Program Name : Masters of Computer Application

Semester : 2

Paper Title : Data Structures

Submitted by : Rishi Raj Chaurasia

Examination Roll Number : 19234757031

Department Roll Number : 31

DATA STRUCTURES ASSIGNMENTS:

1. Programming : Linked List
2. Heap
3. Hash Map & BST
4. AVL Tree
5. Programming : BST, Heap, AVL Tree

Assignment – 1

Q1.Single Linked List

Main Function :

```
int main()
{
    S_LinkedList<double> l;
    l.insert(1.8);
    l.insert(2.4);
    l.insert(3.2);
    l.insert(4.9);
    l.insert(5.1);
    l.insert(7.5);
    l.print();
    l.remove(0);
    l.print();
    l.reverse();
    l.print();
    l.swap(4.9);
    l.print();
    return 0;
}
```

Output :

```
1.8 -> 2.4 -> 3.2 -> 4.9 -> 5.1 -> 7.5 -> NULL
2.4 -> 3.2 -> 4.9 -> 5.1 -> 7.5 -> NULL
7.5 -> 5.1 -> 4.9 -> 3.2 -> 2.4 -> NULL
7.5 -> 4.9 -> 5.1 -> 3.2 -> 2.4 -> NULL
```

Q2. Double Linked List

Main Function:

```
int main()
{
    D_LinkedList<double> l;
    l.insert(1.8);
    l.insert(2.4);
    l.insert(3.2);
    l.insert(4.9);
    l.insert(5.1);
    l.insert(5.7);
    l.print();
    l.remove();
    l.print();
    l.reverse();
    l.print();
    return 0;
}
```

Output:

```
1.8 -> 2.4 -> 3.2 -> 4.9 -> 5.1 -> 5.7 -> NULL
1.8 -> 2.4 -> 3.2 -> 4.9 -> 5.1 -> NULL
5.1 -> 4.9 -> 3.2 -> 2.4 -> 1.8 -> NULL
```

Q3. Circular Linked List

Main Function :

```
int main()
{
    C_LinkedList<double> l;
    l.insert(1.8);
    l.insert(2.4);
    l.insert(3.2);
    l.insert(4.9);
    l.insert(5.1);
    l.insert(7.5);
    l.print();
    l.remove();
    l.print();
    cout << l.front() << "\n";
    cout << l.back() << "\n";
    l.advance();
    l.print();
    cout << l.front() << "\n";
    cout << l.back() << "\n";
    return 0;
}
```

Output :

```
1.8 -> 2.4 -> 3.2 -> 4.9 -> 5.1 -> 7.5 -> 1.8 -> 2.4 -> 3.2 ->
4.9 -> 5.1 -> 7.5 -> START
1.8 -> 2.4 -> 3.2 -> 4.9 -> 5.1 -> 1.8 -> 2.4 -> 3.2 -> 4.9 ->
5.1 -> START
1.8
5.1
2.4 -> 3.2 -> 4.9 -> 5.1 -> 1.8 -> 2.4 -> 3.2 -> 4.9 -> 5.1 ->
1.8 -> START
2.4
1.8
```

Q4. OddEven

Output :

```
Enter the size of array: 6
Enter the elements of the array: 1 3 6 8 5 2
1 3 5 8 6 2
```

Q5. Reverse

Output:

```
Enter the string: All is well
llew si llA
```

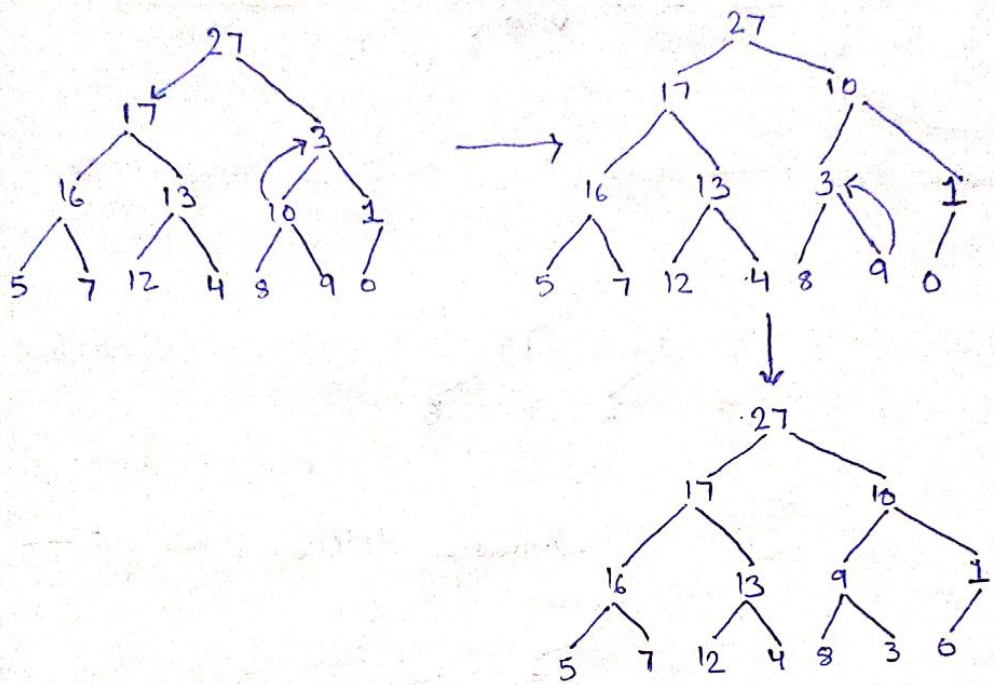
Assignment – 2

DS - Assignment

Rishi Raj Chaurasia
Roll No - 31

Q1) Max.Heap(A, 3), A = {27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9}

