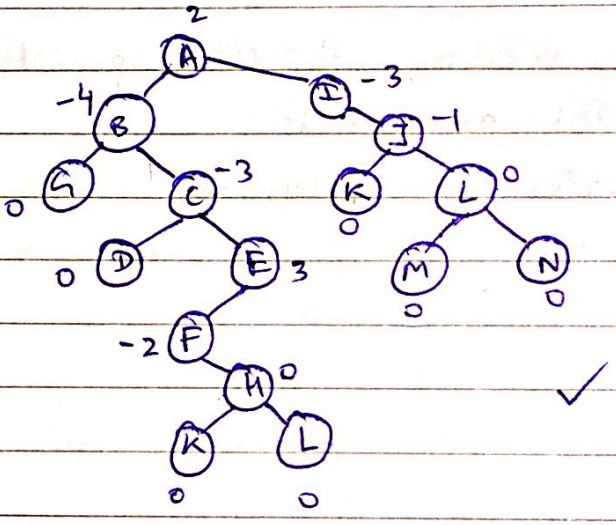
AVL Tree

Height Balanced Tree:

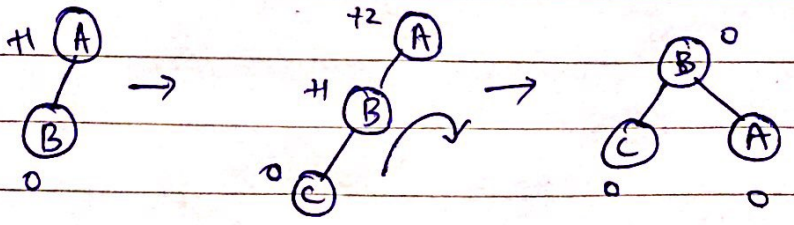
NOTE: Balance factor = height of left subtree - height of right subtree

Q.1 Find balance factor.

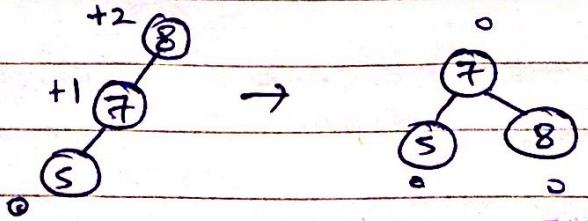


NOTE: AVL tree has balance factor of +1, 0 or -1 for ~~V~~ nodes. If bf goes beyond this then rotation is done to maintain ~~bf~~ bf = +1, 0, -1

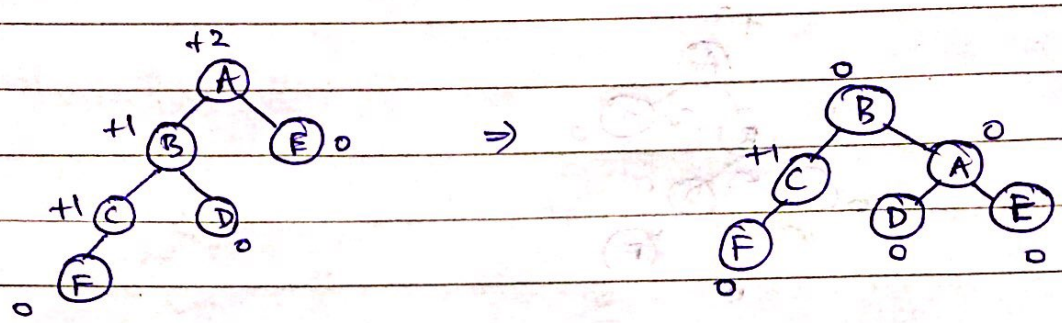
LL Rotation:



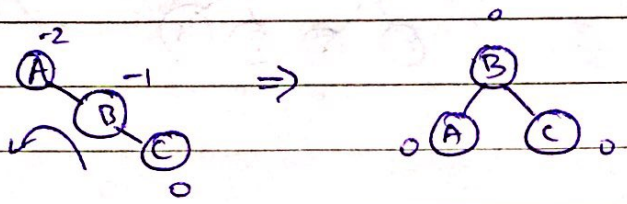
bst eg:

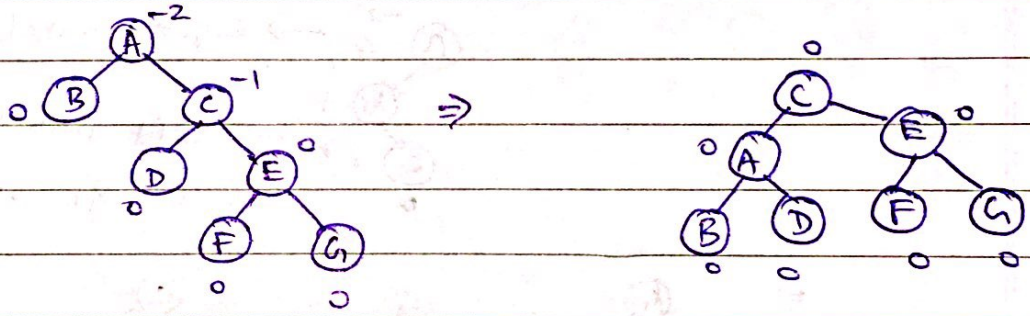


Teacher's Signature

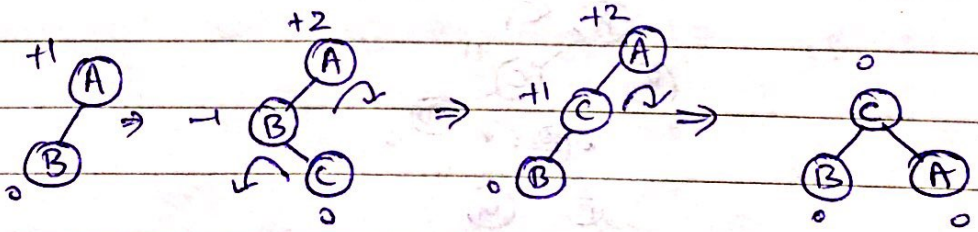


R R Rotation :

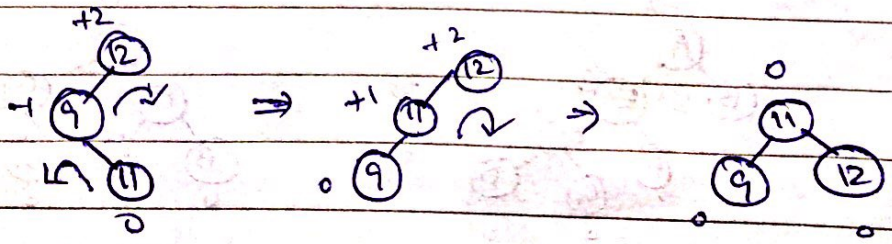




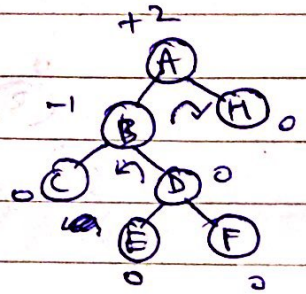
LR Rotation :

