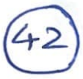
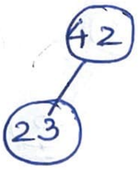
Q1: Way 1:

a)

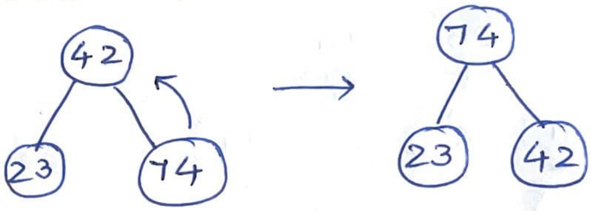
add 42



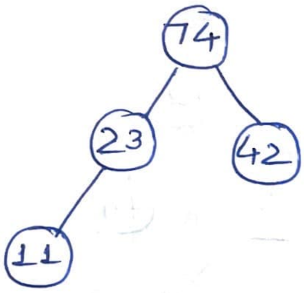
add 23



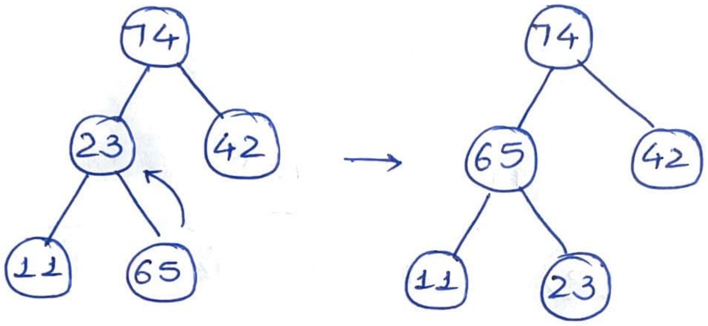
add 74



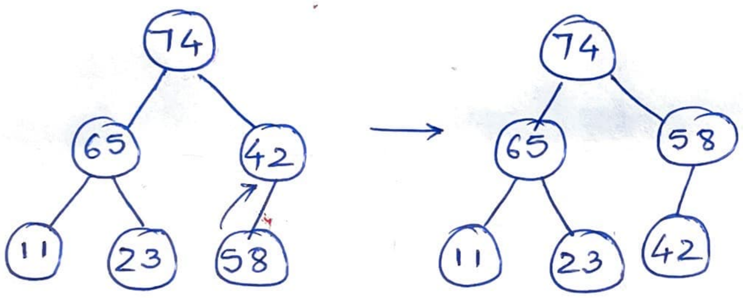
add 11



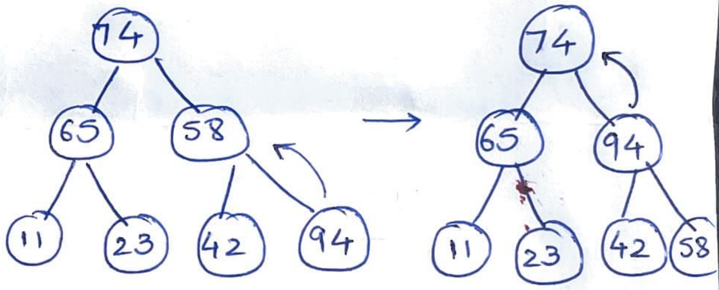
add 65

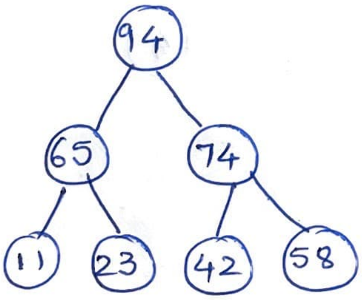


add 58

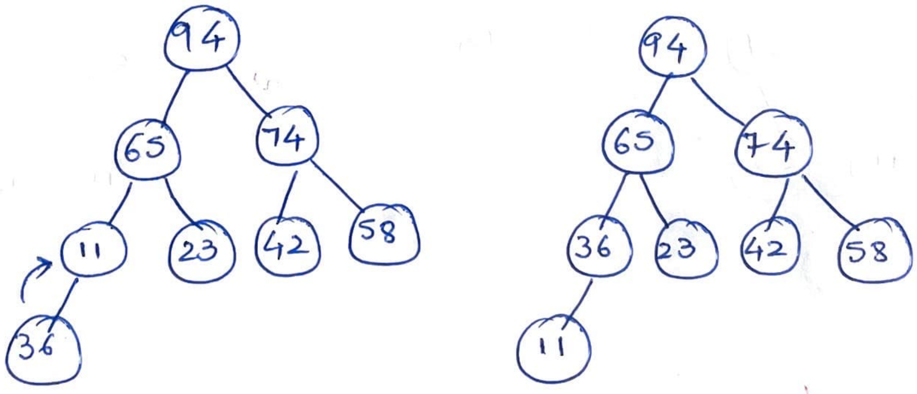


add 94

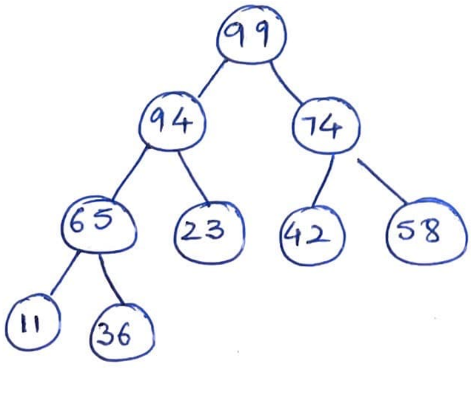




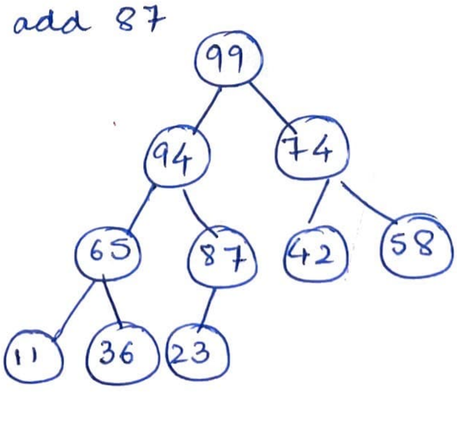
add 36



add 99

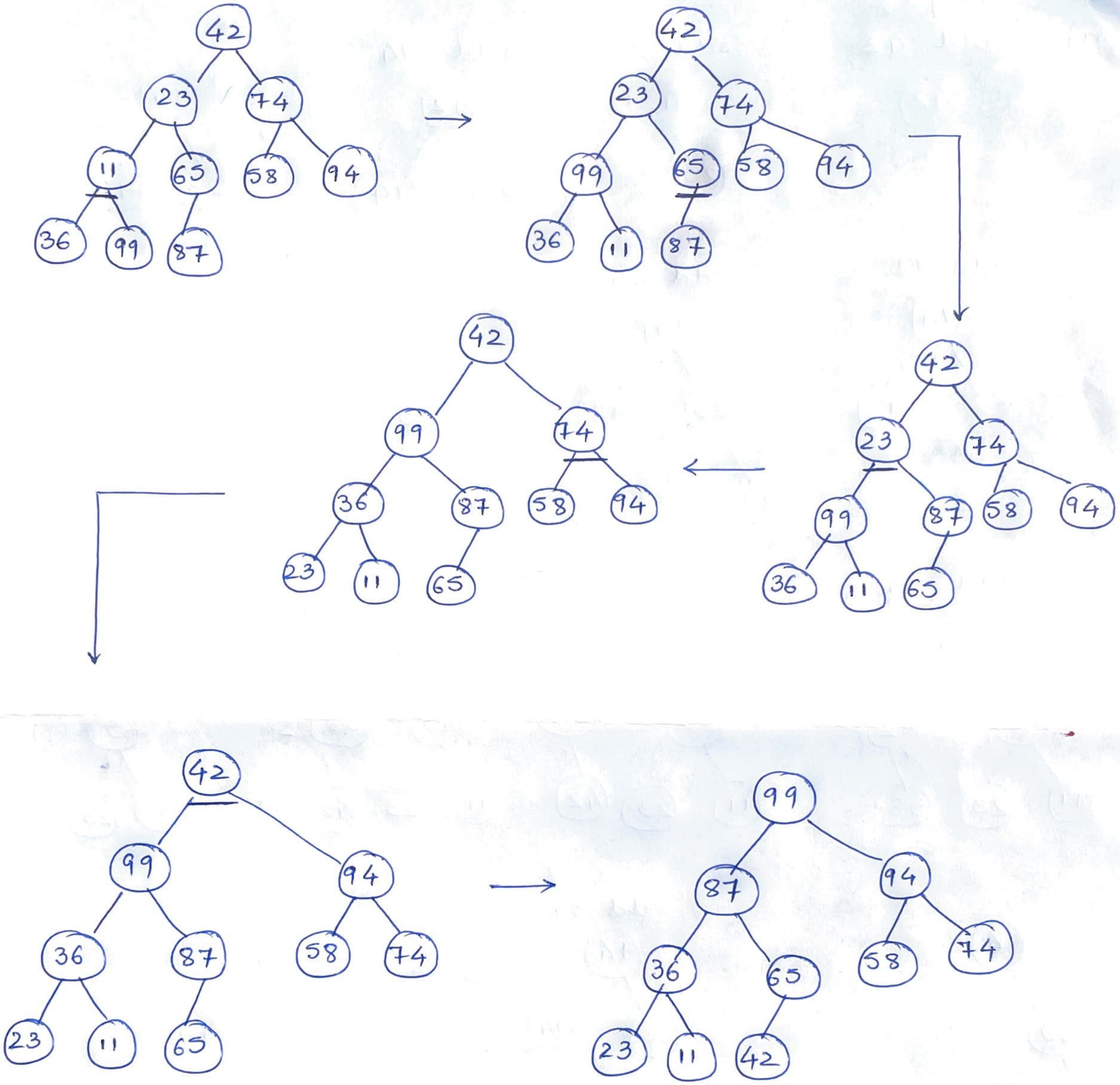


add 87



Day 2:

42, 23, 74, 11, 65, 58, 94, 36, 99, 87



Q1

a)

```
vector <int> data ;                    Way 1:

void add (int item)
{

    data.push_back (item);
    upheapify (data.size()-1);
}

void upheapify (int ci)
{
    int pi = (ci-1)/2;

    if (data[ci] > data[pi])
    {
        swap (data[pi], data[ci]);
        upheapify (pi);

    }
}
```

