Homwork 3

Q1

1/2

```cpp
#include <iostream>
#define INF 1e9

using namespace std;

int dis[10], first[10], second[10], length[10];
int check;

struct edge{
    int first;
    int second;
    int length;
};

edge edge[10];

void BellmanFord(int n, int m){
    for(int j=1; j<n; ++j){  // 最多鬆弛n-1輪
        check = 0;  // 標記在本輪鬆弛中陣列dis是否發生更新
        for(int i=1; i<=m; ++i){
            if(dis[edge[i].first] != INF && dis[edge[i].first] + edge[i].length < dis[edge[i].second]){  // relax
                dis[edge[i].second] = dis[edge[i].first] + edge[i].length;
                check = 1;
            }
        }
        if(check == 0)
            break;
    }
}
```

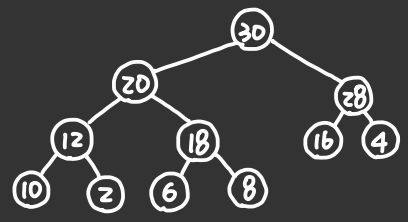
2/2

```cpp
int main(){
    int n, m;
    n=5;
    m=5;

    edge[1].first = 1;
    edge[1].second = 2;
    edge[1].length = -3;

    edge[2].first = 1;
    edge[2].second = 5;
    edge[2].length = 5;

    edge[3].first = 2;
    edge[3].second = 3;
    edge[3].length = 2;

    edge[4].first = 3;
    edge[4].second = 4;
    edge[4].length = 3;

    edge[5].first = 4;
    edge[5].second = 5;
    edge[5].length = 2;

    for(int i=1; i<=n; ++i){
        dis[i] = INF;
    }
    dis[1] = 0;

    BellmanFord(n, m);

    cout << "各個點到點1的最短距離：\n\n";
    for(int i=1; i<=n; i++){
        cout << i << "到點1的最短距離為:" << dis[i];
        cout << endl;
    }

    return 0;
}
```

```
各個點到點1的最短距離： // The shortest length to 1:

1到點1的最短距離為:0  // 1 to 1's shortest length
2到點1的最短距離為:-3  // 2 to 1's shortest length
3到點1的最短距離為:-1  // 3 to 1's shortest length
4到點1的最短距離為:2  // 4 to 1's shortest length
5到點1的最短距離為:4  // 5 to 1's shortest length
Program ended with exit code: 0
```

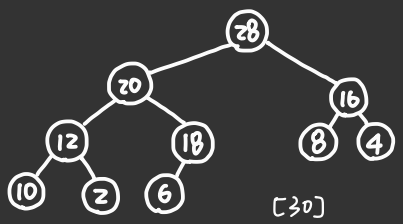
print out

Q2 FIRST loop:



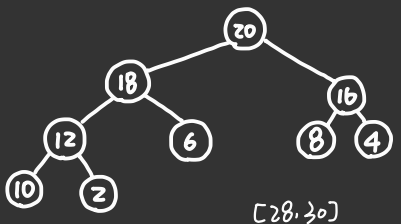
[30, 20, 28, 12, 18, 16, 4, 10, 2, 6, 8]

SECOND loop:



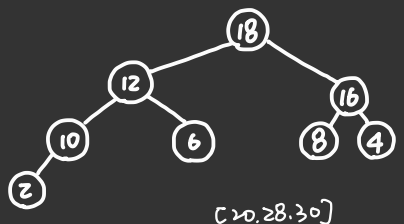
[30]

[28] →



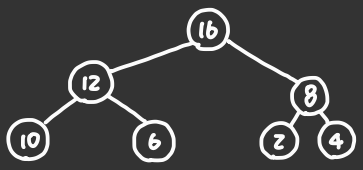
[28, 30]

[20] →



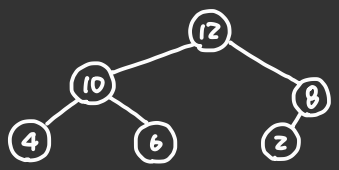
[20, 28, 30]

[18] →



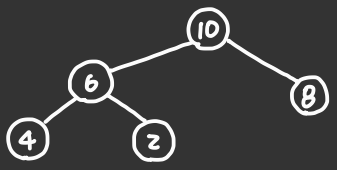
[18, 20, 28, 30]