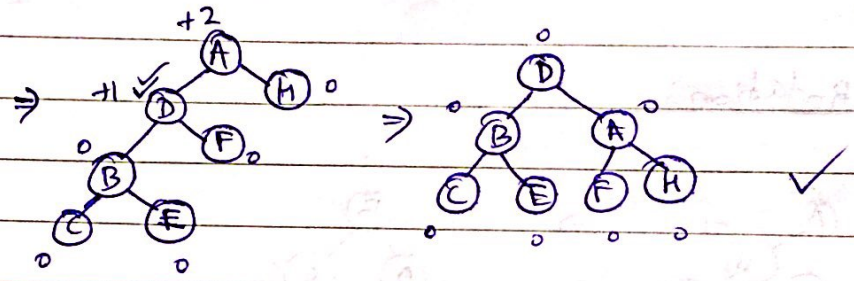


Q:2

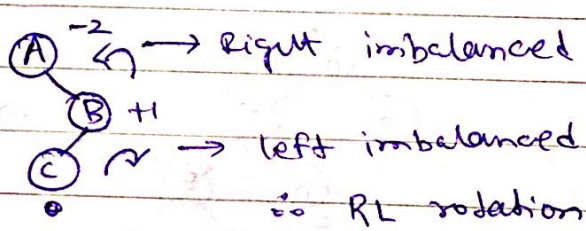


eg: 3





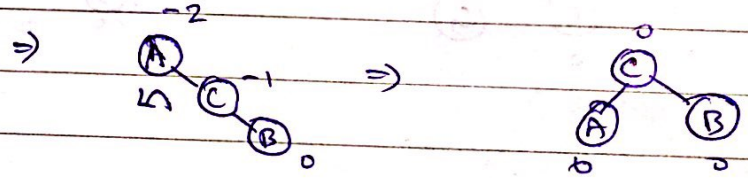
RL Rotation:



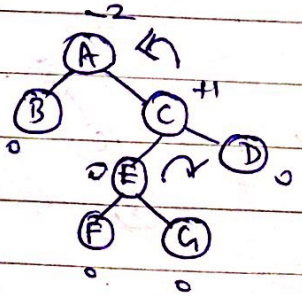
→ Right imbalanced

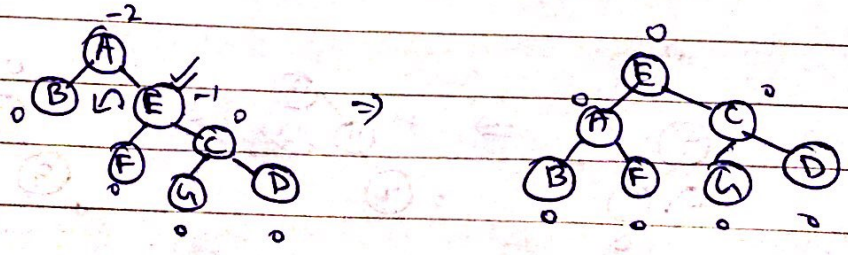
→ left imbalanced

∴ RL rotation required



eg: 4

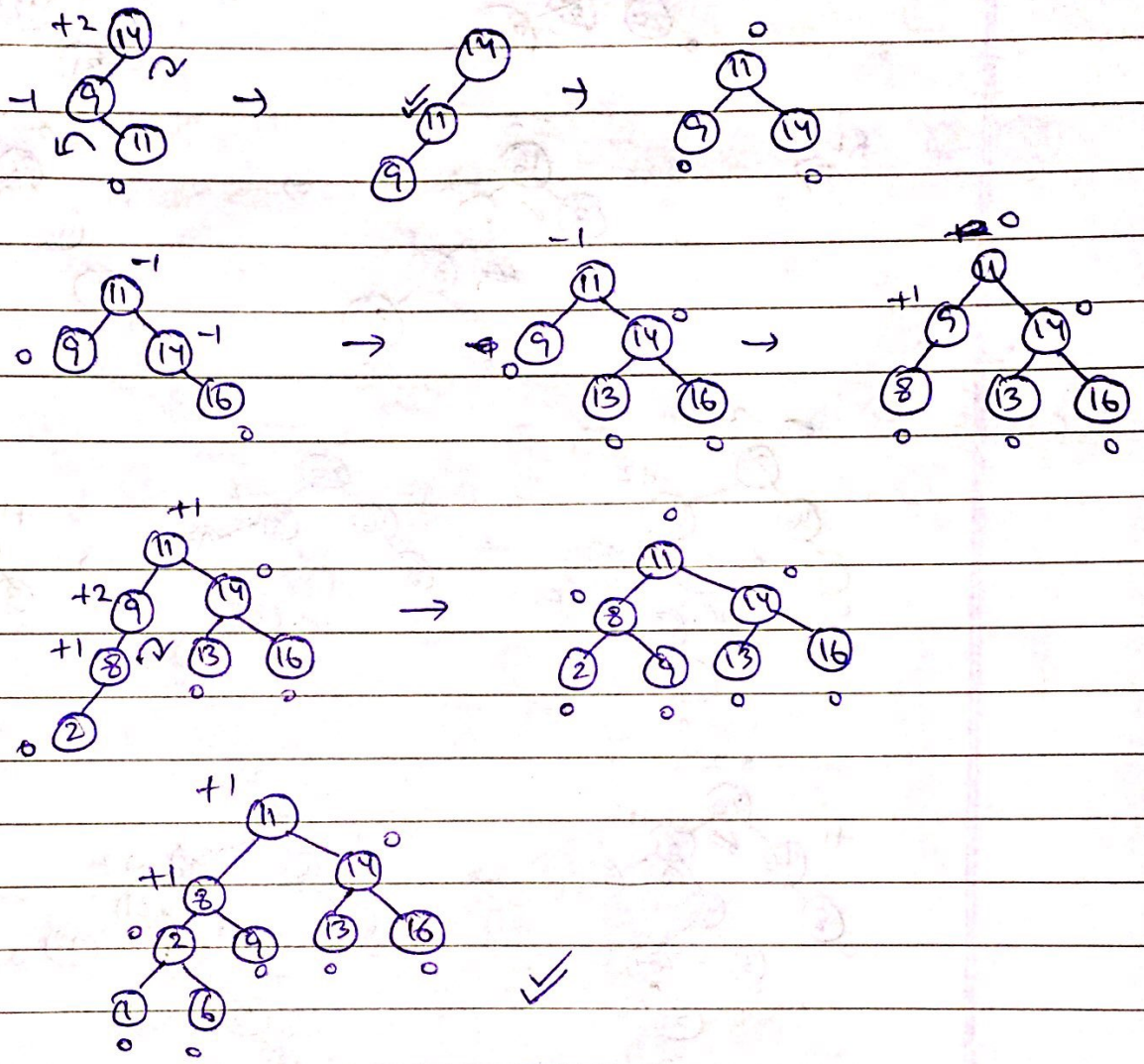




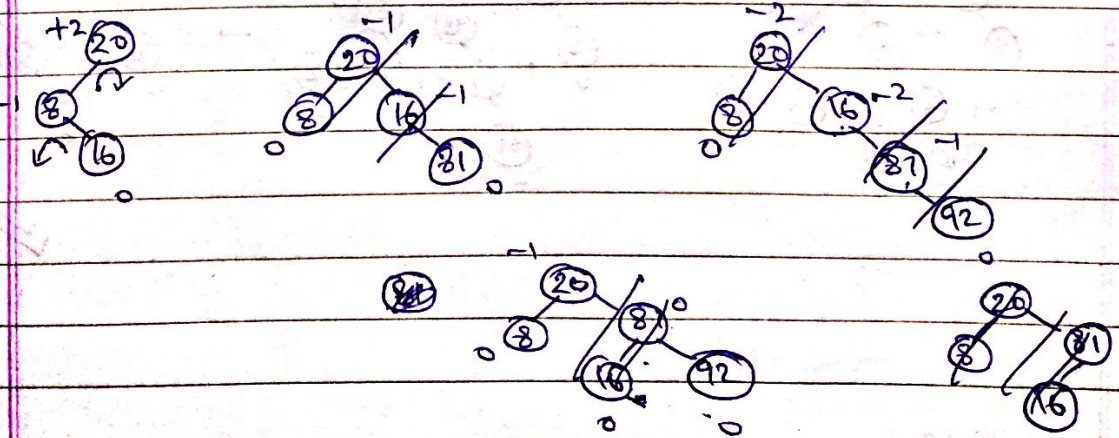
Teacher's Signature

INSERTION IN AVL

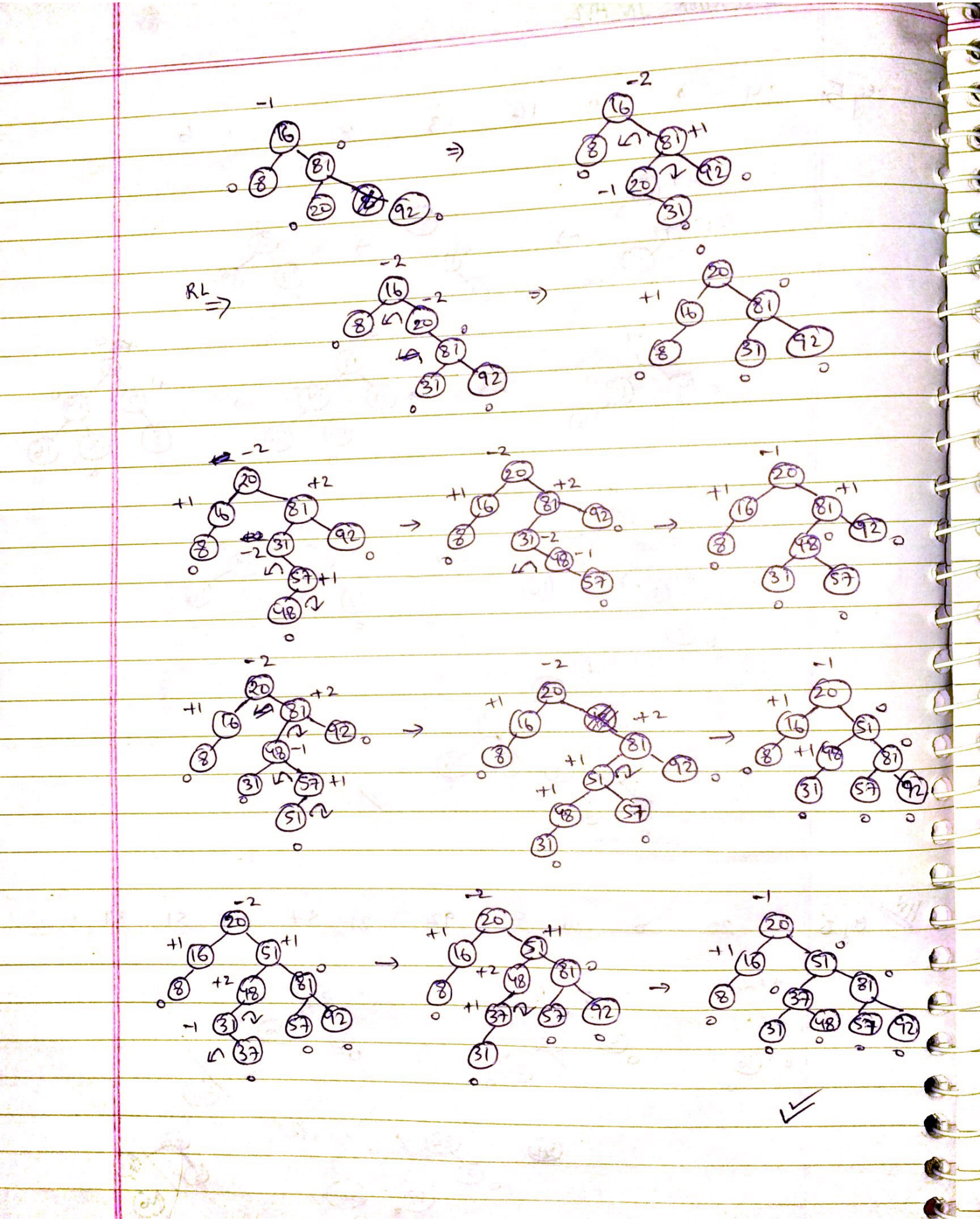
eg 5 14 9 11 16 13 8 2 1 6



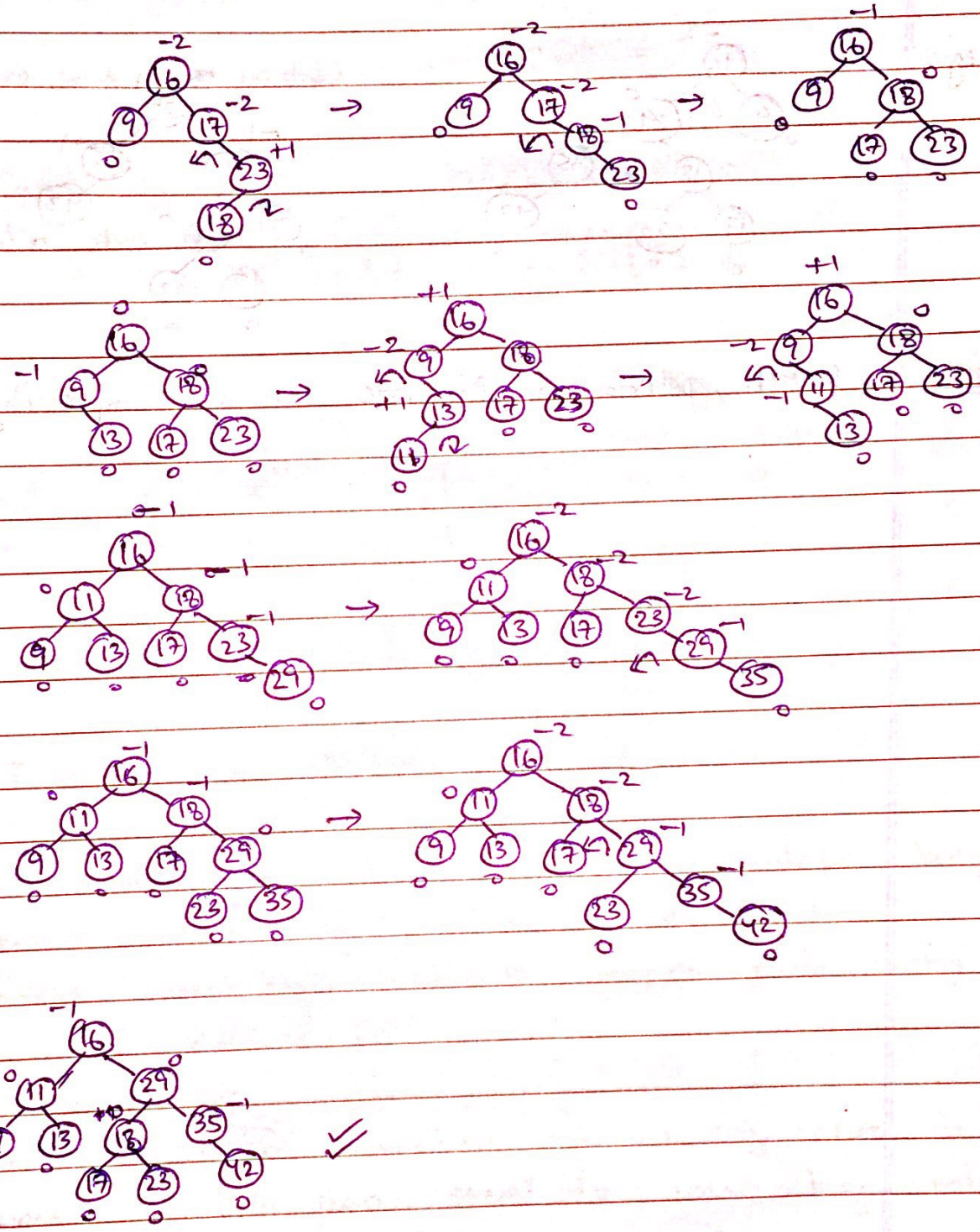
HW eg 6 20 8 16 81 92 31 57 48 51 37



Teacher's Signature

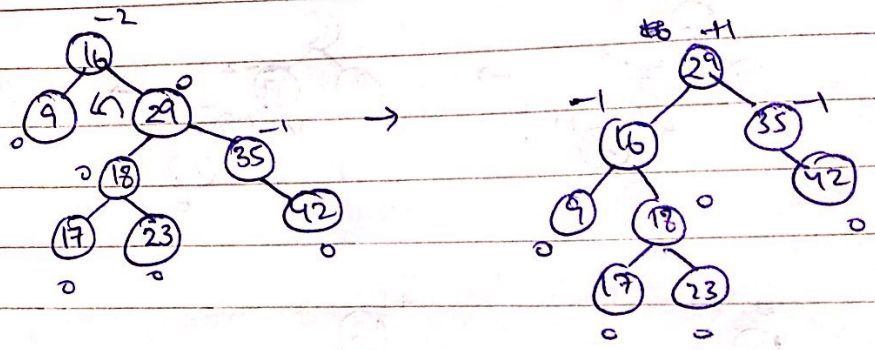

RL
Scanned by CamScanner

eg 3: 16, 9, 17, 23, 18, 13, 11, 29, 35, 42



NOTE: Insertion in AVL tree is of complexity $\log_2 n$.

eg:



NOTE: Search, Deletion in AVL tree is of complexity $\log_2 n$

B-Tree (Multiway Tree) :-

m-way (m paths)

Rule: Max. child = m Max Key = m-1

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

eg:

5 way

	child	key
Max.	5	4
Min	(3)	2

8 way

	child	Key
Max.	8	7
Min	4	3

NOTE: Only root can have 1 value

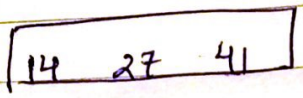
Insertion in B-Tree:

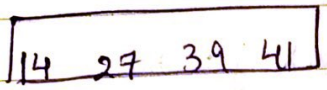
Each node in B-Tree contains multiple key values and they are in sorted order.
Insert new key value in proper node using the same process of BST.

CaseI: If after doing insertion no. of key values in the node is less than equal to max. key values, it is okay (no modification req.)
Else, split the node by making middle node as parent node. All key values at left as left child and all key values at right as right child.

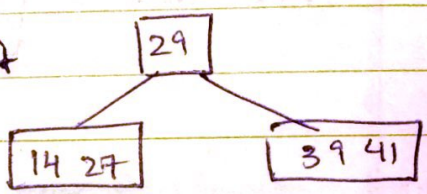
Teacher's Signature

eg: 5-way





→ split



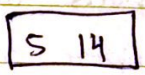
eg: 14, 5, 21, 1, 6, 8, 9, 11, 13, 27, 3, 18, 19, 15, 34, 39