Way 2:

```
    int *data;
    int N;

    void add (int *arr, int N)
    {
        this -> data = arr;
        this -> N    = N;

        for (int i = N/2-1; i>=0; i--)
                downheapify(i);

    }
    void downheapify(int pi)
    {
        int mini = pi;
        int lci = 2*pi+1, rci = 2*pi+2;
        if (lci<N && data[lci] > data[mini])        mini = lci;
        if (rci<N && data[rci] > data[mini])        mini = rci;

        if (mini != pi)
        {
            swap(data[pi], data[mini]);
            downheapify(mini);
        }
  }
```

Q1

b)

```
Node* mergeTWOLL(Node * node1, Node *node 2)
{

    Node * dummy = new Node(-1);
    Node *tail = dummy;

    while(node1 != NULL  && node 2 != NULL)
    {
        if(node1→data <= node 2→data)
        {
            tail→next = node 1;
            node1 = node 1 → next;
        }
        else
        {
            tail→next = node 2;
            node 2 = node 2 → next;
        }
        tail = tail → next;
    }
    if(node1 != NULL)
        tail→next = node1;
    else
        tail→next = node 2;

    return dummy→next;
}
```

Time complexity = $O(\underbrace{m+n}_{\text{Size of 2 LL}})$

Q2:

a)

```
void printInRange(Node *node, int a, int b)
{
    if (node == NULL)
        return;

    if (node → data >= a && node → data <= b)
    {
        printInRange(node → left, a, b);
        cout << node → data ;
        printInRange(node → right, a, b);
    }

    else if (node → data < a)
        printInRange(node → right, a, b);

    else if (node → data > b)
        printInRange(node → left, a, b);
}
```

b)

```
bool Identical (Node *node1, Node *node2)
{
    if(node1 == NULL && node2 == NULL)  return true;
    if (node1 ~~&~~== NULL || node2 == NULL)  return false;

    return  node1 → data == node2 → data  &&
            Identical (node1 → left, node2 → left) &&
            Identical (node1 → right, node2 → right);
}
```

