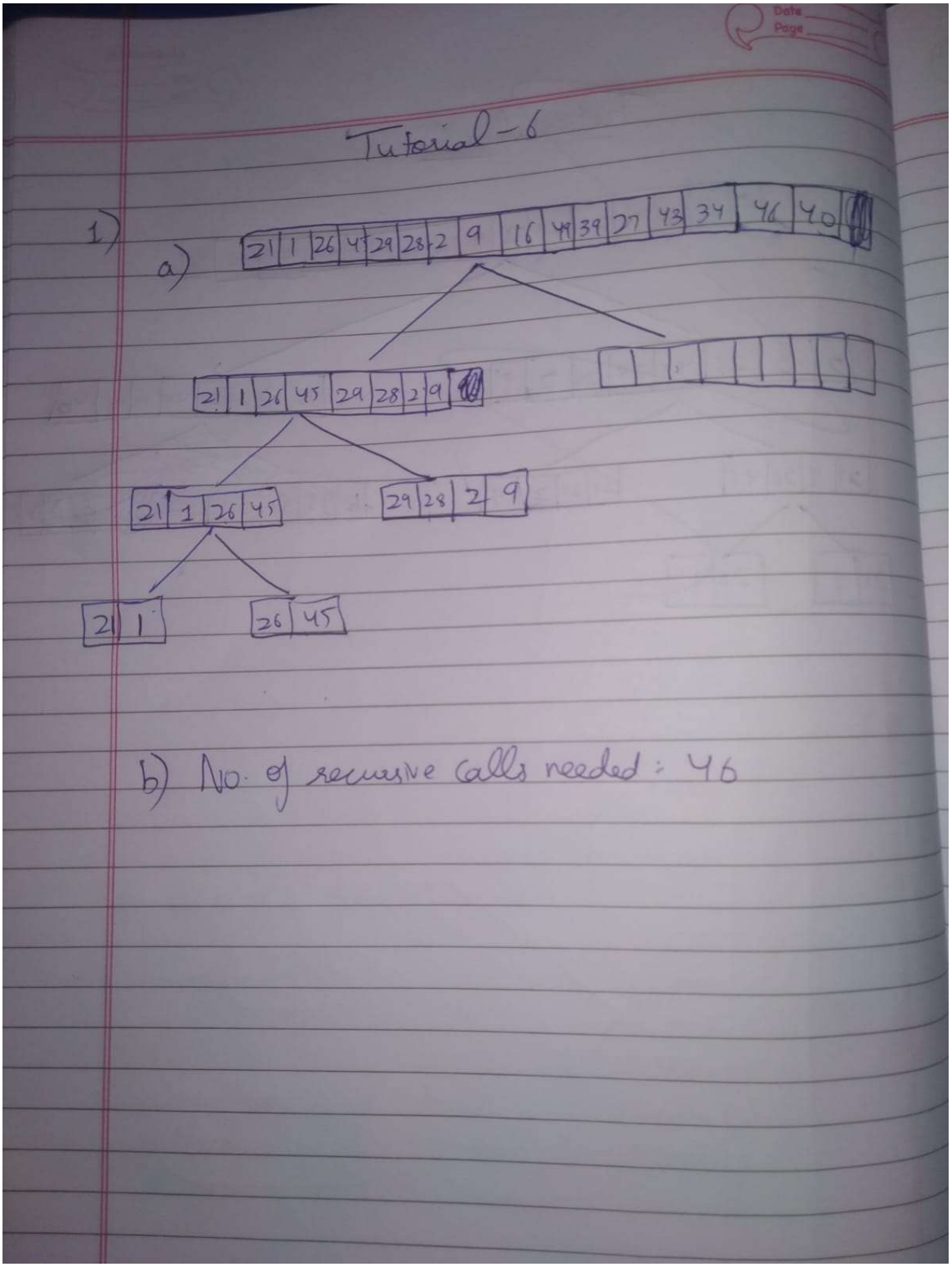
Tutorial 6

1)


Tutorial-6
1)
a)
21 1 26 45 29 28 2 9 16 44 39 27 43 34 46 40
21 1 26 45 29 28 2 9
21 1 26 45
29 28 2 9
21 1
26 45
b) No. of recursive calls needed : 46

