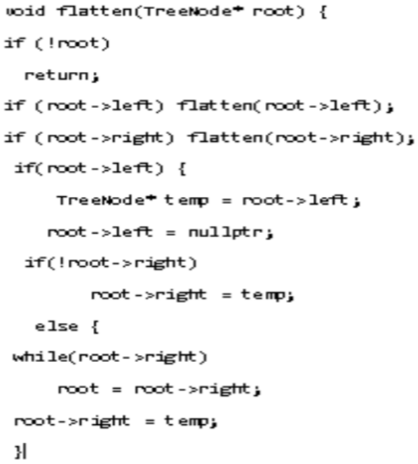
```
void QuickSort(vector<int>& a, int left, int right){
        int i = left, j = right;
        int temp = a[(left + right) / 2];
        do{
                while (a[i] < temp)  i++;
                while (a[j] > temp)  j--;
                if (i <= j){
                        swap(a[i], a[j]);
                        i++;j--;}
        } while (i < j);
        if (left < j)QuickSort(a, left, j);
        if (right > i)QuickSort(a, i, right);}

void SelectionSort(vector<int> a, int n){
        int min;
        for (int i = 0; i < n; i++){
        min = i;
        for (int j = i + 1; j < n; j++){
          if (a[j] < a[min]){min = j;}}
        swap(a[i], a[min]);}}

void Push_Down(int *&a,int index,int limit){
int check = 0;
while (index <= limit / 2 && check != 1){int son;
 if (index == limit / 2 && index * 2 == limit) son = 2*index;
 else{(a[2*index]<a[2*index+1])?son=2*index:son=2*index+1;}
 if (a[index]>a[son]){swap(a[index], a[son]);index = son;}
 else{check = 1;}}}
void HeapSort(int* &a){int n = MAX;
 for (int height = MAX / 2; height > 0; height--){
 Push_Down(a, height,n);}
 for (int k = n; k > 2; k--){
 for (int l = k / 2; l > 0; l--){Push_Down(a, l, k);}}}

 void InsertAnElement(int* &a, int index,int par){
 for (int i = 0; i < index; i+=par){
  if (a[i] > a[index]){
   int temp = a[index];
   for (int j = index; j>i; j-=par){a[j] = a[j - par];}
   a[i] = temp; return;}}}
 void ShellSort(int* &a,int par){
  if (par == 0)return;
  int count = 0;
  while (1){
  InsertAnElement(a, count,par);
  count+=par;if (count >= MAX)break;}ShellSort(a,par/2);}

void BinaryInsertionSort(vector<int> a, int n){
 int i, j, left, mid, right;long long x;
 for (i = 1; i < n;i++){
  x = a[i];left = 0;right = i;
  while (left < right){
   mid = (left + right) / 2;
   if (a[mid] <= x) left = mid + 1;
   else right = mid;}
   for (j = i; j > right; j--){a[j] = a[j - 1];}
  a[right] = x;}}

 void lnr(Node* root){
  Node* pre; Node* current = root;
  while (current){
   if (!current->pLeft){
    cout << current->key << " ";
    current = current->pRight;}
   else{pre = current->pLeft;
    while (pre->pRight && pre->pRight != current)
    {pre = pre->pRight;}
    if (!pre->pRight)
   {pre->pRight = current;current = current->pLeft;}
    else{
     cout << current->key << " ";
     pre->pRight = NULL;
     current = current->pRight;}}}}

 // A utility function to find the vertex with
 // minimum key value, from the set of vertices
 // not yet included in MST
 int minKey(int key[], bool mstSet[])
 {
  // Initialize min value
  int min = INT_MAX, min_index;
  for (int i = 0; i < 6; i++)
   if (mstSet[i] == false && key[i] < min)
   { min = key[i]; min_index = I;}
  return min_index;}

 void print(int parent[], int graph[6][6]){
  for (int i = 1; i < 6; i++)
  {cout << parent[i] + 1 << " - " << i + 1 <<
 " " << "\t" << graph[i][parent[i]] << endl;}}

 // Function to construct and print MST for
 // a graph represented using adjacency
 // matrix representation
 void primMST(int graph[6][6]){
  int parent[6];// Mảng chứa MST được xây dựng
  int key[6];// Giá trị được sử dụng để chọn cạnh có trọn
  bool mstSet[6];// Bieu dien nhung dinh chua bao gom tro
  for (int i = 0; i < 6; i++)// Initialize all keys as INFINITE
  {key[i] = INT_MAX; mstSet[i] = false;}
 // Always include first 1st vertex in MST. Make key 0 so that this vertex is picked as first vertex.
