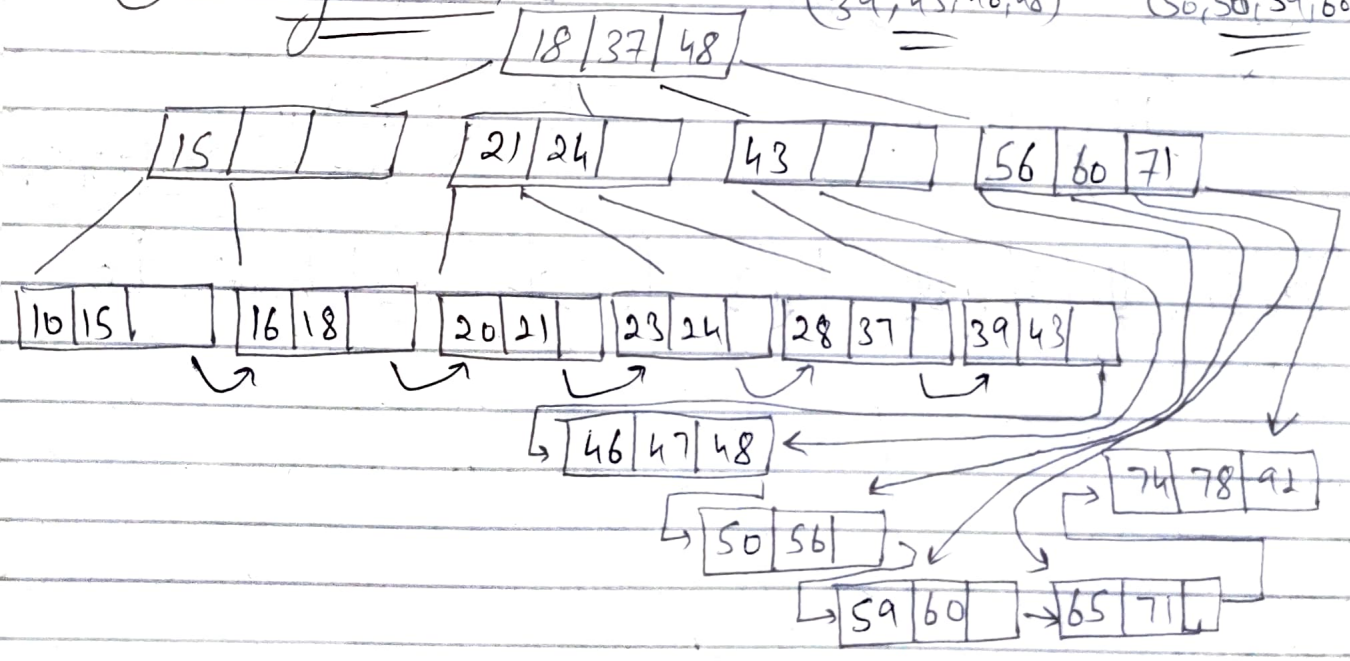
(23, 24, 28, 37)



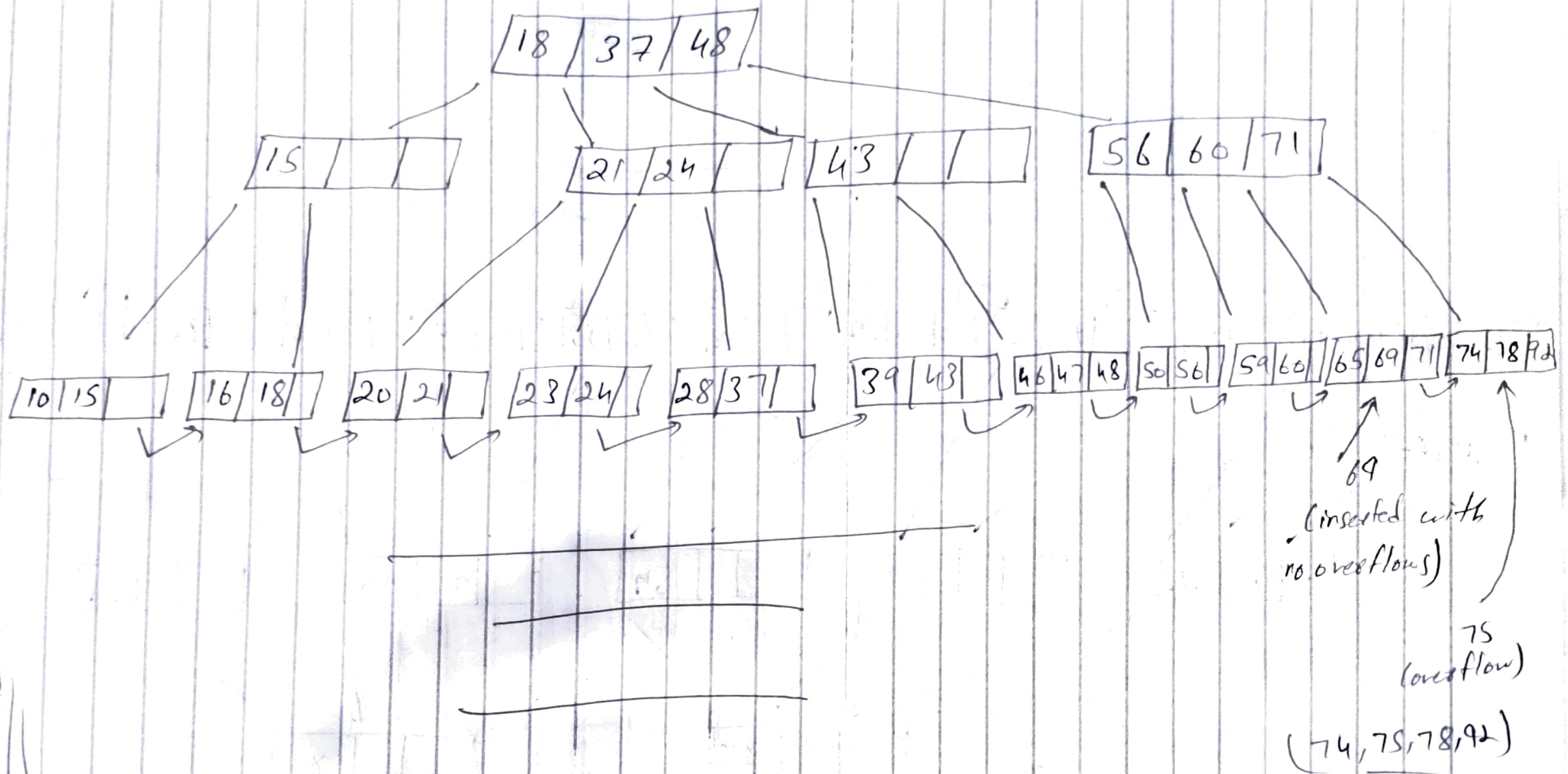
(39, 43, 46, 48)