2)

a)

```
#include <iostream>
using namespace std;

void countSort(int a[], int size) {
 int output[10];
 int count[10];
 int max = a[0];

 for (int i = 1; i < size; i++) {
  if (a[i] > max)
    max = a[i];
 }
 for (int i = 0; i <= max; ++i) {
  count[i] = 0;
 }

 for (int i = 0; i < size; i++) {
  count[a[i]]++;
 }
 for (int i = 1; i <= max; i++) {
  count[i] += count[i - 1];
 }
 for (int i = size - 1; i >= 0; i--) {
  output[count[a[i]] - 1] = a[i];
  count[a[i]]--;
 }
 for (int i = 0; i < size; i++) {
  a[i] = output[i];
 }
```

```
}
void print(int a[], int size) {
  for (int i = 0; i < size; i++)
    cout << a[i] << " ";
  cout << endl;
}

int main() {
  int a[] = {1,9,7,8,8,5,4,6};
  int n = sizeof(a) / sizeof(a[0]);
  countSort(a, n);
  print(a, n);
}
b)
#include <bits/stdc++.h>
#include <string.h>
using namespace std;
void countSort(char a[])
{
        char temp[strlen(a)];
        int count[26], i;
        memset(count, 0, sizeof(count));
        for (i = 0; a[i];i++)
                ++count[a[i]-'a'];
        for (i = 1; i <26; i++)
                count[i] += count[i - 1];
        for (i = 0; a[i];i++) {
                temp[count[a[i]-'a'] - 1] = a[i];
                --count[a[i]-'a'];
        }
```

```
        for (i = 0; a[i]; i++)
                a[i] = temp[i];
}

int main()
{
        char a[] = "hello";

        countSort(a);

        cout << "Sorted character array is " << a;
        return 0;
}
```

3)

```
int getMax(int arr[], int n)
{
   int mx = arr[0];
   for (int i = 1; i < n; i++)
      if (arr[i] > mx)
         mx = arr[i];
   return mx;
}

void countSort(int arr[], int n, int exp)
{
   int output[n]; // output array
   int i, count[10] = { 0 };
   for (i = 0; i < n; i++)
      count[(arr[i] / exp) % 10]++;
   for (i = 1; i < 10; i++)
      count[i] += count[i - 1];
```

```
    for (i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}
void radixsort(int arr[], int n)
{
    int m = getMax(arr, n);
    for (int exp = 1; m / exp > 0; exp *= 10)
        countSort(arr, n, exp);
}
```

4)