[16] →



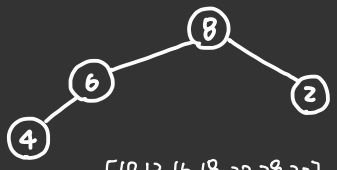
[16, 18, 20, 28, 30]

[12] →



[12, 16, 18, 20, 28, 30]

[10] →



[10, 12, 16, 18, 20, 28, 30]

[8] →

[8, 10, 12, 16, 18, 20, 28, 30]

[6] →

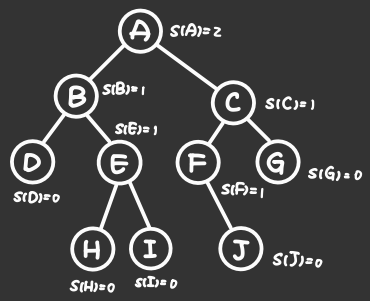
[6, 8, 10, 12, 16, 18, 20, 28, 30]

[4] →

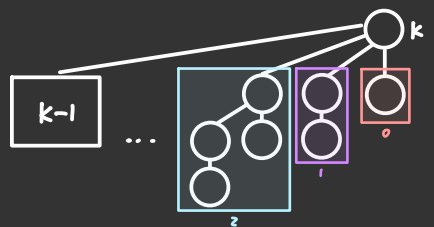
[4, 6, 8, 10, 12, 16, 18, 20, 28, 30]

[2] → [2, 4, 6, 8, 10, 12, 16, 18, 20, 28, 30]

Q3



Q4 Binomial tree B_k



k	node
0	$1 = 2^0$
1	$1+1 = 2 = 2^1$ [B_0+1]
2	$(1+2)+1 = 4 = 2^2$ [B_0+B_1+1]
3	$(1+2+4)+1 = 8 = 2^3$ [$B_0+B_1+B_2+1$]
⋮	
k	$2^0+2^1+2^2+\dots+2^{k-1}+1$ [$B_0+B_1+B_2+\dots+B_{k-1}+1$]

$2^0+2^1+2^2+\dots+2^{k-1}+1$

$= \sum_{n=0}^{k-1} 2^n + 1$

$= \frac{1(1-2^k)}{1-2} + 1$

$= 2^k - 1 + 1 = 2^k$

⇒ the binomial tree B_k has 2^k nodes $(k \geq 0)$ #

Q5

1/3

```
Insert (18,a), (31,b), (1,c), (62,d), (7,e), (0,f), (4,g), (43,h) inorder to the binary search tree.

Before deleting 31, the inorder traversal is:
(0, f) (1, c) (4, g) (7, e) (18, a) (31, b) (43, h) (62, d)

After deleting 31, the inorder traversal is:
(0, f) (1, c) (4, g) (7, e) (18, a) (43, h) (62, d)
Program ended with exit code: 0
```

print out

```cpp
#include <iostream>

using namespace std;

struct node{
    pair<int, char> content;
    node *left;
    node *right;
};

node *newNode(int key, char element){
    node *temp = new node();
    temp->content.first = key;
    temp->content.second = element;
    temp->left = NULL;
    temp -> right = NULL;
  return temp;
}

void print(node *root){
    if(root != NULL){
        print(root->left);
        cout << "(" << root -> content.first << ", " << root -> content.second << ") ";
        print(root->right);
    }
}

node *insert(node *node, pair<int, char> key){
    if(node == NULL)
        return newNode(key.first, key.second);
    if (key.first < node -> content.first)
        node->left = insert(node->left, key);
    else
        node->right = insert(node->right, key);

    return node;
}

node *minValueNode(node *node){
    struct node *current = node;
    // Find the leftmost leaf
    while (current && current->left != NULL)
        current = current->left;
    return current;
}
```

Time Complexity: O(h)

(the heigh (level) of the tree)