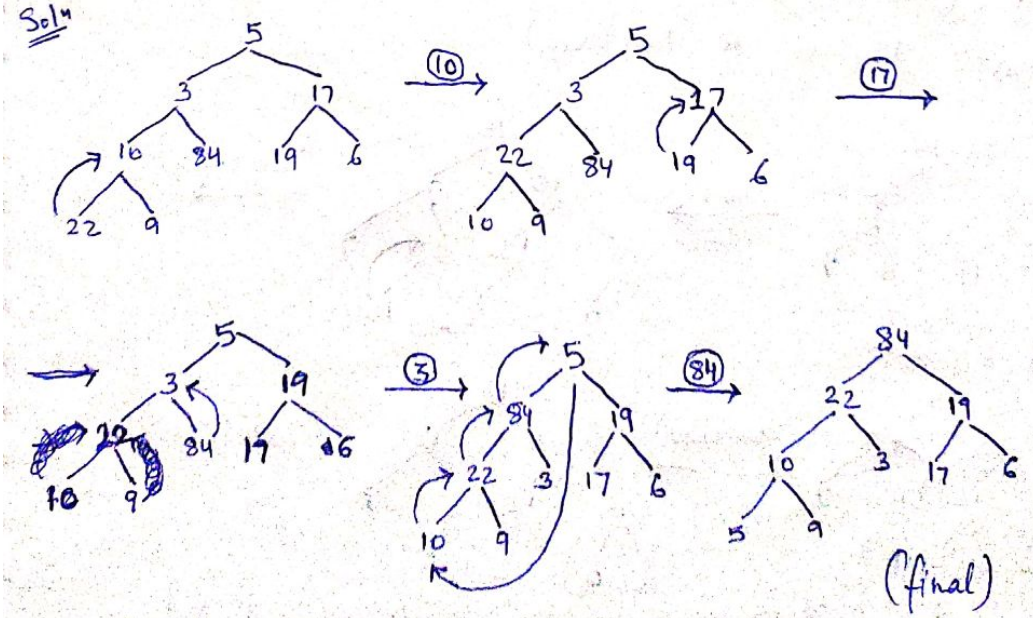
Soln



Q.2) Given Array = {5, 3, 17, 10, 84, 19, 6, 22, 9}

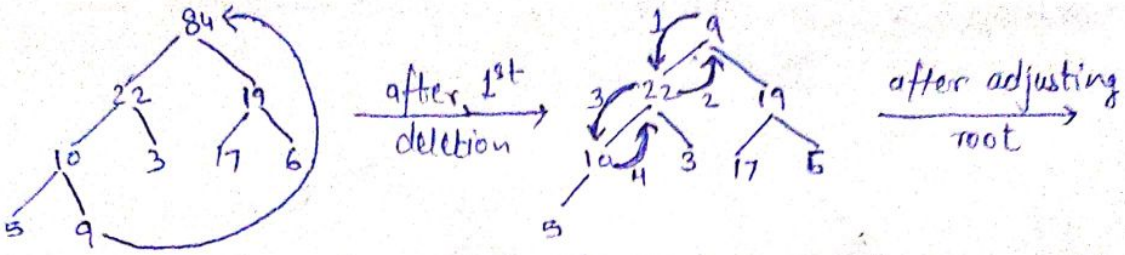
i) Build Max-Heap

Soln

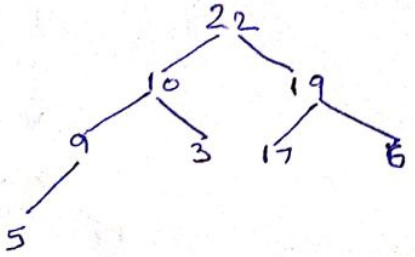


(ii) Heap after first four deletion.

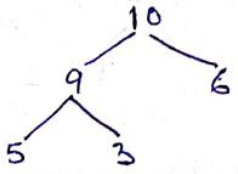
Soln



i.e. swap(9, 22) → Swap (9, 10) = final heap after 1st deletion.



Similarly after next 3 continuous deletion, we get —

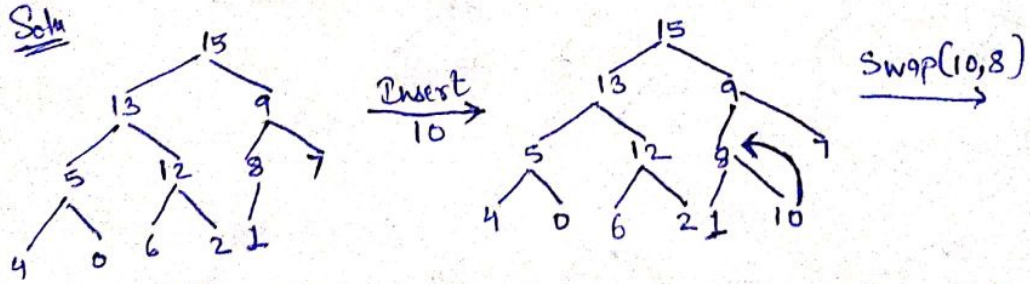


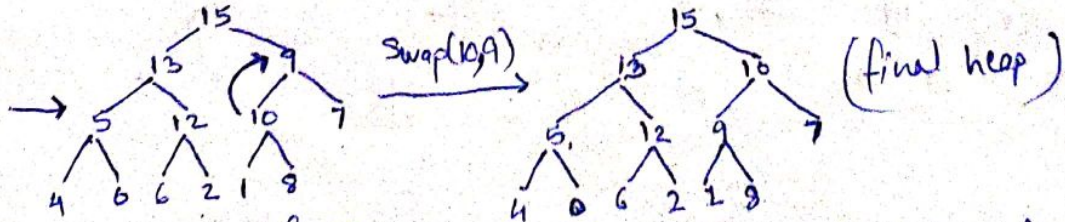
(final heap)

Q3) A = {15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1}

Insert 10 in A (MaxHeap)

Soln





∴ A = {15, 13, 10, 5, 12, 9, 7, 4, 0, 6, 2, 1, 8}

Q.4 a) Min-heapify operation

Soln

```
Min-heapify (A, n, i) {
  // A → Array of sequence
  // n → size of heap
  // i → index.
int smallest = i; int left = 2*i , right = 2*i+1;
if (left<=n && A[Smallest] > A[left]) smallest = left;
if (right<=n && A[smallest] > A[right] smallest = right
if(smallest != i) { int temp = A[smallest]; } Swap(A[i], A[smallest])
                    A[smallest] = A[i];
                    A[i] = temp;
                    min-heapify (A, n, smallest);
                  } }
```

b) Heap-extract-min operation (deletion of min element)

Soln

```
Min-delete (A, n) {
  // A = Array and n = size of heap.
  A[1] = A[n];
  --n;
Min_heapify(A, n, 1);
}
```

** Assuming that array indexing starts with 1.

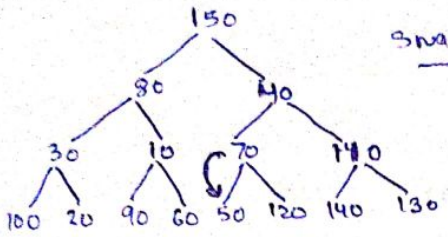
Q5) Array = {150, 80, 40, 30, 10, 70, 110, 100, 20, 90, 60, 50, 120, 140, 130}