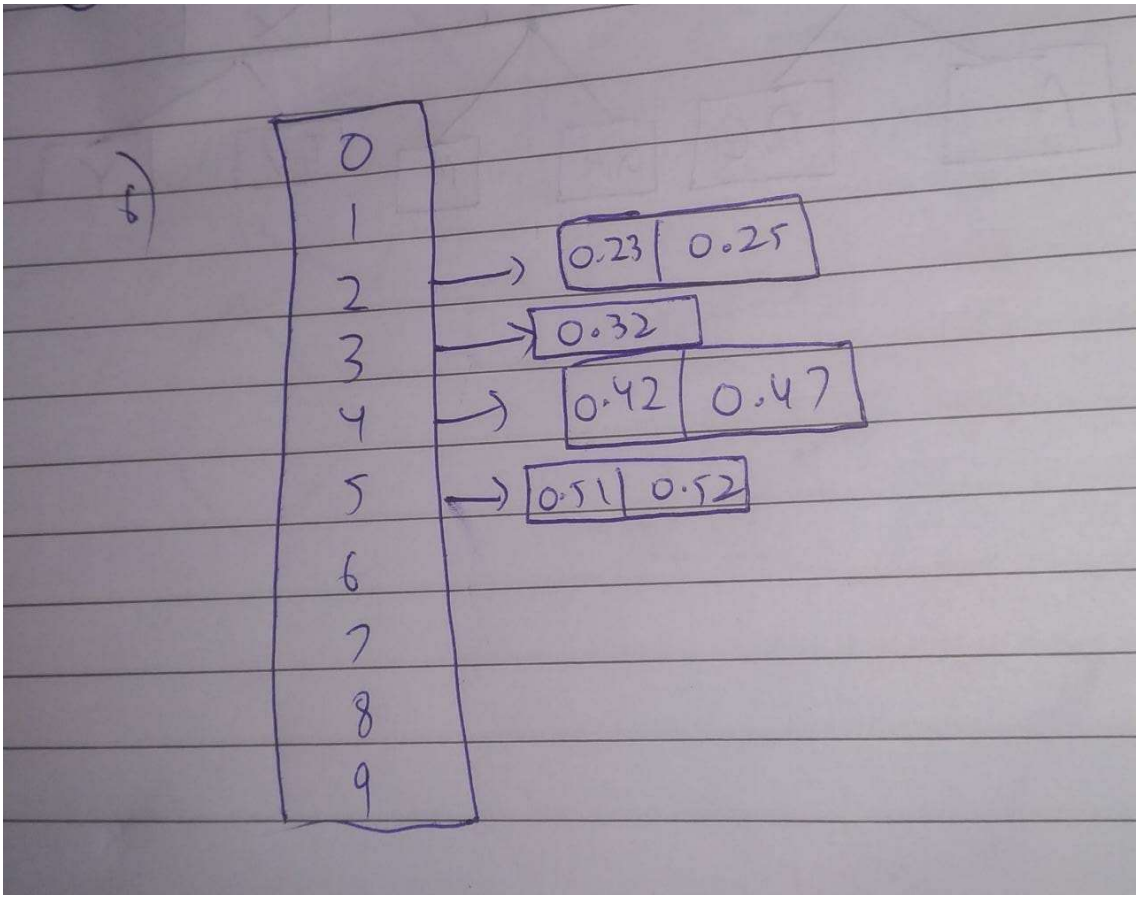
```
#include <bits/stdc++.h>
using namespace std;
void Sort(int a[], int n)
{

        for (int i = 0; i < n; i++)
                if (a[i] & 1)
                        a[i] *= -1;
        sort(a, a + n);
        for (int i = 0; i < n; i++)
                if (a[i] & 1)
                        a[i] *= -1;
}
int main()
{
```

```
        int a[] = { 11,22,23,56,89,91};
        int n = sizeof(a) / sizeof(int);
        Sort(a, n);
        for (int i = 0; i < n; i++)
                cout << a[i] << " ";
        return 0;
}
```

5)

(not done question wrong)

6)



7)

When Pivot is middle element will have least no of steps as the data is in sorted order already so if pivot is middle element it will divide array into two equal parts have log n steps and if pivot is selected last or first element it will take n^2 steps

Tutorial-7

1)

a)

```
#include <bits/stdc++.h>
using namespace std;

#define SIZE(arr) (sizeof(arr)/sizeof(arr[0]))
class Node
{
	public:
	int data;
	Node *next;
	Node *child;
};
Node *createList(int *arr, int n)
{
	Node *head = NULL;
	Node *p;

	int i;
	for (i = 0; i < n; ++i)
	{
		if (head == NULL)
			head = p = new Node();
		else
		{
			p->next = new Node();
			p = p->next;
		}
		p->data = arr[i];
```

```
                p->next = p->child = NULL;
        }
        return head;
}
void print(Node *head)
{
        while (head != NULL)
        {
                cout << head->data << " ";
                head = head->next;
        }
        cout<<endl;
}
void row(Node *head)
{

        if (head == NULL)
        return;

        Node *tmp;
        Node *tail = head;
        while (tail->next != NULL)
                tail = tail->next;
        Node *cur = head;
        while (cur != tail)
        {

                if (cur->child)
                {

                        tail->next = cur->child;
```

```
                        tmp = cur->child;
                        while (tmp->next)
                                tmp = tmp->next;
                        tail = tmp;
                }

                cur = cur->next;
        }
}

int main(void)
{