2/3

```cpp
node *deleteNode(node *root, pair<int, char> key){
    // the tree is empty
    if (root == NULL)
        return root;

    // Find the node to be deleted
    if (key.first < root-> content.first)
        root->left = deleteNode(root->left, key);
    else if (key.first > root->content.first)
        root->right = deleteNode(root->right, key);
    else{
        // one child or no child
        if(root->left == NULL){
            node *temp = root->right;
            free(root);
            return temp;
        }else if(root->right == NULL){
            node *temp = root->left;
            free(root);
            return temp;
        }

        // two children
        node *temp = minValueNode(root->right);

        // Place the inorder successor in position of the node to be deleted
        root->content = temp->content;

        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->content);
    }
    return root;
}
```

3/3

```cpp
int main(){
    node *root = NULL;
    root = insert(root, pair<int, char>(18, 'a'));
    root = insert(root, pair<int, char>(31, 'b'));
    root = insert(root, pair<int, char>(1, 'c'));
    root = insert(root, pair<int, char>(62, 'd'));
    root = insert(root, pair<int, char>(7, 'e'));
    root = insert(root, pair<int, char>(0, 'f'));
    root = insert(root, pair<int, char>(4, 'g'));
    root = insert(root, pair<int, char>(43, 'h'));

    cout << "Insert (18,a), (31,b), (1,c), (62,d), (7,e), (0,f), (4,g), (43,h) inorder to the
        binary search tree.\n\n";
    cout << "Before deleting 31, the inorder traversal is:\n";
    print(root);

    cout << "\n\nAfter deleting 31, the inorder traversal is:\n";
    root = deleteNode(root, pair<int, char>(31, 'b'));
    print(root);
    cout << endl;
    return 0;
}
```

Q6

1/2

```cpp
#include <iostream>
#include <list>
#include <queue>

using namespace std;

void T(vector<int> adj[], int first, int second)
{
    adj[first].push_back(second);
    adj[second].push_back(first);
}

bool BFS(vector<int> adj[], int from, int to, int num, int distance[])
{
    list<int> queue;
    bool visited[num];
    for(int i=0; i<num; i++){
        visited[i] = false;
        distance[i] = INT_MAX;
    }

    visited[from] = true;
    distance[from] = 0;
    queue.push_back(from);

    while(!queue.empty()){
        int temp = queue.front();
        queue.pop_front();
        for(int i = 0; i < adj[temp].size(); i++){
            if(visited[adj[temp][i]] == false){
                visited[adj[temp][i]] = true;
                distance[adj[temp][i]] = distance[temp] + 1;
                queue.push_back(adj[temp][i]);
                if(adj[temp][i] == to)
                    return true;
            }
        }
    }
    return false;
}
```

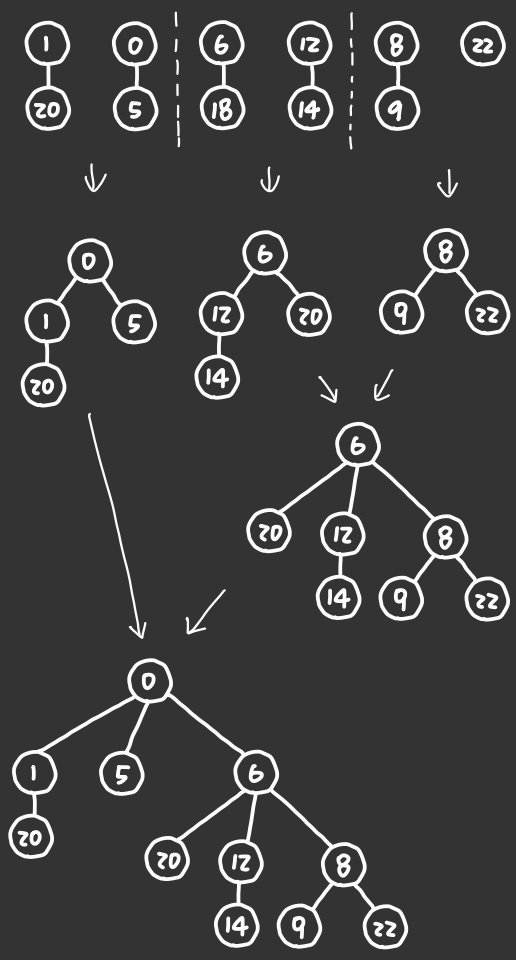
2/2

```cpp
void distance(vector<int> adj[], int v, int remaing_vertice, int num)
{
    int dist[num];
    if(BFS(adj, v, remaing_vertice, num, dist) == false){
        cout << "ERROR";
        return;
    }
    vector<int> path;
    cout << "Shortest path from " << remaing_vertice << " to v: " << dist[remaing_vertice];
}

int main()
{
    int num = 8;
    vector<int> adj[num];
    // assume that the length between every two nodes is 1
    T(adj, 0, 1);
    T(adj, 0, 3);
    T(adj, 1, 2);
    T(adj, 3, 4);
    T(adj, 3, 7);
    T(adj, 4, 5);
    T(adj, 4, 6);
    T(adj, 4, 7);
    T(adj, 5, 6);
    T(adj, 6, 7);
    int v = 0; //root
    for(int remaing_vertice=1; remaing_vertice<8; remaing_vertice++)
    {
        distance(adj, v, remaing_vertice, num);
        cout << endl;
    }
    return 0;
}
```