Build Min-heap :-

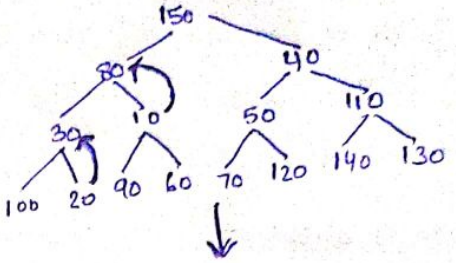
a) Min-heap for the above data :-

Soln

(Every node value must be less than or equal to its child nodes)



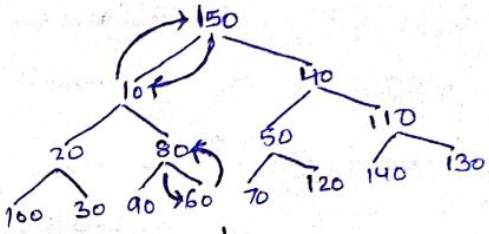
swap(70, 50) →

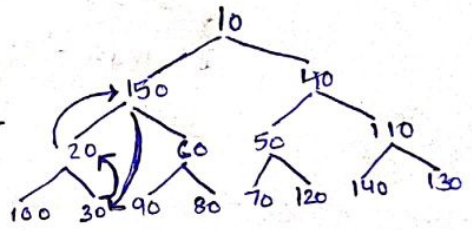


1st non-leaf Node = $\frac{n-1}{2}$

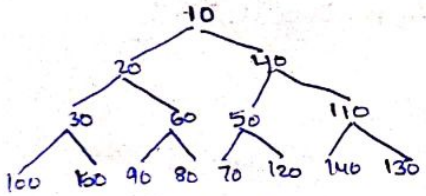
$= 7^{th} = 110$

110 – doesn't violates heap property





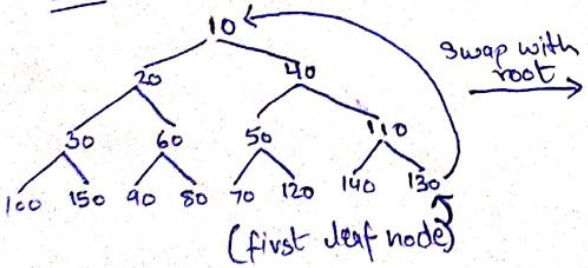
←



(final heap)

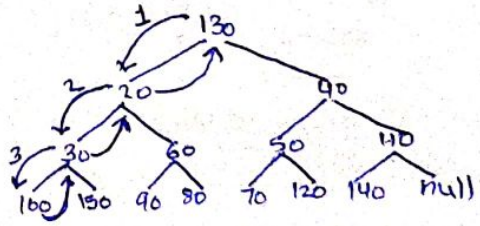
b) Heap after 1st 3 deletions :—

Soln



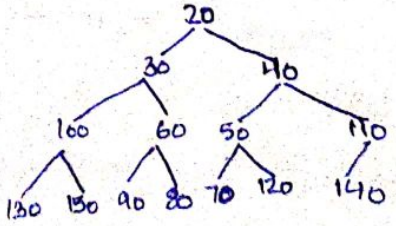
(first leaf node)

swap with root →

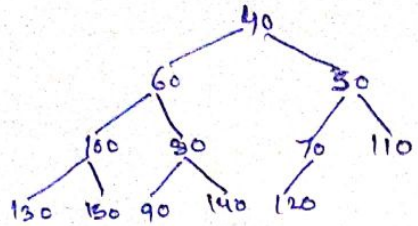


∴ After Min-heapifying :—

heap after 1st deletion →



∴ Similarly after 2 more deletion, heap will look like:—

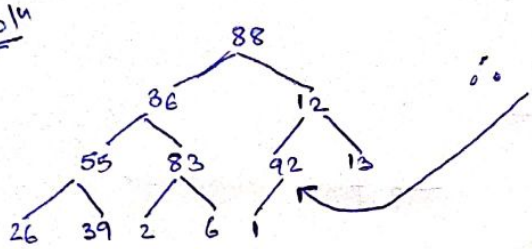


Array = {40, 60, 50, 100, 80, 70, 110, 130, 150, 90, 140, 120}

Q.6) Array = {88, 36, 12, 55, 83, 92, 13, 26, 39, 2, 6, 1}

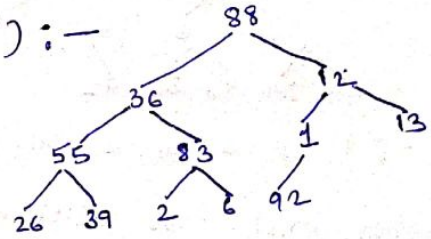
a) Build Min-Heap. —

Soln

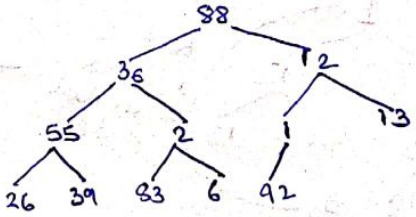


∴ First Non-leaf Node = $\left[\frac{n-1}{2}\right]$ = 6th = 92

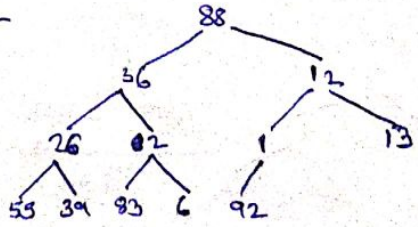
1.) Heapify 92 :- Swap (92, 1) :—



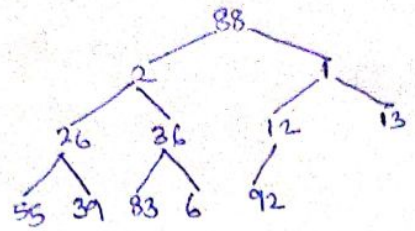
2.) Heapify 83 :- Swap(83, 2) :—



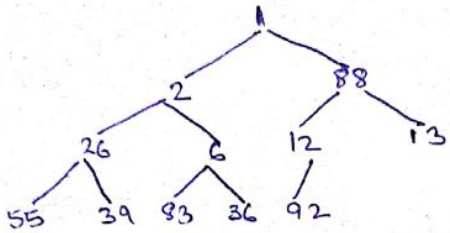
3.) Heapify 55 – Swap (55, 26) :—



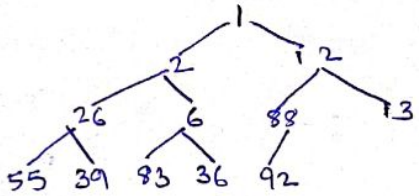
4) Heapify 12 and 36 — Swap (12,1) and Swap (36, 2) :—