        int arr1[] = {1,2,3,4,5};
        int arr2[] = {6,7};
        int arr3[] = {8,9};
        int arr4[] = {10,11};
        int arr5[] = {12};
        Node *head1 = createList(arr1, SIZE(arr1));
        Node *head2 = createList(arr2, SIZE(arr2));
        Node *head3 = createList(arr3, SIZE(arr3));
        Node *head4 = createList(arr4, SIZE(arr4));
        Node *head5 = createList(arr5, SIZE(arr5));
        head1->child = head2;
        head1->next->next->child = head3;
        head2->child = head4;
        head4->child = head5;
        row(head1);
        print(head1);
        return 0;
```

```
}
b)

#include <bits/stdc++.h>
using namespace std;

#define SIZE(arr) (sizeof(arr)/sizeof(arr[0]))
class Node
{
	public:
	int data;
	Node *next;
	Node *child;
};
Node *createList(int *arr, int n)
{
	Node *head = NULL;
	Node *p;

	int i;
	for (i = 0; i < n; ++i)
	{
		if (head == NULL)
			head = p = new Node();
		else
		{
			p->next = new Node();
			p = p->next;
		}
		p->data = arr[i];
		p->next = p->child = NULL;
```

```
        }
        return head;
}
void print(Node *head)
{
        while (head != NULL)
        {
                cout << head->data << " ";
                head = head->next;
        }
        cout<<endl;
}
Node * coloumn(Node *node)
{

        if (node == NULL)
     return NULL;
   static Node *last;
   last = node;
   Node *next = node->next;
   if (node->child)
     node->next =coloumn(node->child);
   if (next)
     last->next =coloumn(next);

   return node;
}

int main(void)
{
        int arr1[] = {1,2,3,4,5};
```

```
        int arr2[] = {6,7};
        int arr3[] = {8,9};
        int arr4[] = {10,11};
        int arr5[] = {12};
        Node *head1 = createList(arr1, SIZE(arr1));
        Node *head2 = createList(arr2, SIZE(arr2));
        Node *head3 = createList(arr3, SIZE(arr3));
        Node *head4 = createList(arr4, SIZE(arr4));
        Node *head5 = createList(arr5, SIZE(arr5));
        head1->child = head2;
        head1->next->next->child = head3;
        head2->child = head4;
        head4->child = head5;
        head1=coloumn(head1);
        print(head1);
        return 0;
}
```