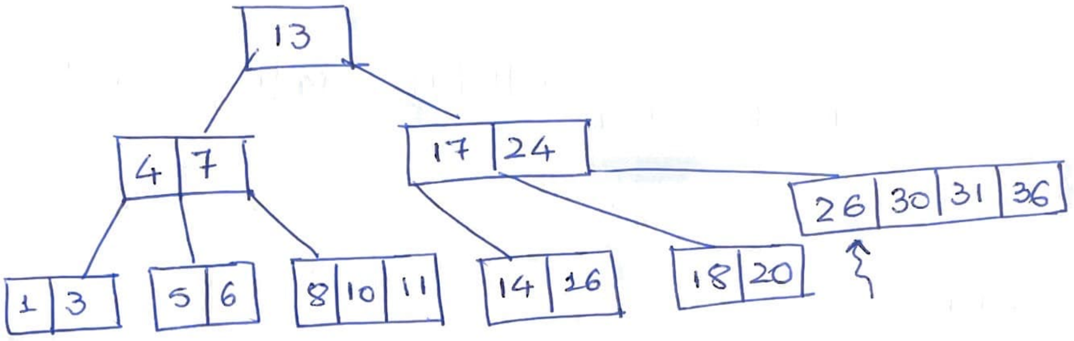
Q3:

Order = 6 ⇒ atmost 6 children

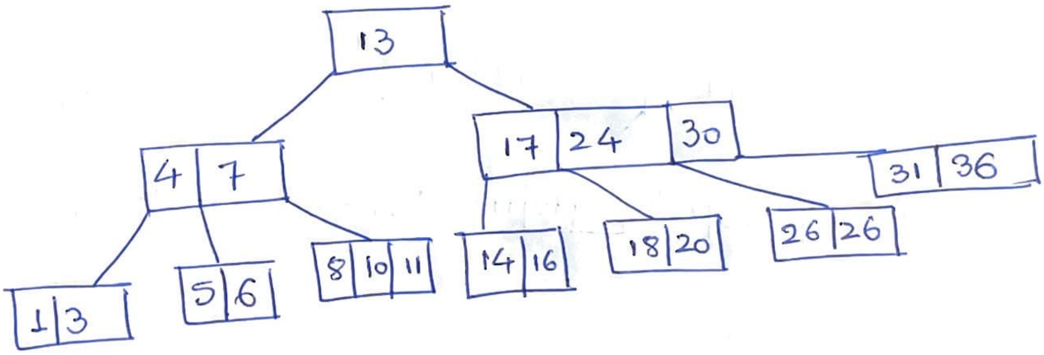
max keys = 6 − 1 = 5

min keys = $\lceil \frac{6}{2} \rceil - 1 = 2$

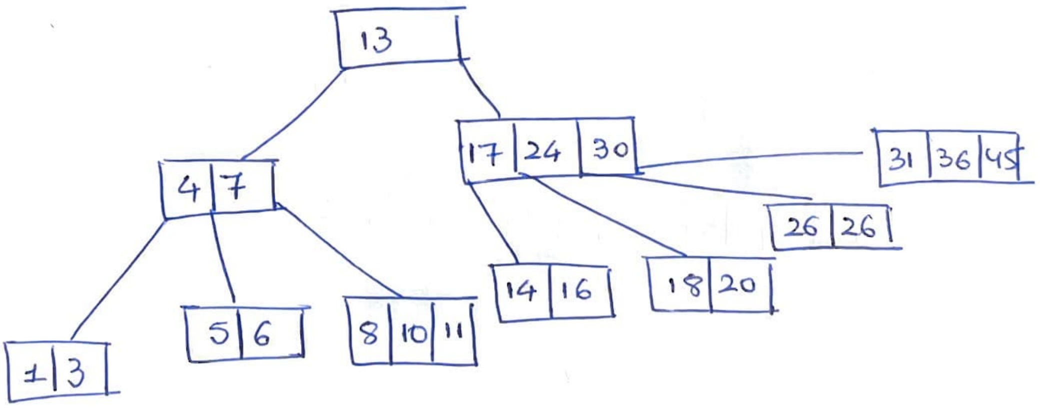
i) insert 26



⇓ insert 26



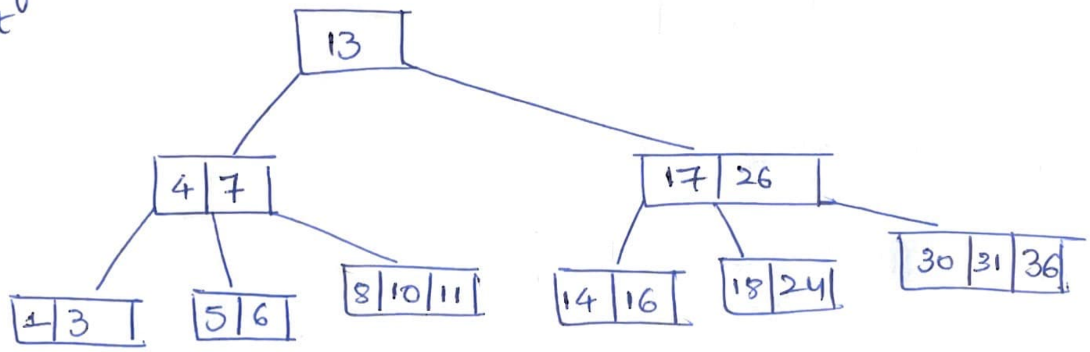
⇓ insert 45



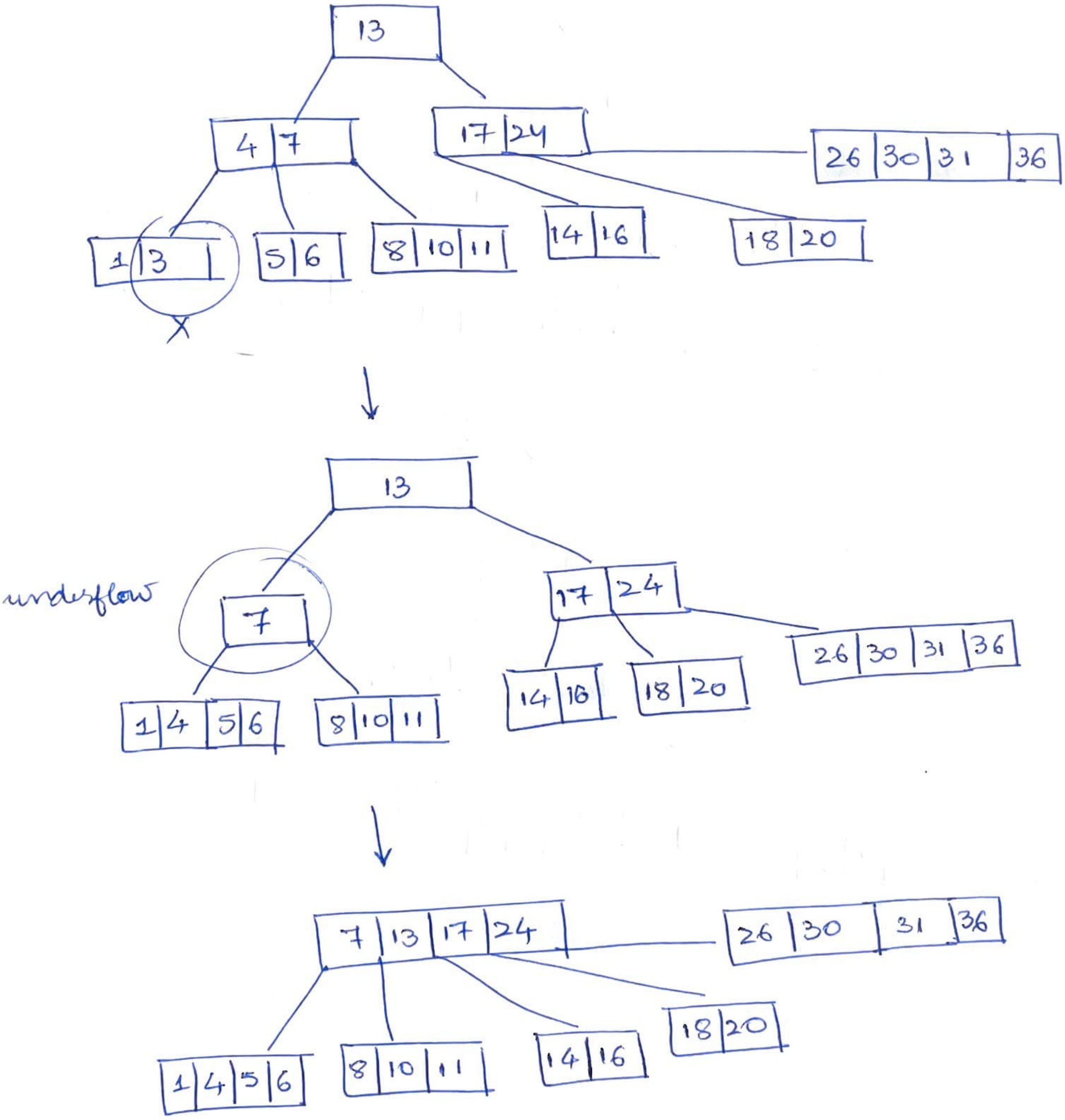
ii) delete 20

Cant borrow from left ~~[illegible] [illegible]~~

borrow from right