5) Heapify 36 and 88 — Swap (36,6) and Swap (88, 1) :—



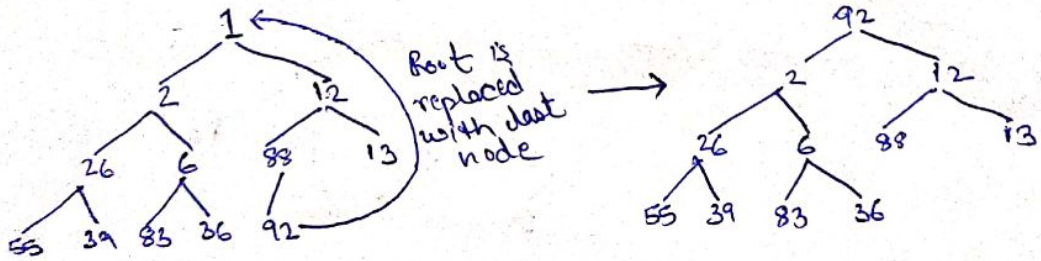
6) Heapify 88 — Swap (88, 12) :—



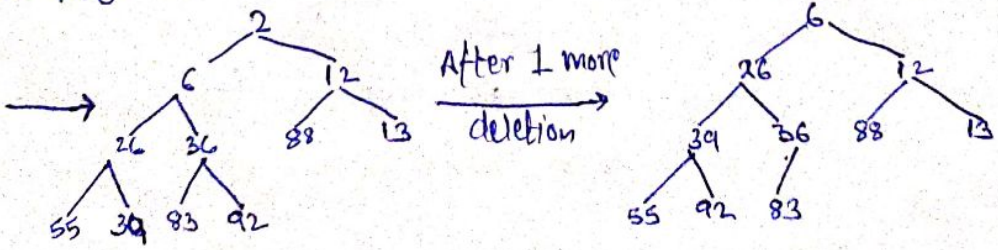
→ final Min-heap.

Array = { 1, 2, 12, 26, 6, 88, 13, 55, 39, 83, 36, 92 }

b) heap after first two deletion :—



Heapify (92)



∴ Array = { 6, 26, 12, 39, 36, 88, 13, 55, 92, 83 }

Assignment – 3

DS – Assignment

Rishi Chaurasia
Roll no. 31

Q.1) keys :- 1, 3, 12, 4, 25, 6, 18, 20, 8 are

N = 10

H(i) = i^2 mod N

Linear Probing.

Soln

Insert Keys	Probe Value
1^2 → 1 mod 10 = 1	1
3 → 3^2 mod 10 = 9	1
12 → 12^2 mod 10 = 4	1
4 → 4^2 mod 10 = 6	1
25 → 25^2 mod 10 = 5	1
6 → 6^2 mod 10 = 6	2
→ (6^2+1) mod 10 = 7	
18 → 18^2 mod 10 = 4	5
→ (18^2+1) mod 10 = 5	
→ (18^2+2) mod 10 = 6	
→ (18^2+3) mod 10 = 7	
→ (18^2+4) mod 10 = 8	
20 → 20^2 mod 10 = 0	1
8 → 8^2 mod 10 = 4	9
(8^2+1) mod 10 = 5	
(8^2+2) mod 10 = 6	
(8^2+3) mod 10 = 7	
(8^2+4) mod 10 = 8	
(8^2+5) mod 10 = 9	
(8^2+6) mod 10 = 0	
(8^2+7) mod 10 = 1	
(8^2+8) mod 10 = 2	

0	20
1	1
2	8
3	3
4	12
5	25
6	4
7	6
8	18
9	3

Max Probe Value = 9

Ques 2 Key set :- 23, 91, 52, 40, 50, 60, 62

$H1(K) = K \bmod m$ $\quad m = 10$

$H2(K) = 1 + (K \bmod (m-2))$

Double Hash func^n

$DH(k,i) = (H1(K) + H2(K)) \bmod m$,

where $i = 0, 1, 2, \ldots, m-1$

Soln

$H_1(23) = 23 \bmod 10 = 3$

$H_1(91) = 91 \bmod 10 = 1$

$H_1(52) = 52 \bmod 10 = 2$

$H_1(40) = 40 \bmod 10 = 0$

$H_1(50) = 50 \bmod 10 = 0 \rightarrow$ Collision

$H_2(50) = 1 + (50 \bmod (8)) = 1 + 2 = 3$

$DH(50,1) = 0 + (3 \times 1) = 3 \rightarrow$ Collision

$DH(50,2) = 0 + (3 \times 2) = 6$

$H_1(60) = 60 \bmod 10 = 0 \rightarrow$ Collision

$H_2(60) = 1 + (60 \bmod 8) = 5$

$DH(60,1) = 0 + (5 \times 1) = 5$

$H_1(62) = 62 \bmod 10 = 2 \rightarrow$ collision

$H_2(62) = 1 + (62 \bmod 8) = 7$

$DH(62,1) = 2 + (7 \times 1) = 9$

40	0
91	1
52	2
23	3
	4
60	5
50	6
	7
	8
62	9

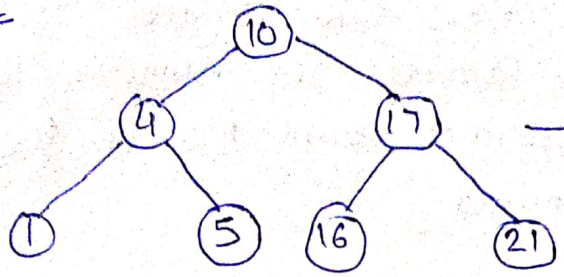
* Collision occurs 4 times.