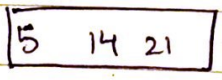
Sol: 4-Way

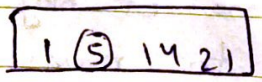
Max: 4 3

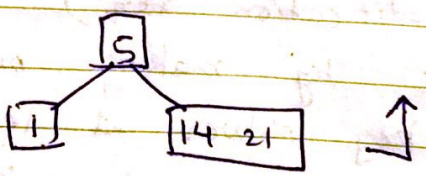
Min: 2 1

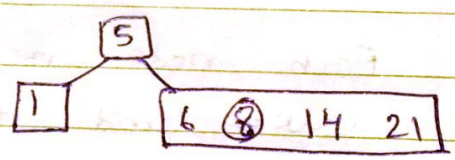
child key

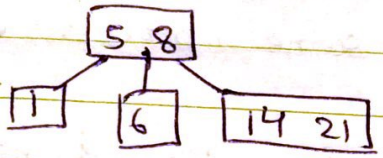


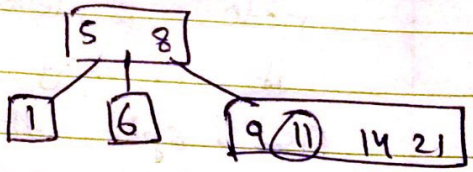




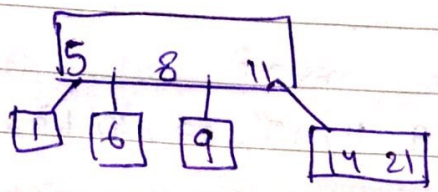


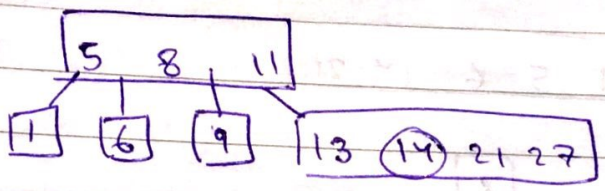


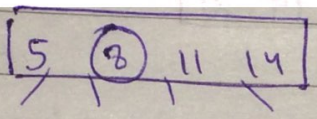


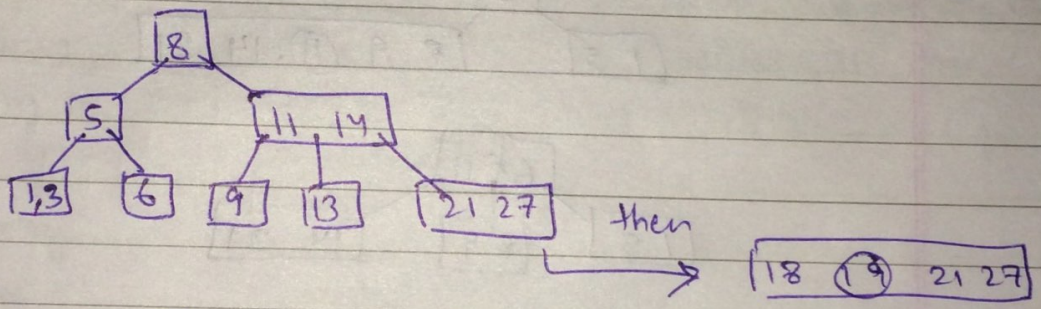


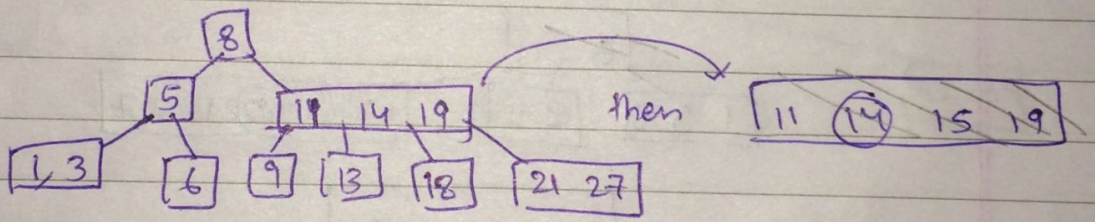
Teacher's Signature

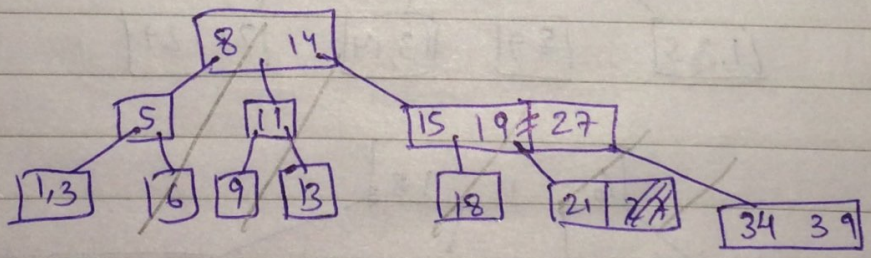


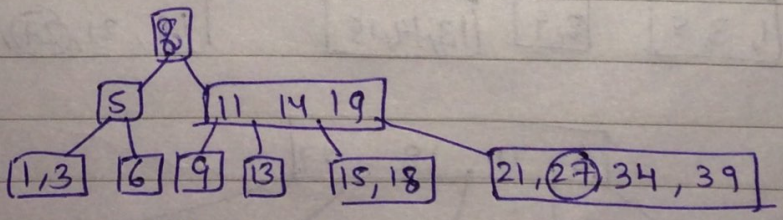


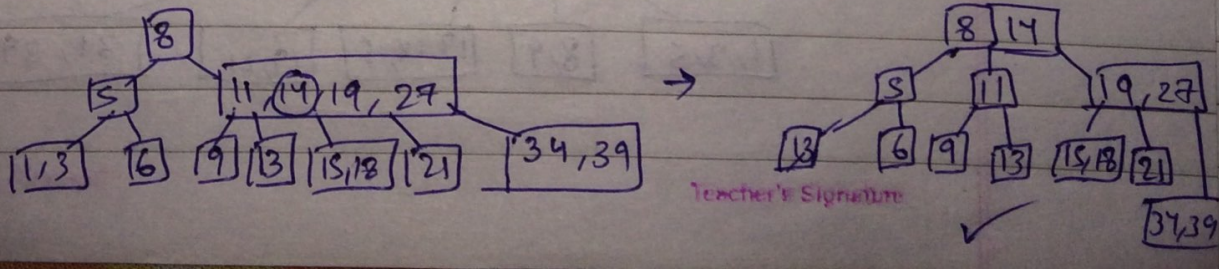








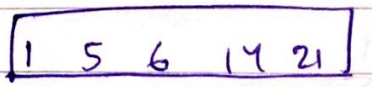


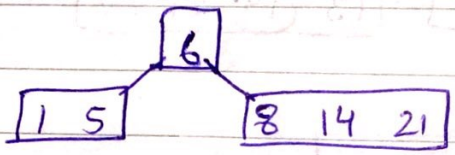


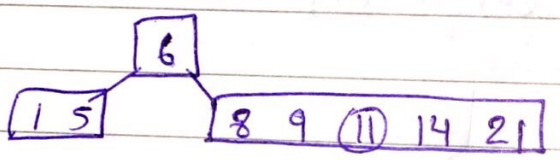
Teacher's Signature

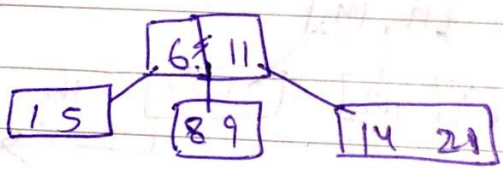
eg: Using 5-way tree insert
14, 5, 21, 1, 6, 8, 9, 11, 13, 27, 3, 18, 19, 15, 34, 39

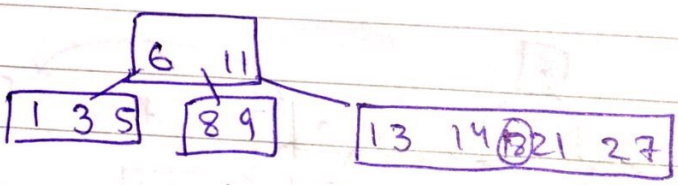
Sol:

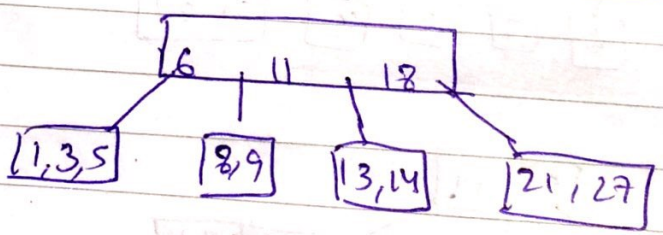


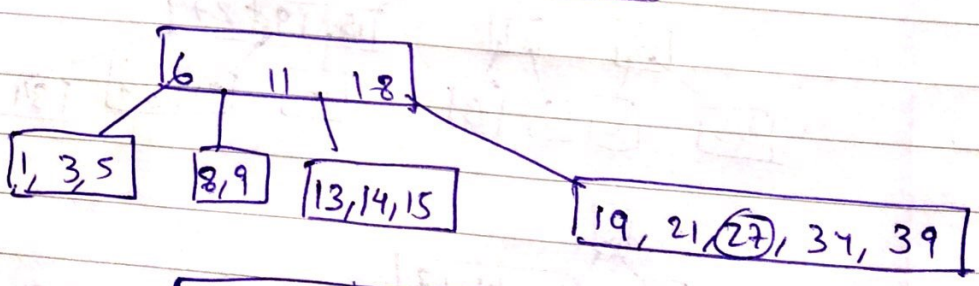


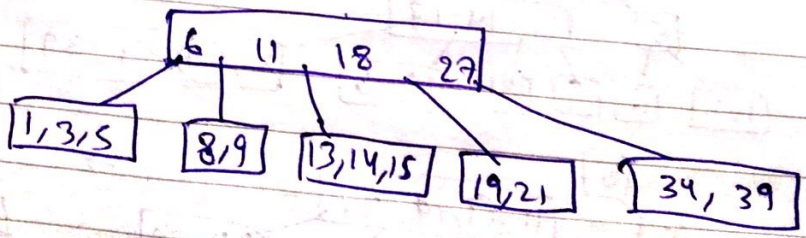












✓

Teacher's Signature

PAGE NO.:

Deletion in B-tree

* **Leaf node:**

→ If node contains more than min key value then simply delete the value

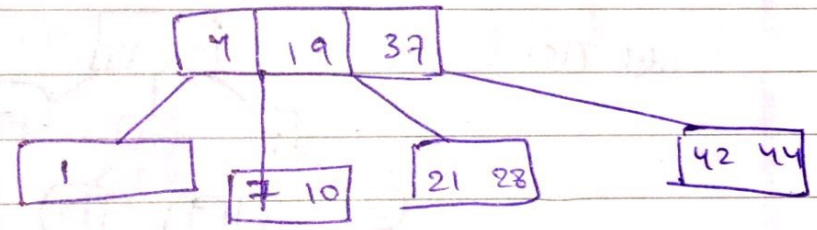
* **Non-leaf node:**

If node contains min keys then replace it with its inorder predecessor or successor whosoever is containing more than min key values. If none is having so, then merge them.

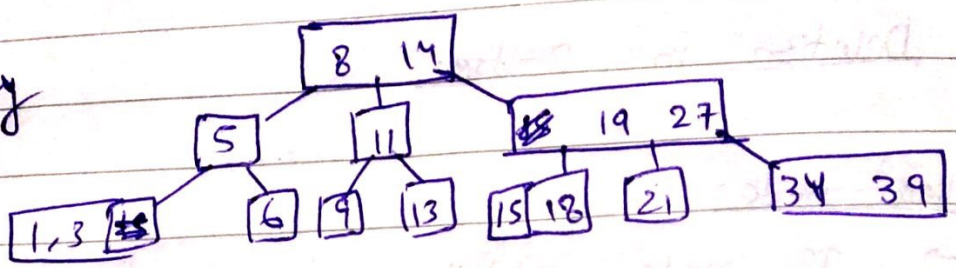
eg: 4-way

min key = 1

max keys = 3



eg: 4-way



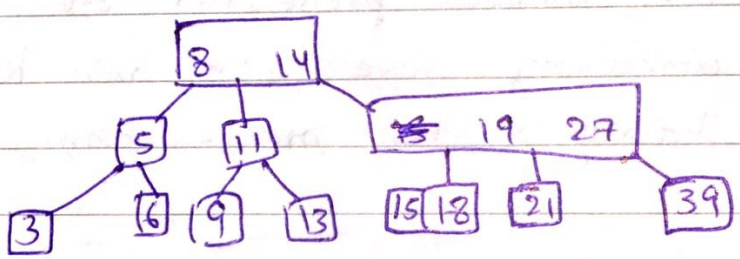
Delete (1)

del (34)