iii) delete 3



Q4 a)

~~$h_1(79) = 1$~~ ~~$h_2(79) = 2$~~

~~$h_1(69) = 4$~~ ~~h_2~~

$h1(79) = 79 \bmod 13 = \underline{1}$

$h1(69) = 69 \bmod 13 : \underline{4}$

$h1(98) = 98 \bmod 13 = \underline{7}$

$h1(85) = 85 \bmod 13 = \underline{7}$ collision

$h_2(85) = 1 + 85 \bmod 11 = 9$

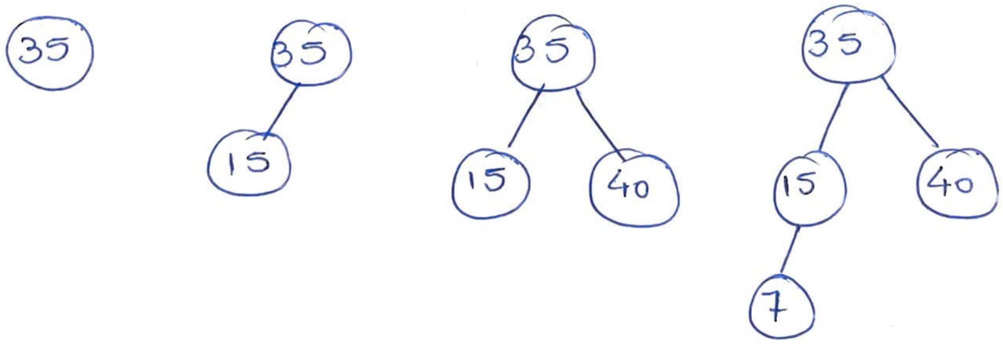
$h_1(27) = 27 \bmod 13 = \underline{1}$ collision