Ques 3 For the set of keys

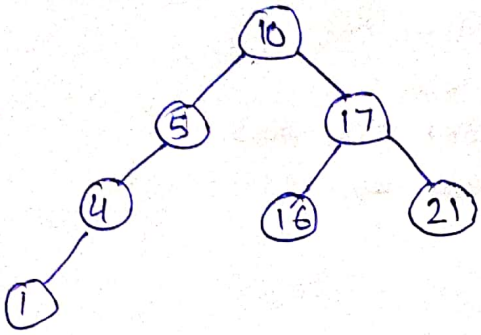
{1, 4, 5, 10, 16, 17, 21}

Draw BST of height 2, 3, 4, 5, 6

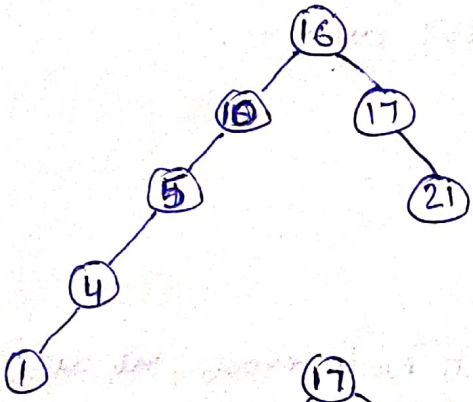
Soln



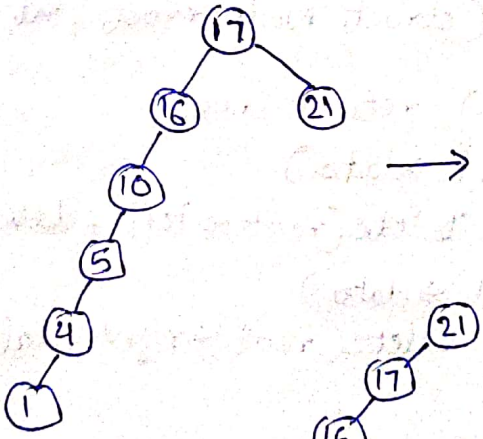
→ BST of height = 2



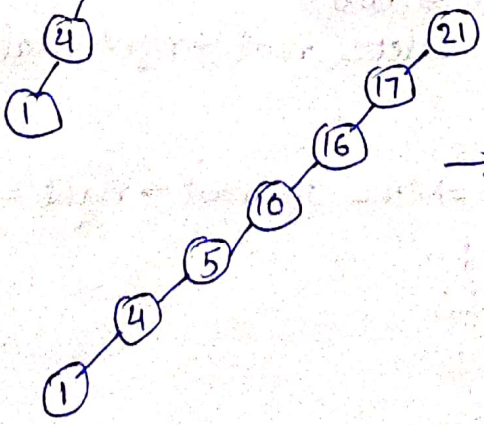
→ BST of height = 3



→ BST of height = 4



→ BST of height = 5



→ BST of height = 6

Ques4 Suppose we have nos. b/w 1 and 1000 in BST and want to search for number 363. Which of the following sequence could not be the sequence of nodes examined ?

a) 2, 252, 401, 398, 330, 344, 397, 363

b) 924, 220, 911, 244, 898, 258, 362, 363

c) 925, 202, 911, 240, 912, 245, 363

d) 2, 399, 387, 219, 266, 382, 381, 278, 363

e) 935, 278, 347, 621, 299, 392, 358, 363

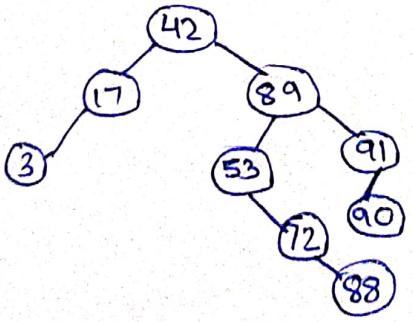
Soln (c), (e)

Ques5 Write Pseudocode for BST deletion.

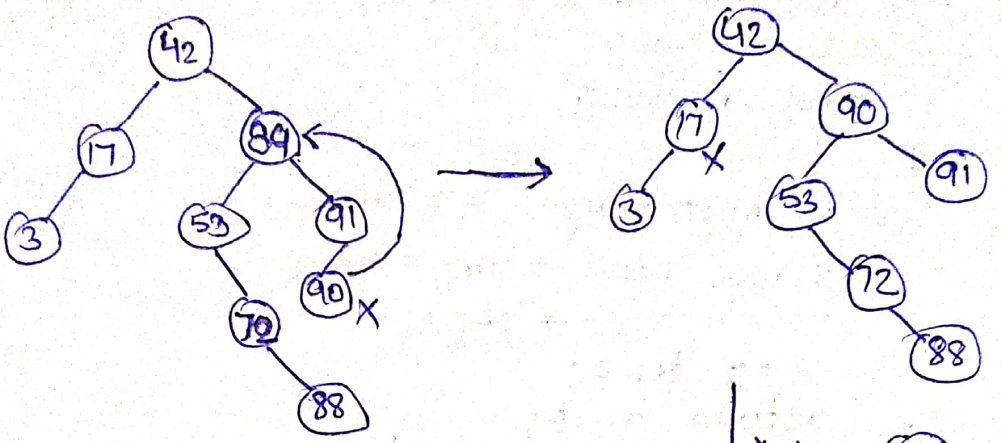
Soln
```
struct Node {
    int data;
    struct Node *left;
    struct Node *right;
};
struct Node* Delete (struct Node *root, int data)
{
  if(root == NULL) return root;
  else if(data < root->data)
      root->left = Delete(root->left, data)
  else if(data > root->data)
      root->right = Delete(root->right, data)
  else
  {
    if(root->left ==NULL && root->right ==NULL)
    { delete root;
      root = NULL;
      return root;
    }
```

```
else if(root->left == NULL) {
    struct Node *temp = root;
    root = root->right;
    delete temp;
    return root;
}
else if(root->right == NULL) {
    struct Node *temp = root;
    root = root->left;
    delete temp;
    return root;
}
else {
    struct Node *temp = FindMin(root->right);
    root->data = temp->data;
    root->right = Delete(root->right, temp->data);
}
}
return root;
}
```

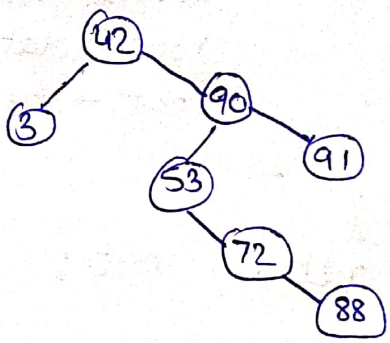
Ques 6 Draw the BST following each insertion from the following data values in the order given 42, 17, 89, 53, 72, 91, 3, 88, 90
From that BST delete 89 and then 17.