~~del (15)~~

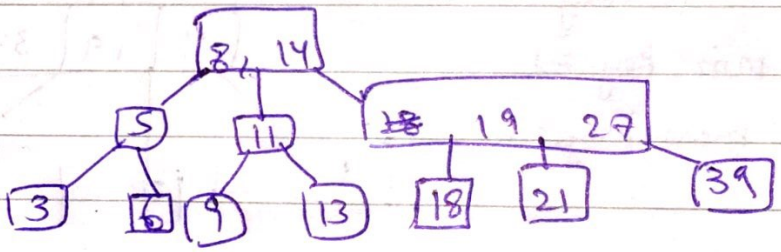
del (18)

Sol: del (1)

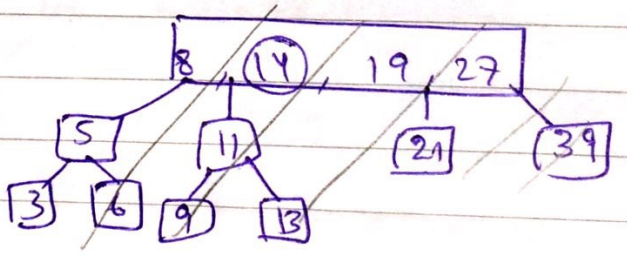


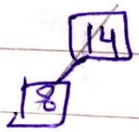
del (34)

del (15)

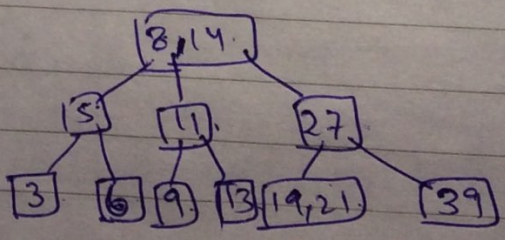


~~del (18)~~





del (18)



NOTE: 2-3 tree (3 Way)

SORTING ALGORITHMS:

(I) Selection Sort:

Min. value is selected and placed in its proper place.

		After pass 1	II	III	IV	V
0	14	1	1	1	1	1
1	9	9	3	3	3	3
2	17	17	17	6	6	6
3	3	3	9	9	9	9
4	6	6	6	17	17	14
5	1	14	14	14	14	17

~~for(i=0 ; i<n ; i++)~~

```
void selection (int a[], int n)
{ int i, j, min, p;
  for (i=0 ; i< n-1 ; i++)
    { min = a[i]; p=i;
      for (j= i+1; j<n ; j++)
        {
          if (a[j] < min)
            { p=j;
              min = a[j];
            }
        }
    swap( a[i], min, p)
```

Teacher's Signature

or

// swap (a[i], min, k)

```
        a[p] = a[i];
        a[i] = min;
    } // i loop
} // selection
```

NOTE: Time complexity of selection sort is $O(n^2)$ ∀ possible case

(2) **Insertion sort**: Insert an ele in already sorted array

0	14	9	9	3	3	1
1	9	14	14	9	6	3
2	17	17	17	14	9	6
3	3	3	3	17	14	9
4	6	6	6	6	17	14
5	1	1	1	1	1	17

```
void insertion (int a[], int n)
{ int i, j, ele;
  for (i=1; i<=n-1; i++)
    { ele = a[i];
      j = i-1;
      while ((j>=0) && (a[j]>ele))
        { a[j+1] = a[j];
          j = j-1;
        }
      a[j+1] = ele;
    }
}
```

Teacher's Signature

~~DSA - BAD 7XXX~~

2-3 Trees:

→ Every internal node is either a 2 node or a 3-node

→ A 2-node has one key and 2 children / subtrees

→ A 3-node has 2 keys and 3 children / subtrees,

→ A 2-3 tree doesn't resemble a full binary tree

→ If a 2-3 tree does not contain 3-nodes, it is like a full binary tree since all its internal nodes have 2 children and all its leaves are at the same level.

→ If a 2-3 tree does have 3 children, the tree will contain more nodes than a full binary tree of same height

→ So, a 2-3 tree of height h has at least $2^h - 1$ nodes.

③ Bubble sort :

0	14	9	9	3	1
1	9	14	3	1	3
2	17	3	1	9	9
3	3	1	14	14	14
4	1	17	17	17	17

```
void bubblesort (int a[], int n)
{  int i, j, flag;
   for (i=0; i<n-1; i++)
    {  flag=0;
       for (j=0; j<n-i-1; j++)
          { if (a[j] > a[j+1])
                {swap (a[j], a[j+1]); flag=1;}
          }
       if (flag==0) break;
    }
}
```

complexity ≡ O(n) ∀ best, ~~worst~~, ~~avg~~. case
≡ O(n²) ∀ worst, avg case

④ Merging : It takes 2 sorted arrays as input and o/p a sorted array.

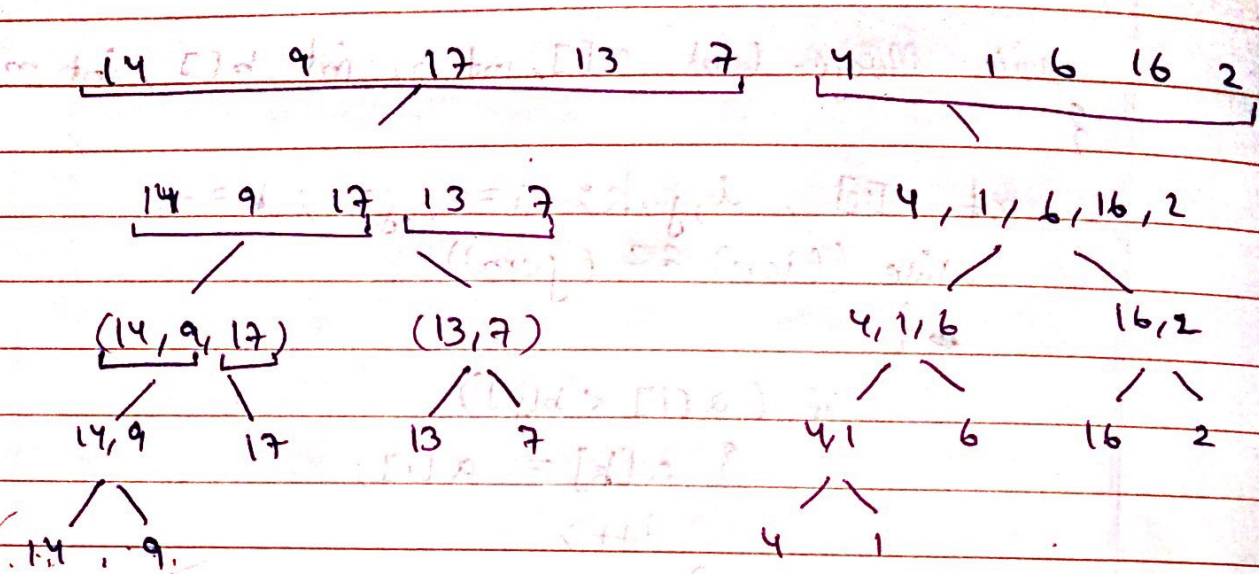
```
void Merge (int a[], int n, int b[], int m)
{
  int c[20], i, j, k; i=0; j=0; k=0;
  while ((i<n) && (j<m))
  {
      if (a[i] < b[j])
          { c[k] = a[i];
            i++;
          }
      else { c[k] = b[j];
             j++;
           }
      k++; }
  if (i==n)
  { for ( ; j<m; j++)
      c[k++] = b[j];
  }
  else { for ( ; i<n; i++)
           c[k++] = a[i];
       }
}
```

⑤ Merge Sort:

0	1	2	3	4	5	6	7	8	9
14	9	17	13	7	4	1	6	16	2

Divide & conquer: $l=0$, $h=9$, $m=\frac{0+9}{2}=4$

Teacher's Signature



~~void Mergesort (int a[], int n)~~

```
{
  void Mergesort (int low, int high)
{
  int mid;
  if (low < high)
  {   mid = (low + high)/ 2;
      Mergesort (low, mid);
      Mergesort (mid+1, high);
      Merge (low, mid, high);
  }
}
  void Merge (int low, int mid, int high)
{
  int c[20], i, j, k;
   i = low; j = mid +1; k = low;
while (( i<= mid) && (j<= high))
  {
     if (a[i] < a[j])
```

Teacher's Signature

```
{ c[k] = a[i];
  i++;
}
else { c[k] = a[j];
       j++;
     }
  k++;
}
if (i==n)
{ // copy b's elements)
  for ( ; j<n; j++)
     {  c[k++] = a[j];
}  }
 else {
        for ( ; i<n; i++)
             c[k++] = a[i];
    for (i=low; i<=high; i++)
         a[i] = c[i];
```

Complexity:

$T(n) = T(n/2) + T(n/2) + n$

$T(n) = 2T(n/2) + n$

$T(n) = 2T(n/2) + n$

$= 2[2T(n/4) + n/2] + n$

$= 4T(n/4) + n + n = 4T(n/4) + 2n$

$= 4(2 \cdot T(n/8) + n/4) + 2n = 8T(n/8) + 3n$

~~$T(n) = 2^i$~~

Teacher's Signature

$$T(n) = 2^i T(n/2^i) + i \times n$$

$$= n \cdot T(1) + n \times \log_2 n$$

$$= n + n \log_2 n$$

$$T(n) = O(n \log_2 n)$$ //

$$\frac{n}{2^i} = 1$$

$$n = 2^i$$

$$i = \log_2 n$$

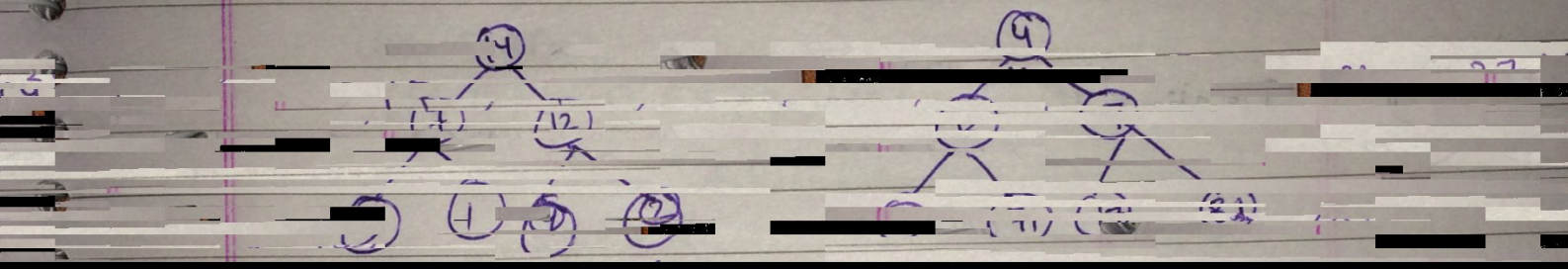
HEAP:

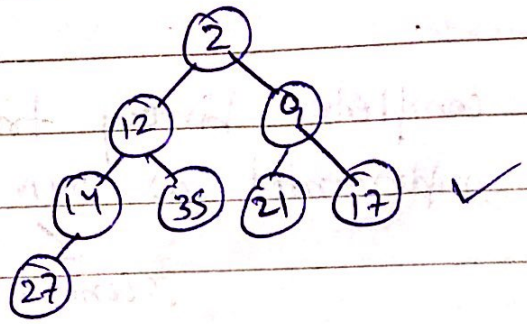
→ It's almost complete binary tree

→ It can be implemented as MIN heap / MAX heap

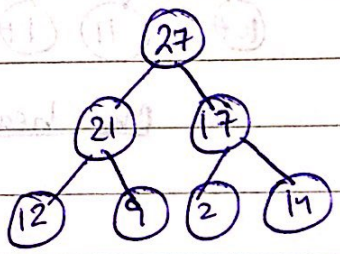
parent < child

parent > child

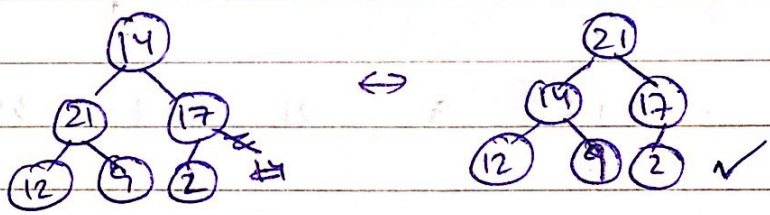




II. Deletion :



del (27)



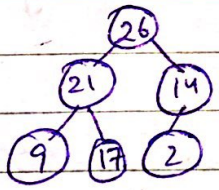
DATE: 1/4/17

Insertion in Heap (Algorithm by HMS) Horowitz & Sahni

```
insert (a,n)
{
    i=n;
    item = a[n];
    while ((i>=1) && (a[i/2] < item))
        { a[i] = a[i/2];
          i = i/2;
        }
    a[i]=item;
}
```

```
for (n=1; n<=p; n++)
{
    sf("%d", a[n])
        adjust(a,n)
}
```

14 9 17 21 26



1	2	3	4	5	6
26	21	14	9	17	2

```
Delmax (a,n,x)
{   if (n==0)
        if ("Heap empty");
    else
        { x = a[i];
          a[i] = a[n];
          adjust (a,1,n-1)
        }
}
```

Teacher's Signature

```
adjust (a, i, n)
{
    J = 2 * i;
    item = a[i];
    while (J <= n)
    {
    if ((J<n) and (a[J] < a[J+1])) J = J+1;
    if (item >= a[J])
                    break;
      a[J/2] = a[J];
        J = 2 * J;
    }
      a[J/2] = item;
}
```

Q. ~~Implement 3-ary heap.~~

Q1: Which one of the following is valid sequence of elements in an array representing 3-ary heap.

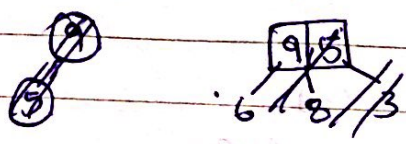
(A) 1, 3, 5, 6, 8, 9

(B) 9, 6, 3, 1, 8, 5

(C) 9, 3, 6, 8, 5, 1

(D) 9, 5, 6, 8, 3, 1

Q2: Insert 7, 2, 10, 4 in (D)



Teacher's Signature

Q.3: In a binary max heap what is the time complexity to find min element?

Ans: O(n)

Q.4: Complexity of converting an array to heap:

(A) O(n)

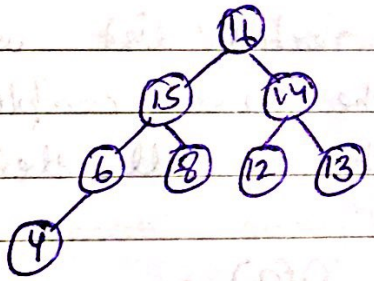
(B) O(log n)

(C) O(n log n)

(D) None of these

Q.5: How many swaps are required to extract max. element from max. heap.