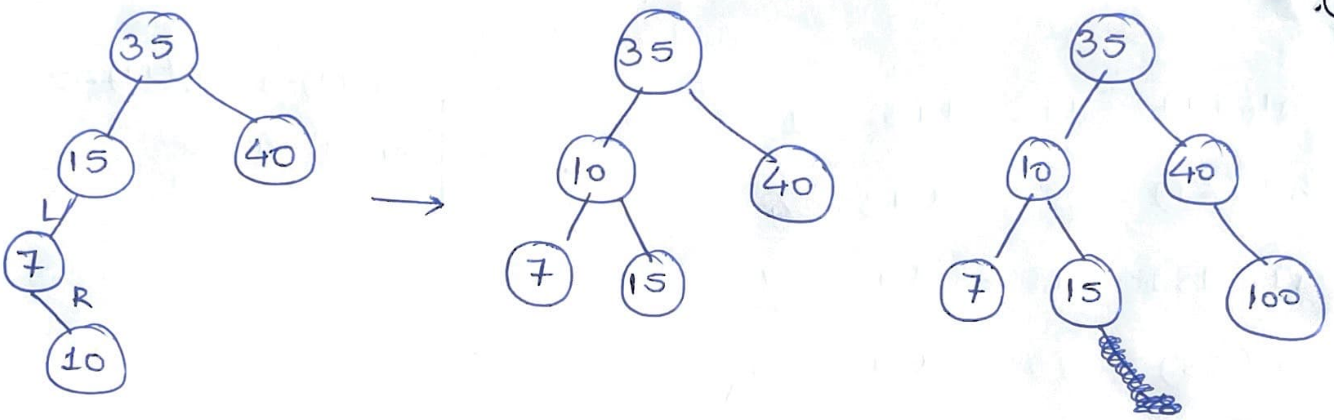
$h_2(27) = 1 + 27 \bmod 11 = 6$

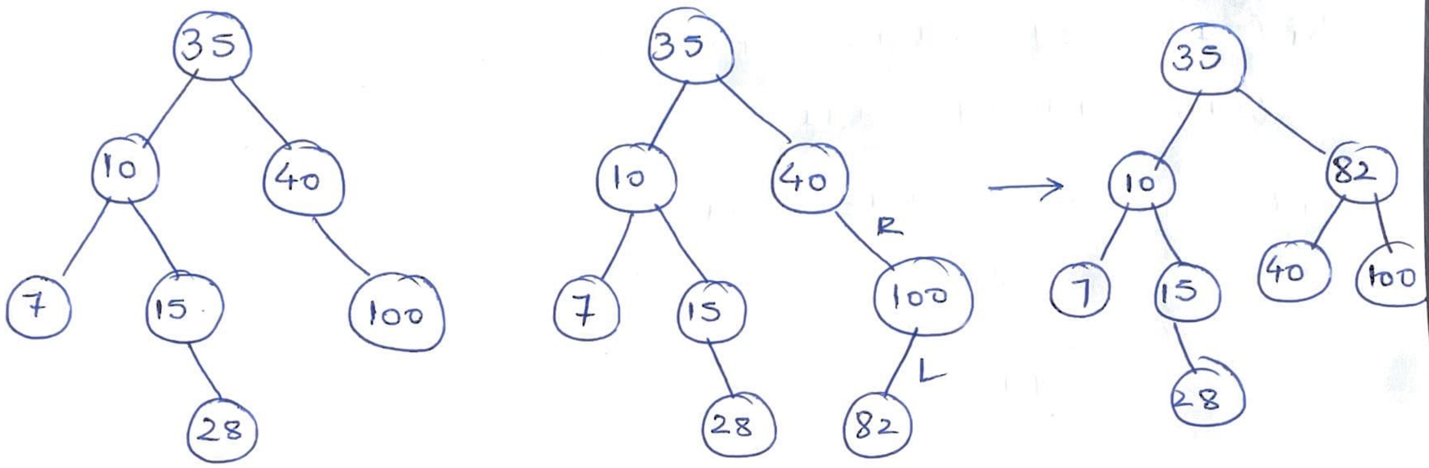
$h_1(50) = 50 \bmod 13 = \underline{11}$