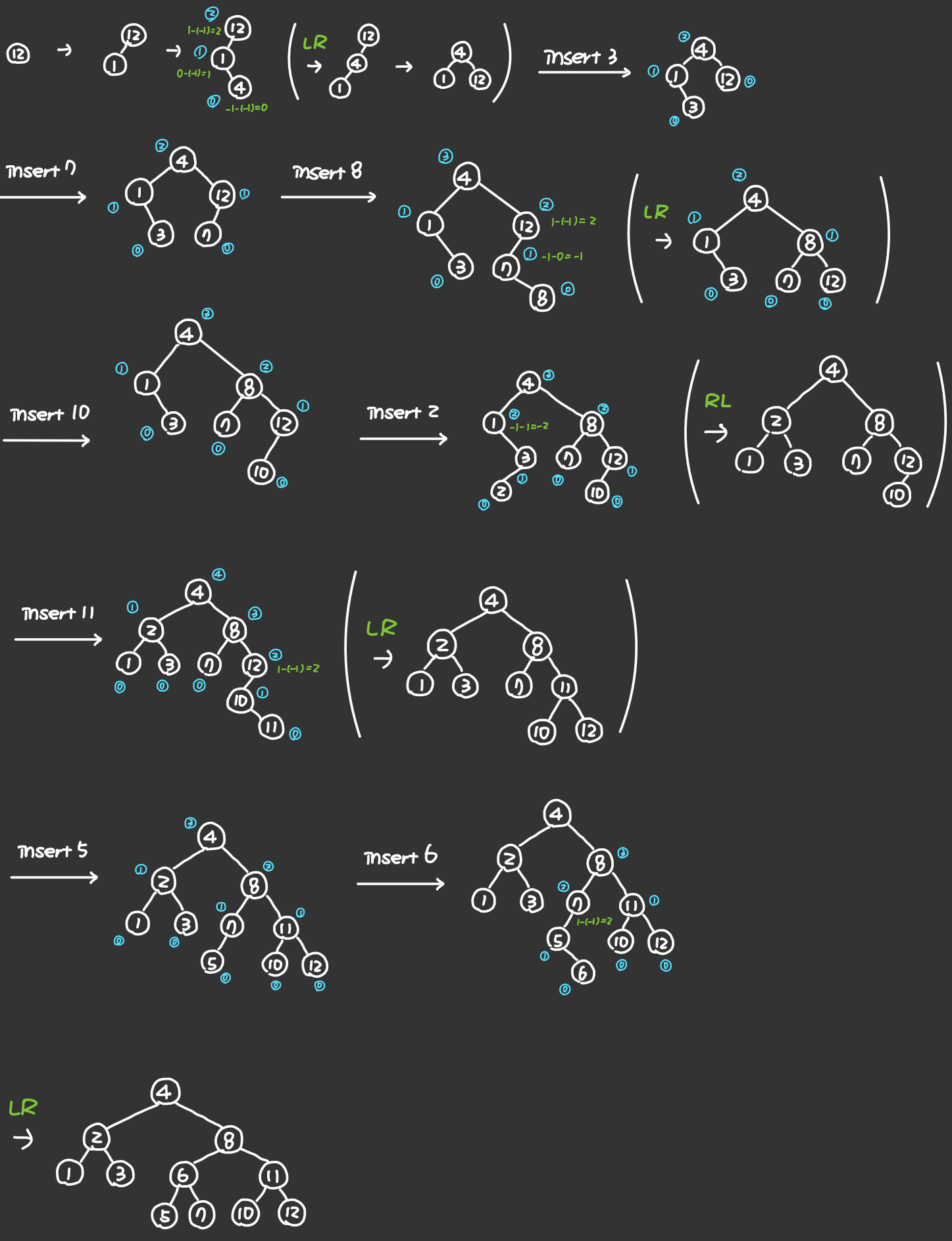
```
Shortest path from 1 to v: 1
Shortest path from 2 to v: 2
Shortest path from 3 to v: 1
Shortest path from 4 to v: 2
Shortest path from 5 to v: 3
Shortest path from 6 to v: 3
Shortest path from 7 to v: 2
Program ended with exit code: 0
```