2)

a)

```
#include<iostream>
#define R 5
#define C 5
using namespace std;
class value_list
{public:
int column_index;
int value;
```

```
value_list *next;
};

class row_list
{public:
int row_number;
row_list *link_down;
value_list *link_right;
};

void create_value_node(int data, int j, row_list **z)
{ value_list *temp, *d;
temp = new value_list();
temp->column_index = j+1;
temp->value = data;
temp->next = NULL;
if ((*z)->link_right==NULL)
(*z)->link_right = temp;
else
{ d = (*z)->link_right;
while(d->next != NULL)
d = d->next;
d->next = temp;
}
}

void create_row_list(row_list **start, int row,int column,
int Sparse_Matrix[R][C])
{
for (int i = 0; i < row; i++)
{
```

```
row_list *z, *r;
z = new row_list;
z->row_number = i+1;
z->link_down = NULL;
z->link_right = NULL;
if (i==0)
*start = z;
else
{
r = *start;
while (r->link_down != NULL)
r = r->link_down;
r->link_down = z;
}
for (int j = 0; j < 5; j++)
{
if (Sparse_Matrix[i][j] != 0)
{
create_value_node(Sparse_Matrix[i][j], j, &z);
} } } }

void print_LIL( row_list *start)
{
row_list *r;
value_list *z;
r = start;
while (r != NULL)
{
if (r->link_right != NULL)
{
cout<<"\nrow= "<< r->row_number;
```

```
z = r->link_right;
while (z != NULL)
{
cout<<"\n column= "<< z->column_index<< " value= "<<z->value;
z = z->next;
}
}
r = r->link_down;
}
}

int main()
{
int Sparse_Matrix[R][C] =
{
{0 , 0 , 2 , 0 , 3},
{0 , 0 , 4 , 0, 0 },
{0 , 0 , 0 , 0 , 0 },
{0 , 1 , 5 , 0 , 0 },
{0 , 0 , 0, 0 , 4}

};
row_list* start = NULL;
create_row_list(&start, R, C, Sparse_Matrix);
print_LIL(start);
return 0;
}
```