  key[0] = 0; parent[0] = -1; // First node is always root of MST
 // The MST
  for (int count = 0; count < 6 - 1; count++){
   int u = minKey(key, mstSet);// chọn key tối thiểu từ tập hợp các đỉnh chưa có trong MST
   mstSet[u] = true;// Thêm đỉnh đã chọn vào MST
 // Cập nhật giá trị khóa và chỉ mục của parent của các đỉnh liền kề được chọn
 // CHi xét những đỉnh chưa có trong MST
  for (int v = 0; v < 6; v++)
 // graph[u][v] is non zero only for adjacent vertices of m mstSet[v] is false for vertices not yet included in MST
 // Update the key only if graph[u][v] is smaller than key[v]
 if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v]) {parent[v] = u; key[v] = graph[u][v];}
 print(parent, graph);// print the constructed MST}
```

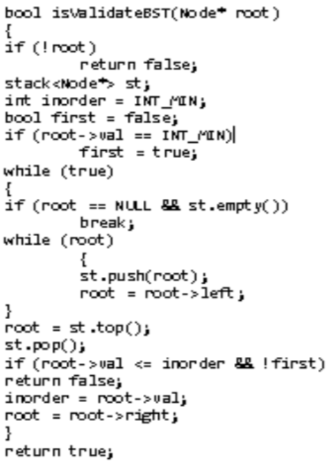
```
void Merge(vector<int> a, int nb, int nc, int k){
 int p, pb, pc, ib, ic, kb, kc;
 p = pb = pc = 0; ib = ic = 0;
 while ((0 < nb) && (0 < nc)){
  kb = min(k, nb); kc = min(k, nc);
  if (b[pb + ib] <= c[pc + ic]){
  a[p++] = b[pb + ib]; ib++;if (ib == kb){
   for (; ic < kc; ic++) a[p++] = c[pc + ic];
   pb += kb;pc += kc;ib = ic = 0;nb -= kb;nc -= kc;}}
  else{
   a[p++] = c[pc + ic]; ic++; if (ic == kc) {
   for(; ib < kb; ib++) a[p++] = b[pb  + ib];
   pb += kb; pc += kc;ib = ic = 0;nb -= kb; nc -= kc;}}}}
void MergeSort(vector<int> &a, int n){
int p, pb, pc;int i, k = 1;do{p = pb = pc = 0;
 while (p < n){
  for (i = 0; (p < n) && (i < k); i++) b[pb++] = a[p++];
  for (i = 0; (p < n) && (i < k); i++) c[pc++] = a[p++];}
 Merge(a, pb, pc, k);k *= 2;} while (k < n);}


int partition(int arr[], int l, int h) {
int x = arr[h]; int i = (l - 1);
for (int j = l; j <= h - 1; j++) {if (arr[j] <= x) {
 i++; swap(&arr[i], &arr[j]);}}
 swap(&arr[i + 1], &arr[h]); return (i + 1);}
void quickSortIterative(int arr[], int l, int h) {
int stack[h - l + 1]; stack[++top] = l; stack[++top] = h;
// Keep popping from stack while is not empty
 while (top >= 0) {// Pop h and l
 h = stack[top--]; l = stack[top--];
//Setpivotelementatits correct positioninsortedarray
 int p = partition(arr, l, h);
 if (p - 1 > l) {stack[++top] = l; stack[++top] = p - 1;}
 if (p + 1 < h) {stack[++top] = p + 1; stack[++top] = h;}}}

      void getVerticalOrder(TREE* root, int hd, map<int, vector<int>> &m){
       if (root == NULL)   return;
       m[hd].push_back(root->Data);
       getVerticalOrder(root->Left, hd - 1, m);
       getVerticalOrder(root->Right, hd + 1, m);}

      void printVerticalOrder(TREE* root){
       map < int, vector<int> > m;
       int hd = 0; int j;
       getVerticalOrder(root, hd, m);
       map< int, vector<int> > ::iterator it;
       for (it = m.begin(), j= 0 ; it != m.end(); it++, j++){
       cout << char(65 + j) << ": ";
       for (int i = 0; i<it->second.size(); ++i) cout << it->second[i] << " ";
       cout << endl;       }}

        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 };
        // Store count of occurrences in count[]
         for (i = 0; i < n; i++) count[(arr[i] / exp) % 10]++;
        // Change count[i] so that count[i] now contains actual
        position of this digit in output[]
         for (i = 1; i < 10; i++) count[i] += count[i - 1];
        // Build the output array
        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){
              // Find the maximum number to know number of digits
        int m = getMax(arr, n);
        for (int exp = 1; m / exp > 0; exp *= 10) countSort(arr, n, exp);}