print out

Q7



Q8


12
1-(-1)=2
0-(-1)=1
-1-(-1)=0
LR
Insert 3
Insert 7
Insert 8
-1-(-1)= 2
-1-0= -1
LR
Insert 10
Insert 2
-1-1=-2
RL
Insert 11
1-(-1)=2
LR
Insert 5
Insert 6
1-(-1)=2
LR

Q9

1/3

```cpp
#include<iostream>

using namespace std;

class Node{
    public:
    int key;
    Node *left;
    Node *right;
    int height;
};

Node* newNode(int key){
    Node* node = new Node();
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return(node);
}

int max(int a, int b){
    if(a>b)
        return a;
    else
        return b;
}

int getHeight(Node *n){
    if (n == NULL)
        return 0;
    return n->height;
}

Node *rotate_right(Node *y){
    Node *x = y->left;
    Node *temp = x->right;
    x->right = y;
    y->left = temp;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;
    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    return x;
}
```

2/3

```cpp
Node *rotate_left(Node *x){
    Node *y = x->right;
    Node *temp = y->left;
    y->left = x;
    x->right = temp;
    x->height = max(getHeight(x->left), getHeight(x->right)) + 1;
    y->height = max(getHeight(y->left), getHeight(y->right)) + 1;

    return y;
}

int BF(Node *N){
    if (N == NULL)
        return 0;
    return getHeight(N->left) - getHeight(N->right);
}

Node* insert(Node* node, int key){
    if (node == NULL)
        return(newNode(key));
    if (key < node->key)
        node->left = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);
    else
        return node;

    node->height = 1 + max(getHeight(node->left), getHeight(node->right));

    int bf = BF(node);
    if (bf > 1 && key < node->left->key)
        return rotate_right(node);
    if (bf < -1 && key > node->right->key)
        return rotate_left(node);
    if (bf > 1 && key > node->left->key){
        node->left = rotate_left(node->left);
        return rotate_right(node);
    }
    if (bf < -1 && key < node->right->key){
        node->right = rotate_right(node->right);
        return rotate_left(node);
    }
    return node;
}
```