b)

```
#include<iostream>
#define R 5
#define C 5
```

```
using namespace std;
class value_list
{public:
int column_index;
int value;
value_list *next;
};

class row_list
{public:
int row_number;
row_list *link_down;
value_list *link_right;
};

void create_value_node(int data, int j, row_list **z)
{ value_list *temp, *d;
temp = new value_list();
temp->column_index = j+1;
temp->value = data;
temp->next = NULL;
if ((*z)->link_right==NULL)
(*z)->link_right = temp;
else
{ d = (*z)->link_right;
while(d->next != NULL)
d = d->next;
d->next = temp;
}
}
```

```
void create_row_list(row_list **start, int row,int column,
int Sparse_Matrix[R][C])
{
for (int i = 0; i < row; i++)
{
row_list *z, *r;
z = new row_list;
z->row_number = i+1;
z->link_down = NULL;
z->link_right = NULL;
if (i==0)
*start = z;
else
{
r = *start;
while (r->link_down != NULL)
r = r->link_down;
r->link_down = z;
}
for (int j = 0; j < 5; j++)
{
if (Sparse_Matrix[i][j] != 0)
{
create_value_node(Sparse_Matrix[i][j], j, &z);
} } } }

void print_LIL( row_list *start)
{
row_list *r;
value_list *z;
r = start;
```

```
while (r != NULL)
{
if (r->link_right != NULL)
{
cout<<"\nrow= "<< r->row_number;
z = r->link_right;
while (z != NULL)
{
cout<<"\n column= "<< z->column_index<< " value= "<<z->value;
z = z->next;
}
}
r = r->link_down;
}
}

int main()
{
int Sparse_Matrix[R][C] ;
for(int i=0;i<R;i++)
{
for(int j=0;j<C;j++)
{
cin>> Sparse_Matrix[i][j];
}
row_list* start = NULL;
create_row_list(&start, R, C, Sparse_Matrix);
print_LIL(start);
return 0;
}
```

Tutorial-8

1)

Pre: A B F I J G K C D H L M E

Post: I J K G B C L M H D E A

Inorder:I F J B K G C A D L H M E

2)

a) YES

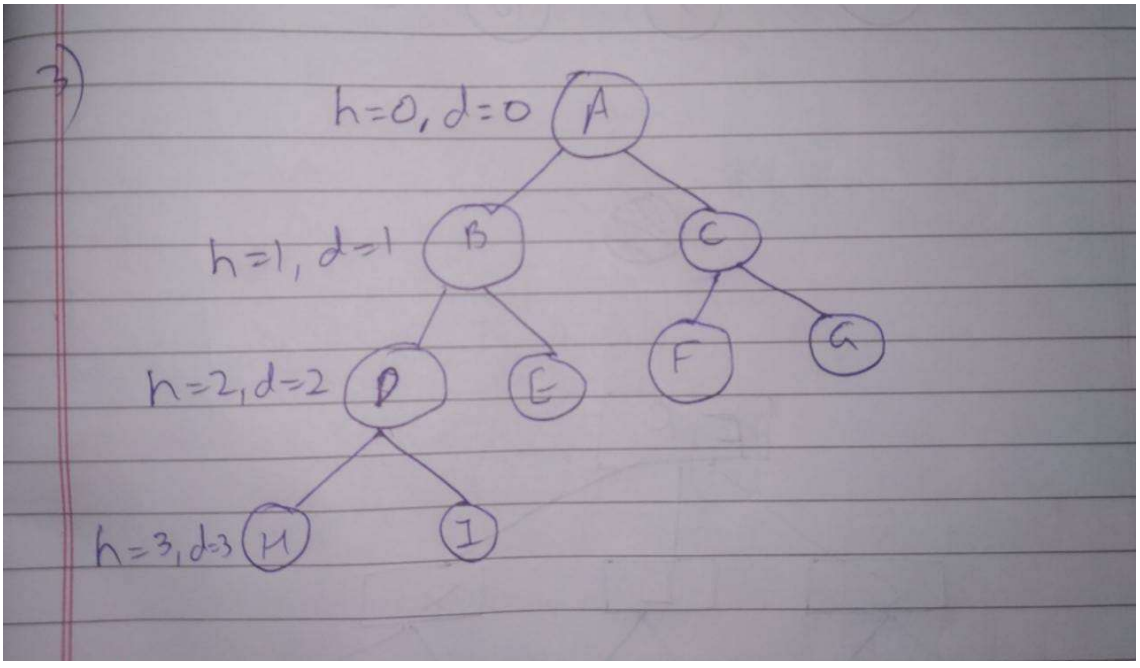
b) A B C D E F G H I 0 0 0 0 0 0 0

c) (n-1)/2

d)2n+1 and 2(n+1)

e)efficient one as it has height of log(n)

3)



4)

a)

inorder:4 2 7 5 1 3 1 0 8 6 9 11

Preorder:1 2 4 5 7 3 6 8 10 9 11

Postorder:4 7 5 10 8 11 9 6 3 1

d)