Delete 89



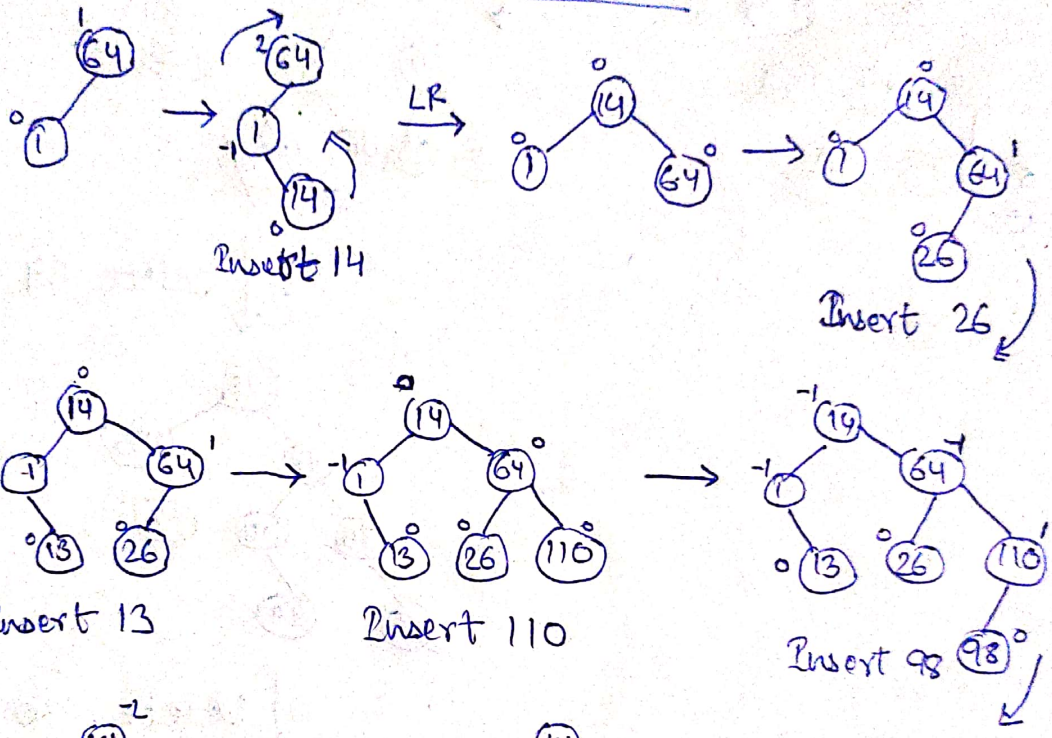
Delete 17

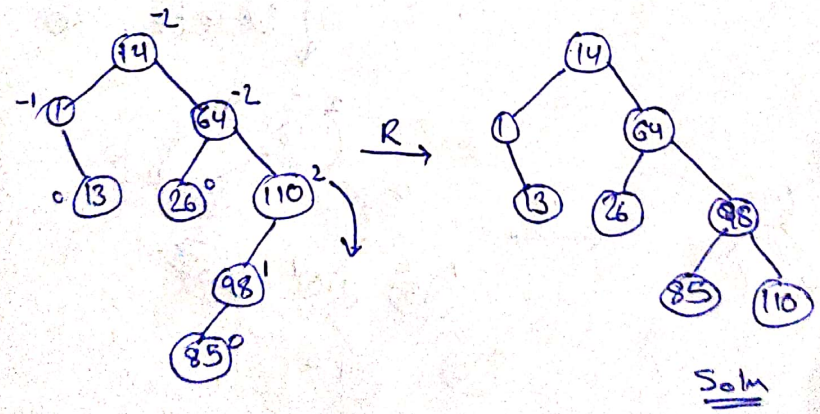


DS ASSIGNMENT-4

Q:1) Soln

64, 1, 14, 26, 13, 110, 98, 85