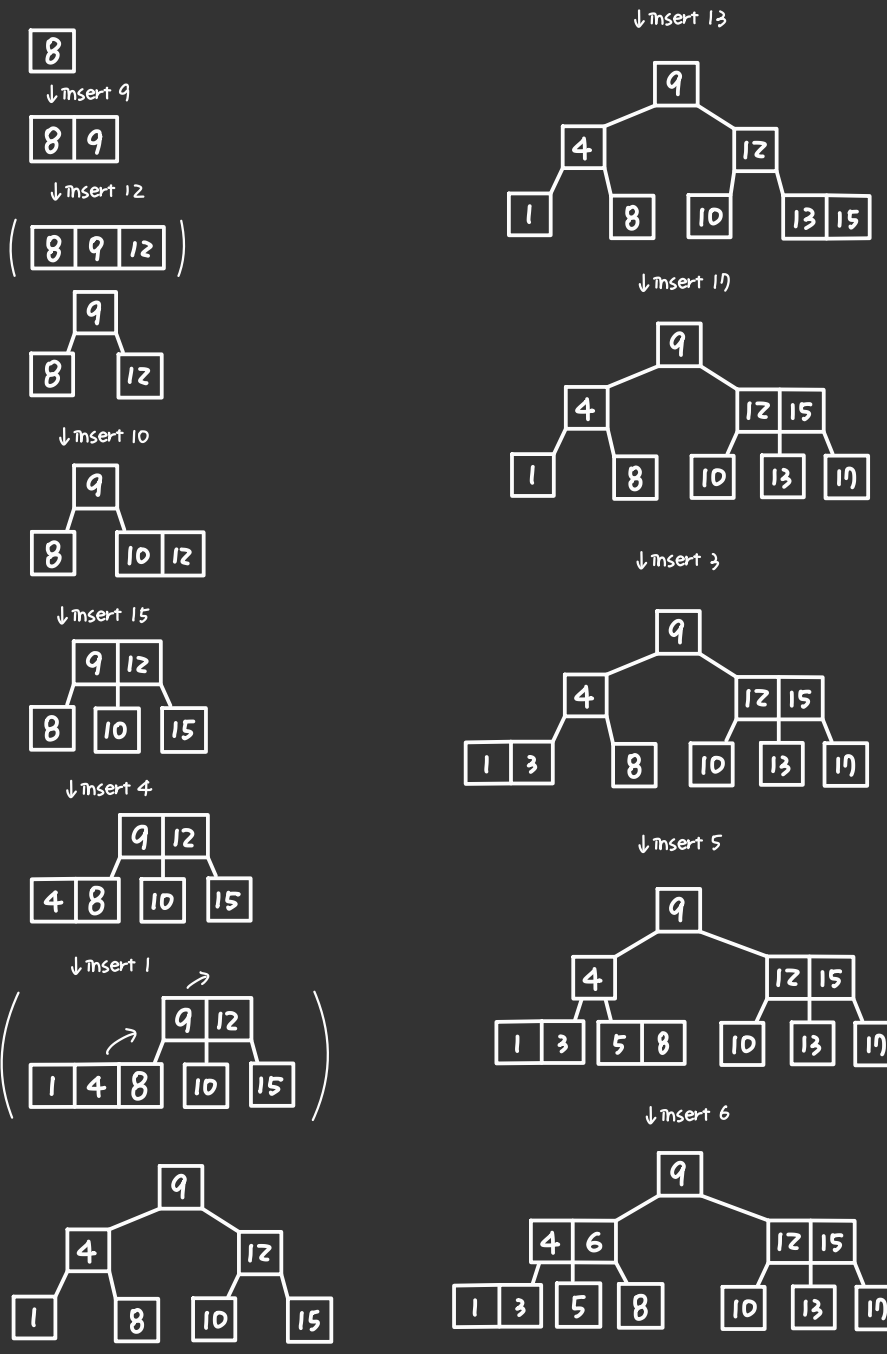
3/3

```cpp
void print(Node *root){
    if(root != NULL){
        print(root->left);
        cout << root->key << " ";
        print(root->right);
    }
}

int main(){
    Node *root = NULL;

    root = insert(root, 12);
    root = insert(root, 1);
    root = insert(root, 4);
    root = insert(root, 3);
    root = insert(root, 7);
    root = insert(root, 8);
    root = insert(root, 10);
    root = insert(root, 2);
    root = insert(root, 11);
    root = insert(root, 5);
    root = insert(root, 6);

    cout << "Ascending order of the AVL tree is: \n";
    print(root);
    cout << endl;

    return 0;
}
```

```
Ascending order of the AVL tree is:
1 2 3 4 5 6 7 8 10 11 12
Program ended with exit code: 0
```

print out

Q 10 $m=3$


8
↓ Insert 9
8 9
↓ Insert 12
8 9 12
9
8 12
↓ Insert 10
9
8 10 12
↓ Insert 15
9 12
8 10 15
↓ Insert 4
9 12
4 8 10 15
↓ Insert 1
9 12
1 4 8 10 15
9
4 12
1 8 10 15
↓ Insert 13
9
4 12
1 8 10 13 15
↓ Insert 17
9
4 12 15
1 8 10 13 17
↓ Insert 3
9
4 12 15
1 3 8 10 13 17
↓ Insert 5
9
4 12 15
1 3 5 8 10 13 17
↓ Insert 6
9
4 6 12 15
1 3 5 8 10 13 17