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;


class Node {
public:
	int data;
	Node *left, *right;

	Node(int data)
	{
		this->data = data;
		this->left = NULL;
		this->right = NULL;
	}
};

int findMax(Node* root)
{

	if (root == NULL)
		return INT_MIN;
	int res = root->data;
	int lres = findMax(root->left);
	int rres = findMax(root->right);
	if (lres > res)
		res = lres;
	if (rres > res)
```

```
                res = rres;
        return res;
}
int findMin(Node* root)
{

        if (root == NULL)
                return INT_MAX;
        int res = root->data;
        int lres = findMin(root->left);
        int rres = findMin(root->right);
        if (lres < res)
                res = lres;
        if (rres < res)
                res = rres;
        return res;
}

int main()
{
        Node* NewRoot = NULL;
        Node* root = new Node(1);
        root->left = new Node(2);
        root->right = new Node(3);
        root->left->right = new Node(4);
        root->left->right = new Node(5);
        root->left->right->left = new Node(7);
        root->right->right = new Node(6);
        root->right->right->left = new Node(8);
        root->right->right->right = new Node(9);
        root->right->right->left->left = new Node(10);
```

```
    root->right->right->right->right = new Node(11);
    cout << "Maximum element is " << findMax(root) << endl;
    cout << "Minimum element is " << findMin(root) << endl;
    return 0;
}
```

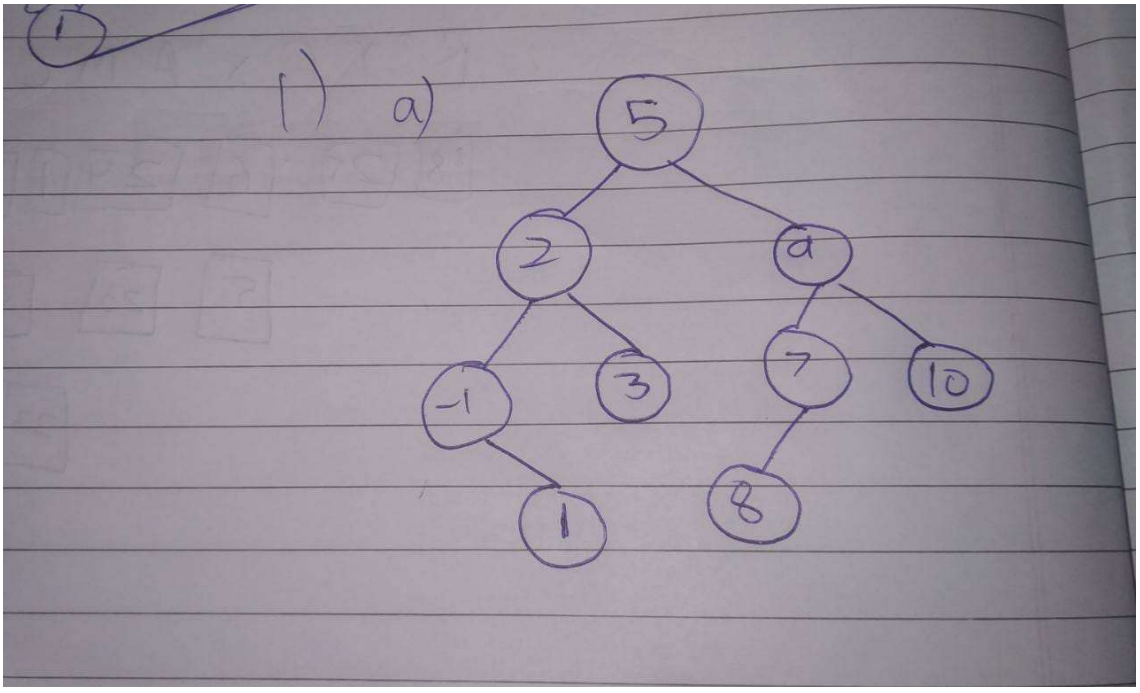
5)

If binary tree has height h, maximum number of nodes will be when all levels are completely full. Total number of nodes will be 2^0 + 2^1 + …. 2^h = 2^(h+1)-1.

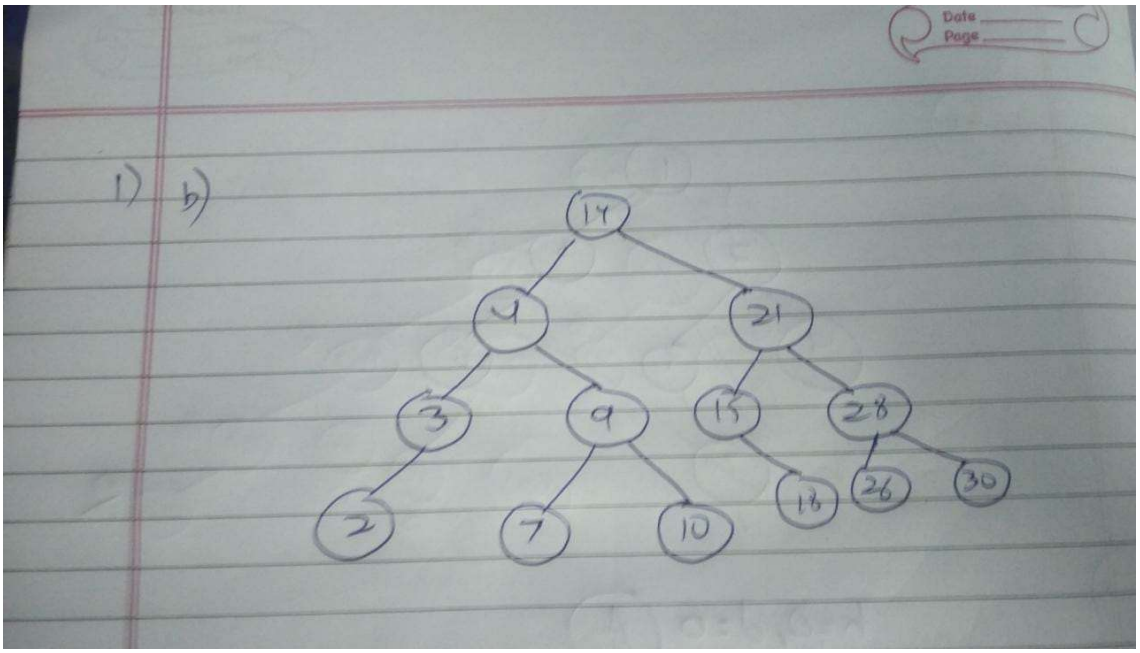
If binary tree has **height h**, minimum number of **nodes is h+1** (in case of left skewed and right skewed binary tree).

Tutorial -9

1) a)



b)



2)