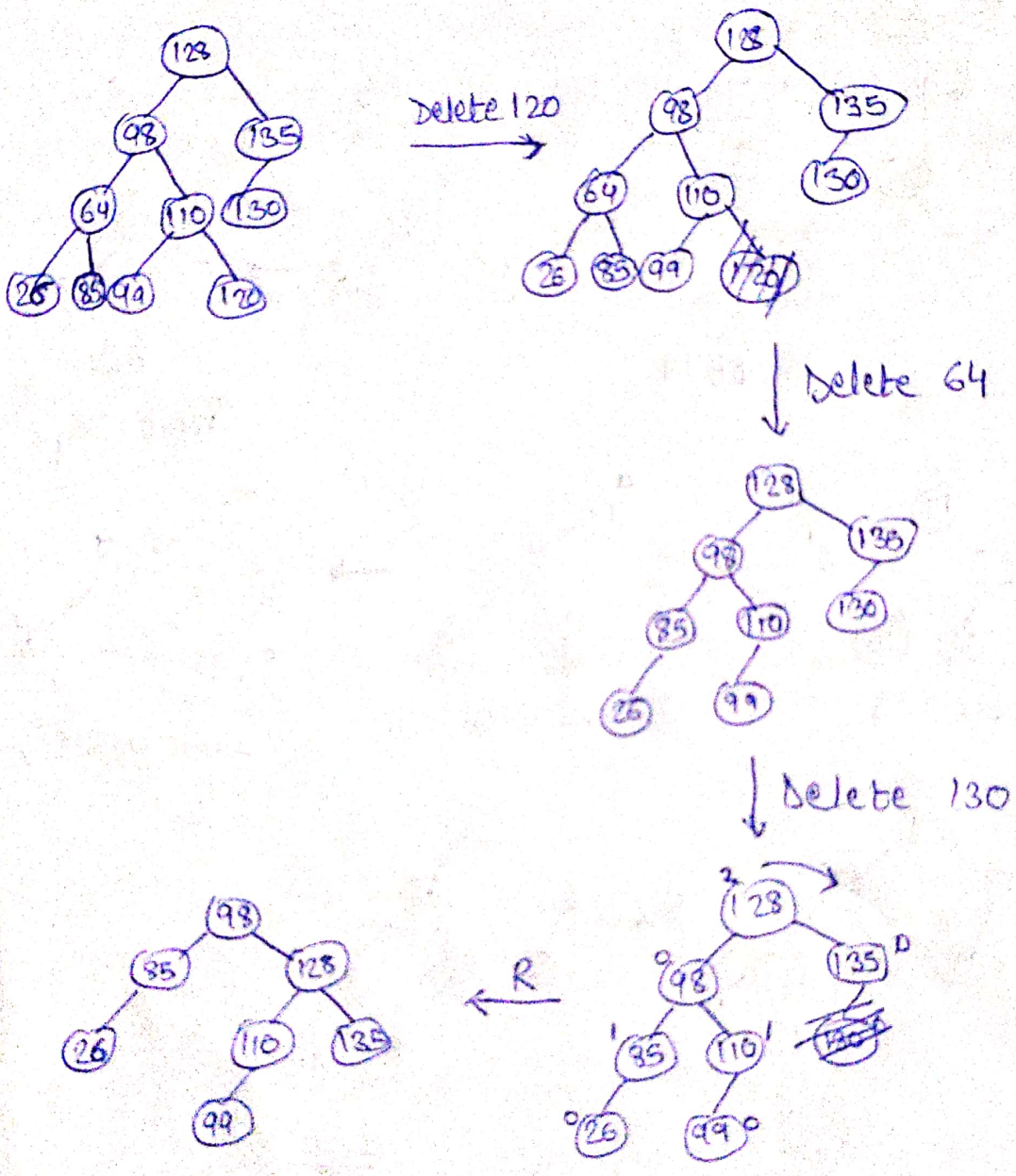




Soln

Q.2)
Soln

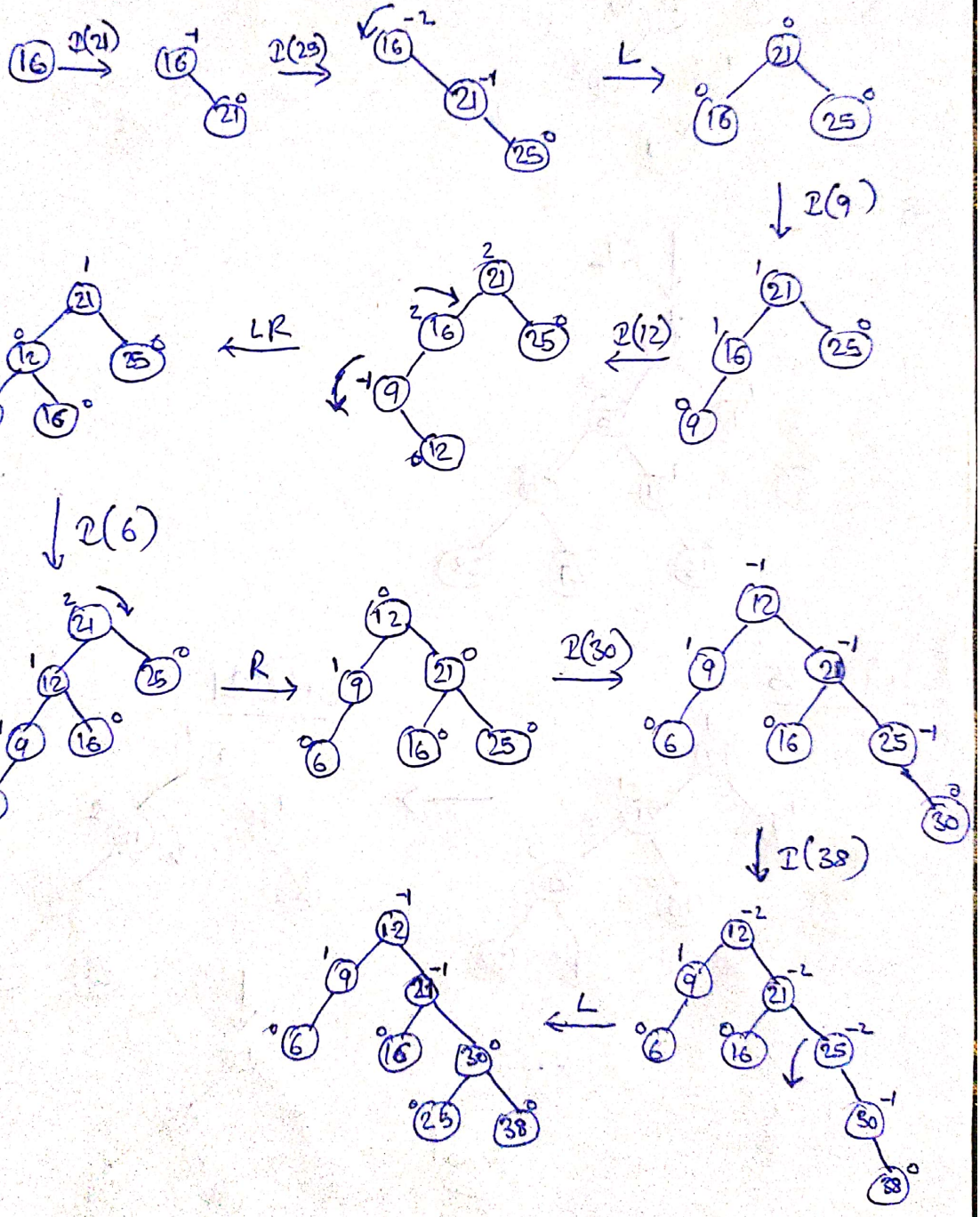
To Delete 120, 64 & 130



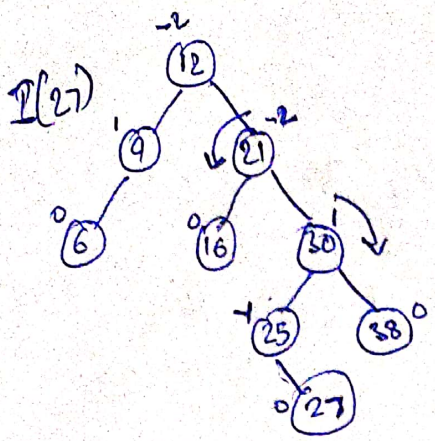
Q.3)