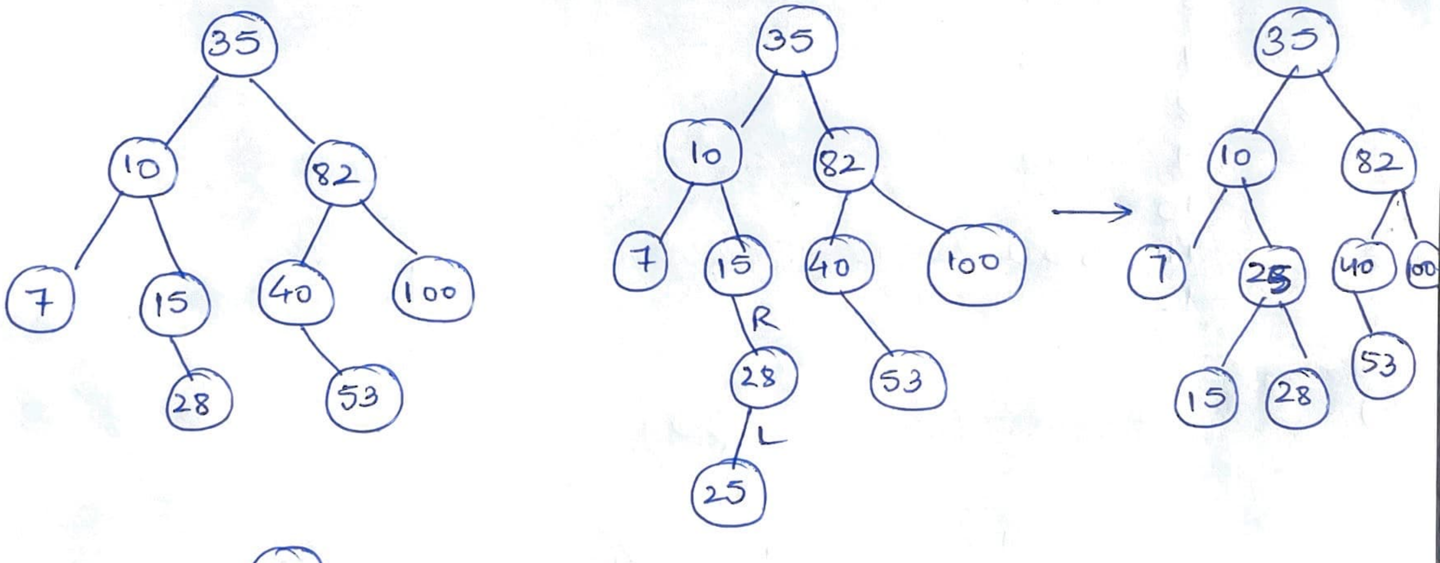
0
1 → 79
2
3
4 → 69
5
6 → 27
7 → 98
8
9 → 85
10
11 → 50
12

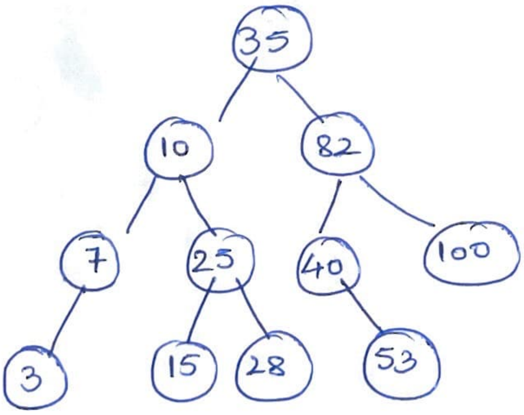
b)




35
15
40
L
7
R
10
35
10
40
7
15
35
10
40
7
15
100


35
10
40
7
15
100
28
35
10
40
R
7
15
100
L
28
82
35
10
82
7
15
40
100
28


35
10
82
7
15
40
100
28
53
35
10
82
7
15
40
100
R
28
53
L
25
35
10
82
7
25
40
100
15
28
53


35
10
82
7
25
40
100
3
15
28
53

Q5:

a) isTree → isconnected
→ no cycle.

b) BFS :

Remove
Mark visited
Printing
Nbrs.

Q6:

```
class MinHeap
{
    int *data;
    int cap;
    int N;

             int cap
    MinHeap (~~int data~~)
    {
        ~~this→data=data;~~
        data= new int[cap];
        this→cap= cap;
        this→ N=0;
    }
    int capacity()
    {
        return cap;
    }
    bool isEmpty()
    {
        return N==0;
    }
    bool isFull()
    {
        return N==cap;
    }
```

```
void enqueue (int item)
{
    data[N] = item;
    N++ ;
    upheapify (N-1);
}

void upheapify(int ci)
{
    int pi= (ci-1)/2;
    if (data[ci] < data[pi])
    {
        swap (data[pi], data [ci]);
        upheapify (pi);
    }
}

void dequeue()
{
    swap (data[0], data[N-1]);
    N--;
    downheapify(0);
}

void downheapify (int pi)
{
    int mini = pi;
    int lci = 2*pi+1;
    int rci = 2* pi+2;

    if (lci < N && data[lci] < data[mini])  mini = lci;
    if (rci < N  && data [rci] < data [mini])  mini = rci;

    if (mini != pi)
    {
        swap(data [pi], data[mini])
        down heapify (mini);
    }
}
```

```
void makeEmpty()
{
    N=0;
}
~~void~~ int minvalue()
  {
      return data[0];
  }

void decrease Value (i,delta)
{
    data[i]=data[i]-delta;
    upheapify(i);
}
```