16, 15, 14, 6, 8, 12, 13, 4 – Heap structure



Sol: 2 swaps

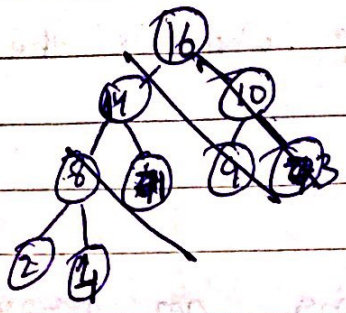
[Note: 4 is just copied to 16 not swapped,
15 is swapped with 4,
4 is swapped with 8]

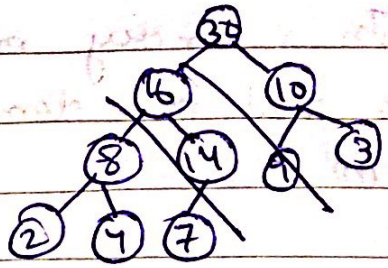
Q.6: Consider a max. heap whose Inorder traversal is:

2, 8, 4, 14, 1, 7, 16, 9, 10, 3

Insert 30 in this heap. Write Inorder traversal for new heap.

Teacher's Signature





Ans: 2, 8, 4, 16, 1, 14, 7, 30, 9, 10, 3 ✓

Solutions:

Q. 7 Suppose there are $\log n$ sorted list with with $n/\log n$ elements. Find the time complexity of producing a sorted list of all elements.

Sol: $O\left(\frac{n}{\log n} \cdot \log n\right) \cong O(n)$. ✓

HASHING: Used to search an element in min. no. of comparison.

NOTE: Time complexity of hashing is O(1).

The hash fun(): $h(x) = x \% 10$ is called Division Remainder Method.

Mid-Square Method:

pick middle y digits

$h(x) = x^2$ $\quad y \rightarrow$ is defined based on array size

$x = 12$ $\quad x = 1632$

$x^2 = 144$ $\quad x^2 = 2663\underline{24}$

Folding Method:

→ x is partitioned into equal size part $x_1, x_2, \ldots x_m$

→ Add the subpart

→ Ignore carry

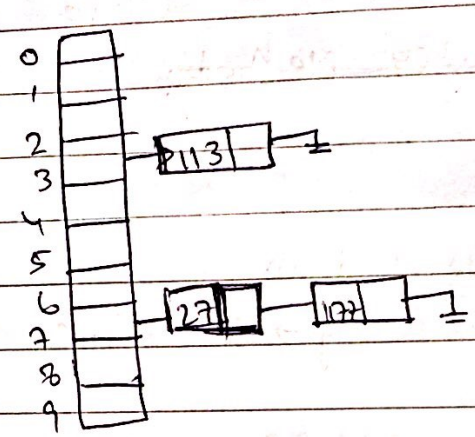
eg $x = \underline{17}, \underline{23}, \underline{1}$, $y = 17 + 23 + 1 = 41$

$x = \underline{92}, \underline{17}, \underline{3}$, $y = 92 + 17 + 3 = 112$

Hash collisions: When 2 or more values are generated through same hash function, it's called hash collision.

Teacher's Signature

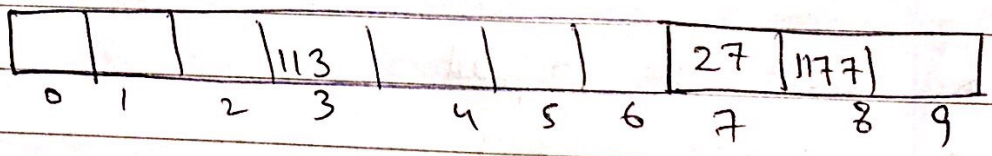
To solve the above problem linked list can be created. But it's not efficient as time complexity again becomes O(n).



Open addressing :

↳ "Linear probing"

$y' = (y+i) \% 10$



"Quadratic probing" :

$$y' = (y + c_1 i^2 + c_2 i + c_3) \% 10$$

Double Hashing :

two diff. hash functions

$y = h(x)$, $y' = h'(x)$

$l = h, h+h', h+2h', h+3h'$

↳ location value

Teacher's Signature

PAGE NO.:

Rehashing: Increasing the storage space for data and performing hashing again of the elements.

Q1. A hash table of length 10 use open addressing with hash function $h(k) = k \% 10$ and use linear probing to resolve collision. After inserting 6 values ~~your~~ hash table is as follows:-

0	1	2	3	4	5	6	7	8	9
		42	23	34	52	46	33		

(i) Which one of the following gives a possible order in which key values could have inserted in table

(a) 46, 42, 34, 52, 23, 33 — A - 52

(b) 34, 42, 23, 52, 33, 46 — B - 33

(c) 46, 34, 42, 23, 52, 33 — D - 33

(d) 42, 46, 33, 23, 34, 52

(ii) How many different insertion seq. of key val(s) using the same hash fun() and linear probing are possible?

(a) 10

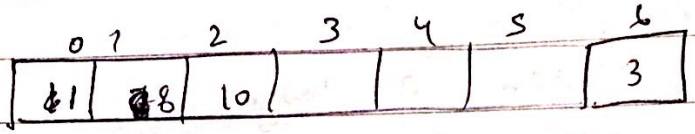
(b) 20

(c) 30

(d) 40

PAGE NO :

Q2. Consider hash table of size 7 (starting index 0)
$h(x) = (3x + 4) \% 7$
Hash table is initially empty. Insert 1, 3, 8, 10. If collision use linear probing



(a) 8 _ _ _ _ _ 10

(b) 1 _ _ _ _ _ _ 3

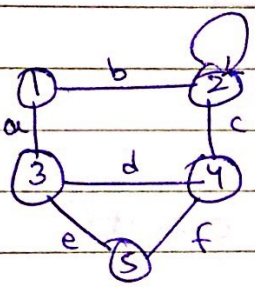
(c) ✓ 1 8 10 _ _ _ 3

(d) 1 10 8 _ _ _ 3

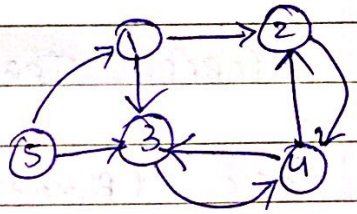
DATE: 11/4/17
PAGE NO.:

GRAPH:

→ It's a non-linear data structure
→ Graph is tree with cycle
→ Graph can be represented by (V, E)
V → set of vertices
E → " edges



Undirected graph



Directed graph

Graph can be represented through two methods:

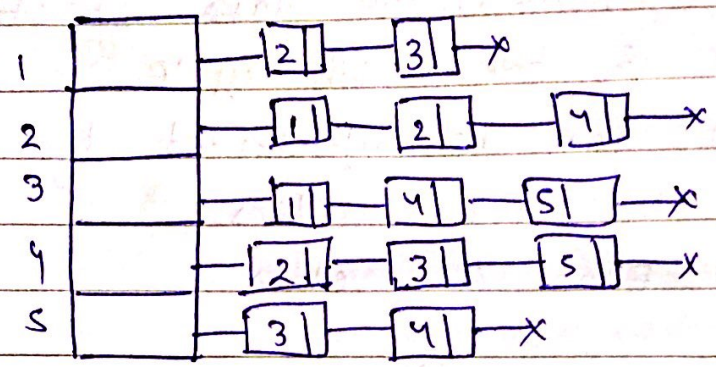
I. Adjacency Matrix: (VXV)

	1	2	3	4	5
1	0	1	1	0	0
2	1	1	0	1	0
3	1	0	0	1	1
4	0	1	1	0	1
5	0	0	1	1	0

If there is an edge b/w vertex i and vertex j then G[i][j] = 1 else entry is 0

Teacher's Signature

II. Adjacency list:



GRAPH TRAVERSAL:

I BFS (Breadth first search) - (Queue)

II DFS (Depth ") - (stack)

I. BFS (V)

{ visited [1:V] initialized to zero
insert_queue (v)
while (queue not empty)
{ w = delete queue()
if visited [w] == 0
{ print w ;
visited [w] = 1
insert all adjacent of w } }

→ w = delete_queue
visited [w] = 1
print w
ins k adj ' tow queue
if visited [w] = 0

II DFS - Just replace insert () by push()
and delete () by pop().

Teacher's Signature