Solu Build AVL Tree

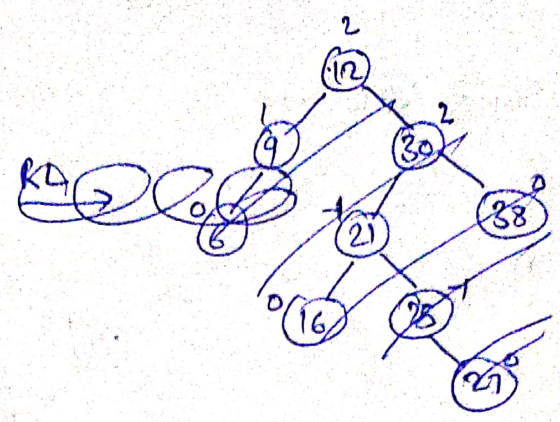
16, 21, 25, 9, 12, 6, 30, 38, 27



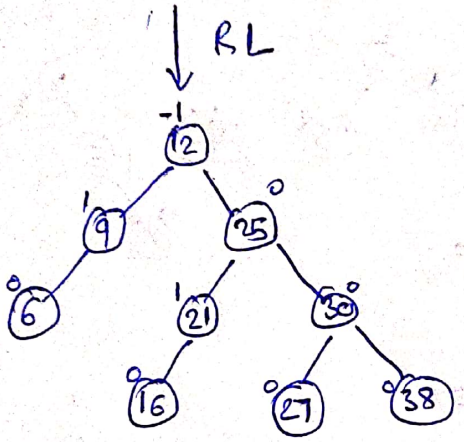
I(27)



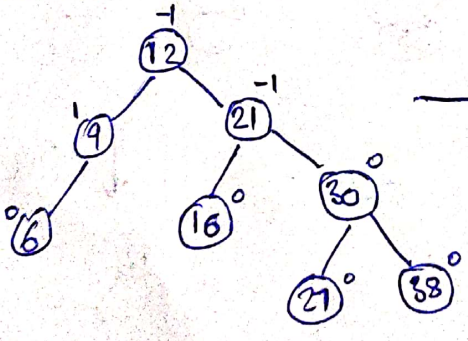
RL



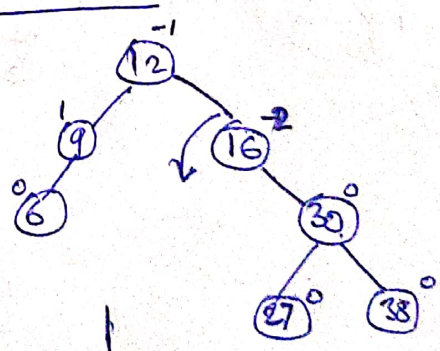
RL



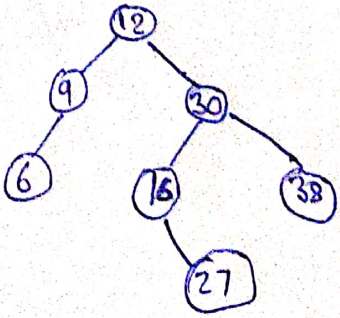
Delete 25



Delete 21



Final



Assignment – 5

Q1. WAP to print postorder traversal of a binary tree from given inorder and preorder traversals.

Main Function :

```
int main() {
     int in[] = {4, 2, 5, 1, 6, 3, 7};
     int pre[] = {1, 2, 4, 5, 3, 6, 7};
     int n = sizeof(in)/sizeof(in[0]);
     printPostorder(in, pre, 0, n-1);
     return 0;
}
```

Output :

```
4 5 2 6 7 3 1
```

Q2. WAP to print preorder traversal of a binary tree from given inorder and postorder traversals.

Main Function :

```
int main() {
     int in[] = {4, 2, 5, 1, 6, 3, 7};
     int post[] = {4, 5, 2, 6, 7, 3, 1};
     int n = sizeof(in)/sizeof(in[0]);
     cout << "Postorder : ";
     printPreorder(in, post, 0, n-1, n-1);
     return 0;
}
```

Output :

```
Postorder : 1 2 4 5 3 6 7
```

Q3. WAP to create Binary Search Tree and perform insertion and deletion in it.

Main Function :

```
int main()
{
    BST b;
    b.insert(b.root, 50);
```

```
    b.insert(b.root, 30);
    b.insert(b.root, 20);
    b.insert(b.root, 40);
    b.insert(b.root, 70);
    b.insert(b.root, 60);
    b.insert(b.root, 80);

    b.display(b.root);
    cout << endl;
    b.deleteNode(b.root, 50);
    b.display(b.root);
    return 0;
}
```

Output :

```
20 30 40 50 60 70 80
20 30 40 60 70 80
```

Q4. WAP to implement heapsort(max heap).

Main Function :

```
int main()
{
    int arr[] = {54, 21, 13, 76, 3, 11, 33};
    int n = sizeof(arr)/sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is : ";
    printArray(arr, n);
}
```

Output :

```
Sorted array is : 3 11 13 21 33 54 76
```

Q5. WAP to implement AVL tree.

```
Menu::
1.Insert Element.
2.Show AVL Tree.
3.Print InOrder traversal.
4.Print PreOrder traversal.
5.Print PostOrder traversal.
6.Exit

Enter your Choice: 1
```

```
Enter value to be inserted: 34  6

Menu::
1.Insert Element.
2.Show AVL Tree.
3.Print InOrder traversal.
4.Print PreOrder traversal.
5.Print PostOrder traversal.
6.Exit

Enter your Choice: 1

Enter value to be inserted: 45

Menu::
1.Insert Element.
2.Show AVL Tree.
3.Print InOrder traversal.
4.Print PreOrder traversal.
5.Print PostOrder traversal.
6.Exit

Enter your Choice: 1

Enter value to be inserted: 36
Left-Left RotationRight-Left RotationRight-Right Rotation
Menu::
1.Insert Element.
2.Show AVL Tree.
3.Print InOrder traversal.
4.Print PreOrder traversal.
5.Print PostOrder traversal.
6.Exit

Enter your Choice: 1

Enter value to be inserted: 67

Menu::
1.Insert Element.
2.Show AVL Tree.
3.Print InOrder traversal.
4.Print PreOrder traversal.
5.Print PostOrder traversal.
6.Exit

Enter your Choice: 1

Enter value to be inserted: 23

Menu::
1.Insert Element.
```

```
2.Show AVL Tree.
3.Print InOrder traversal.
4.Print PreOrder traversal.
5.Print PostOrder traversal.
6.Exit

Enter your Choice: 2

AVL Tree:

                        67
                45
Root -> 36
                        23
                6

Menu::
1.Insert Element.
2.Show AVL Tree.
3.Print InOrder traversal.
4.Print PreOrder traversal.
5.Print PostOrder traversal.
6.Exit

Enter your Choice: 3

Inorder Traversal:
6 23 36 45 67

Menu::
1.Insert Element.
2.Show AVL Tree.
3.Print InOrder traversal.
4.Print PreOrder traversal.
5.Print PostOrder traversal.
6.Exit

Enter your Choice: 6
```

----------x----------