```
voidTraversal(Node* root)
{
   struct Node *c, *p;

   if (root == NULL)
      return;

   c= root;
   while (c!= NULL) {

      if (c>left == NULL) {
         cout<< c->data;
         c = c->right;
      }
      else {
         pre = c->left;
         while (p->right != NULL&& p->right != c)
            p = p->right;
```

```
        if (p->right == NULL) {
           p->right = c;
           c = c->left;
        }

        else {
           pre->right = NULL;
           cout<<c->data;
           c = c->right;
        }
     }
  }
}
```

(3,4,5) not to be done

6)

1) Get the Middle of the array and make it root.

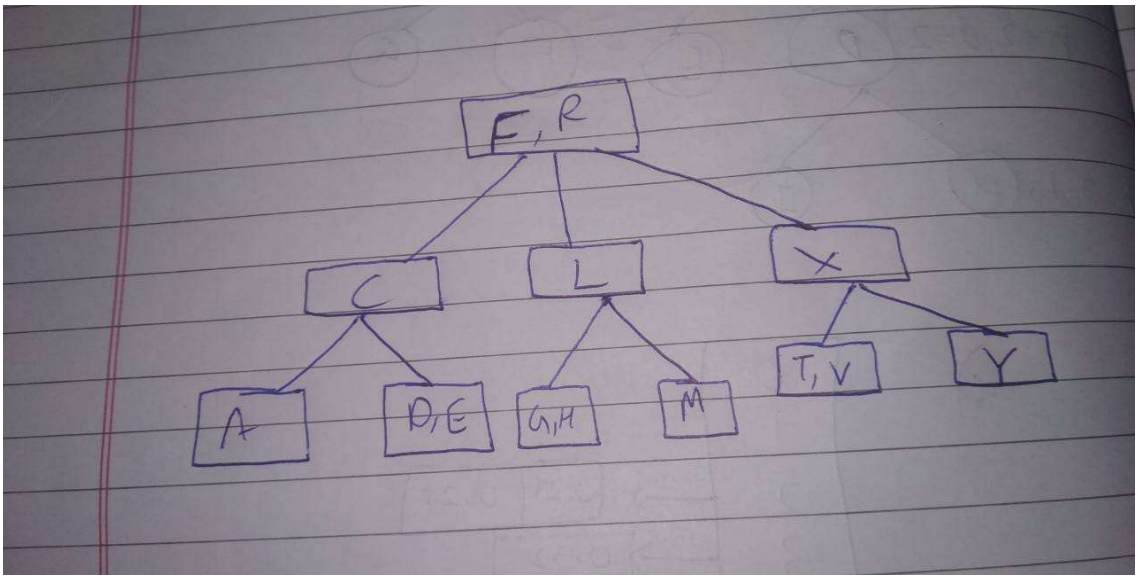
2) Recursively do same for left half and right half.

a) Get the middle of left half and make it left child of the root created in step 1.

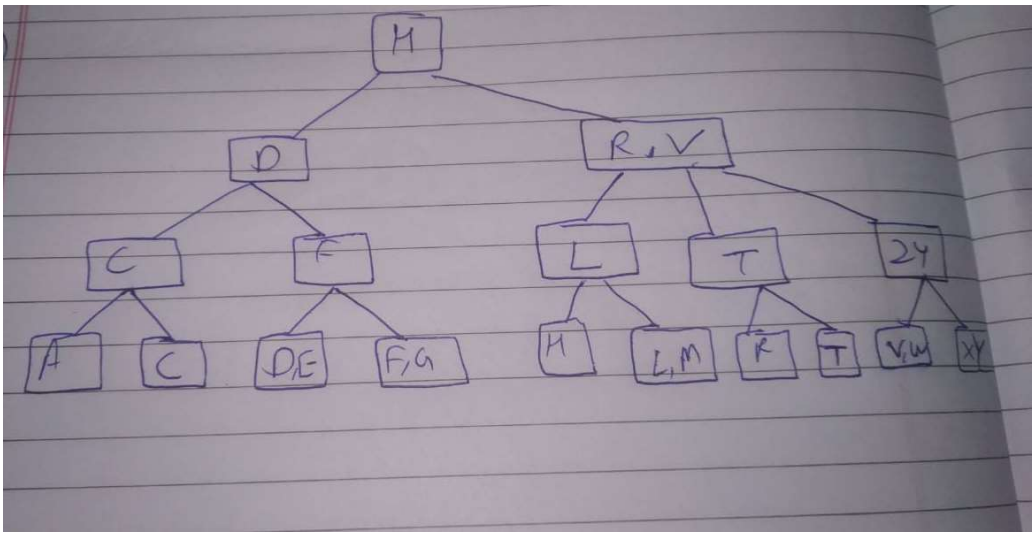
b) Get the middle of right half and make it right child of the root created in step 1.

7) rotation in avl tree are more complex making the process complex and more time complex.

8)



9)

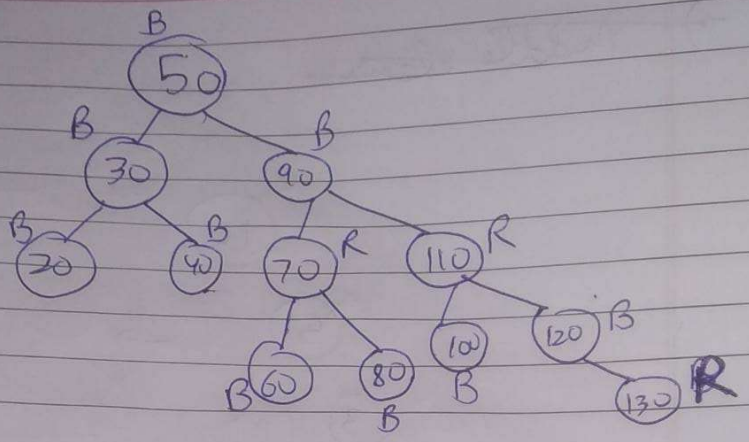


10)

B+-Trees store records at the leaf level of the tree, they maximize the branching factor of the internal nodes. High branching factor allows for a tree of lower height. Lower tree height allows for less searching,insertion and deletion time.

11)

11)



After deletion RB tree left