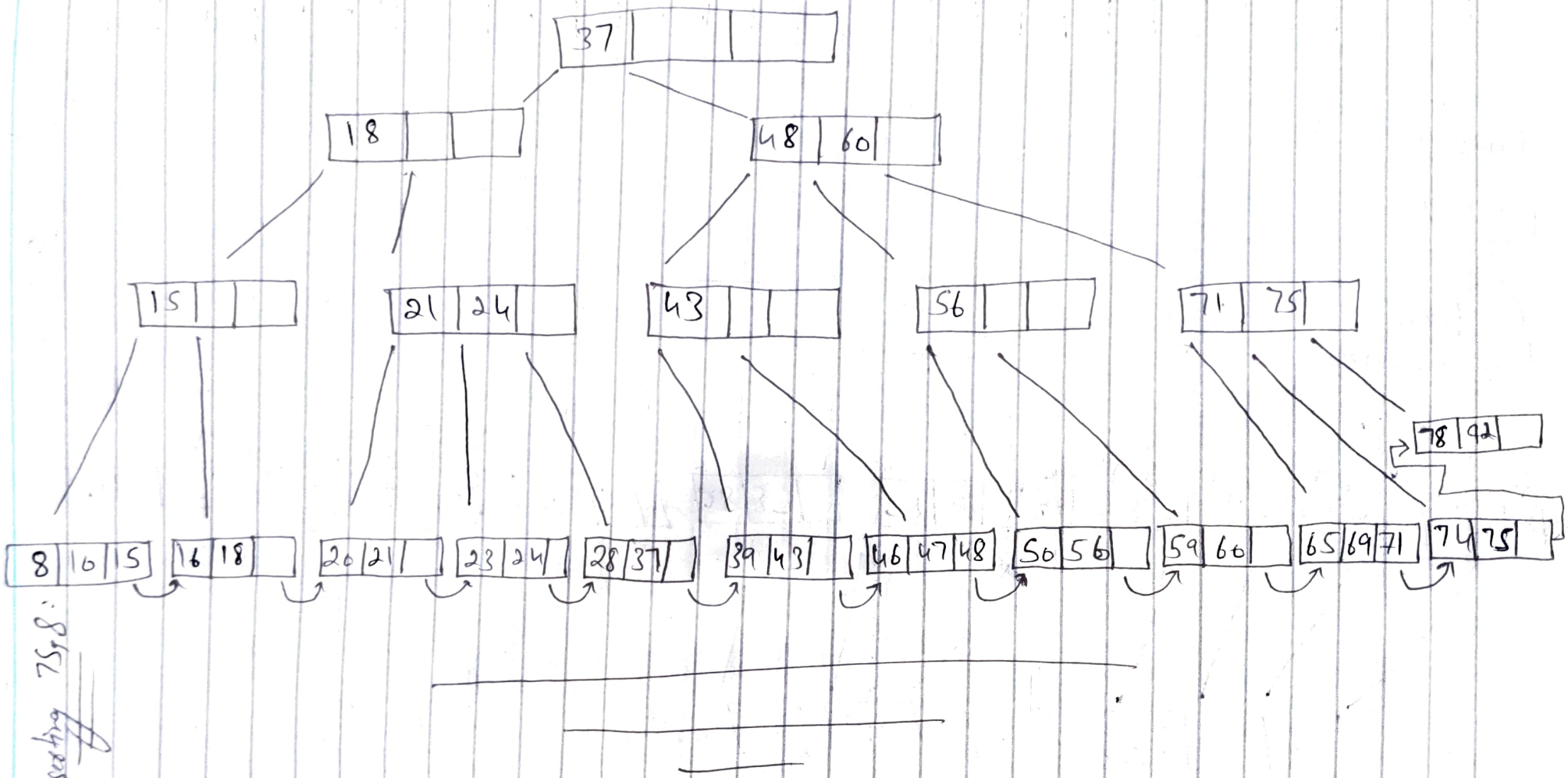
(50, 56, 59, 60)

(5) Inserting 43, 47, 50:



⑥ Inserting 69:





⑤ Inserting 75, 8:

(ADB)

⑥ Inserting 49, 33, 38 (No overflows):