          int minDistance(int dist[], bool sptSet[]){
           int min = INT_MAX, min_index; // Initialize min value
           for (int v = 0; v < 9; v++)
            if (sptSet[v] == false && dist[v] <= min)
             min = dist[v], min_index = v;
           return min_index;}

          // A utility function to print the constructed distance array
          void printSolution(int dist[])
          {
                 printf("Vertex \t\t Distance from Source\n");
                 for (int i = 0; i < 9; i++)
                        printf("%d \t\t %d\n", i, dist[i]);
          }
```

```
Node* RightRotate(Node* y)
{
        Node* x = y->pLeft;
        Node* T2 = x->pRight;

        //Thực hiện xoay
        x->pRight = y;
        y->pLeft = T2;

        //Cập nhật lại chiều cao
        y->Bal = getBalance(y);
        x->Bal = getBalance(x);

        return x;
}

Node* LeftRotate(Node* x)
{
        Node* y = x->pRight;
        Node* T2 = y->pLeft;

        //Thực hiện xoay
        y->pLeft = x;
        x->pRight = T2;

        //Cập nhật lại chiều cao
        y->Bal = getBalance(y);
        x->Bal = getBalance(x);

        return y;
}

int getBalance(Node* p)
{
 if (p == NULL)return 0;
 return Height(p->pRight) - Height(p->pLeft);
}
```

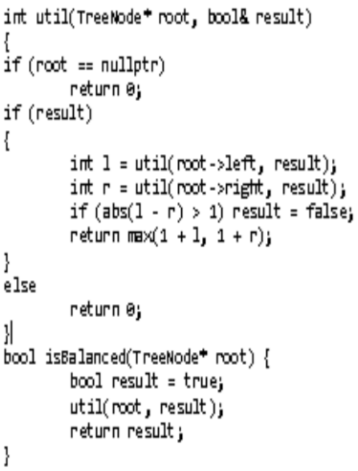
```
      void Insert(Node*& root, int k){
       if (root == NULL){
        root = getNode(k);
        return;}
       if (root->Data > k) Insert(root->pLeft, k);
       else if (root->Data < k) Insert(root->pRight, k);
       else return;
       root->Bal = getBalance(root);//Lấy hệ số cân bằng
       int bal = getBalance(root);
             //Lệch trái cùng phía
       if (bal < -1 && k < root->pLeft->Data) root = RightRotate(root);
             //Lệch phải cùng phía
       else if (bal > 1 && k >root->pRight->Data) root = LeftRotate(root);
             //Lệch trái khác kía
       else if (bal < -1 && k > root->pLeft->Data)
      {root->pLeft = LeftRotate(root->pLeft); root = RightRotate(root);}
             //Lệch phait khác phía
       else if (bal > 1 && k < root->pRight->Data)
      {root ->pRight = RightRotate(root->pRight); root = LeftRotate(root);}}

Node* FindPos(Node* root){
 Node* cur = root;
 while (cur && cur->pLeft != NULL)
 {cur = cur->pLeft;}return cur;}

void Delete(Node*& root, int k){
 if (root == NULL) return;
 if (root->Data > k) Delete(root->pLeft, k);
 else if (root->Data < k)  Delete(root->pRight, k);
 else{if (root->pLeft == NULL)
{ Node* temp = root; root = root->pRight; delete[]temp;}
else if (root->pRight == NULL)
{Node* temp = root;root = root->pLeft;delete[]temp;}
else{
       Node* temp = FindPos(root->pRight);
       root->Data = temp->Data;
       temp->Data = k;
       Delete(root->pRight, k);}}
root->Bal = getBalance(root);//Lấy hệ số cân bằng
int bal = getBalance(root);
//Lệch trái cùng phía
if (bal < -1 && k < root->pLeft->Data)
 root = RightRotate(root);
//Lệch phải cùng phía
else if (bal > 1 && k > root->pRight->Data)
 root = LeftRotate(root);
//Lệch trái khác kía
else if (bal < -1 && k > root->pLeft->Data){
 root->pLeft = LeftRotate(root->pLeft);
 root = RightRotate(root);}
//Lệch phait khác phía
else if (bal > 1 && k < root->pRight->Data){
 root->pRight = RightRotate(root->pRight);
 root = LeftRotate(root);}}

int LeftMostValue(Node* root){
node* current = node;
 while (current && current->left != NULL)
         current = current->left;


     return current;
}
node* deleteNode(struct node* root, int key){
 if (root == NULL) return root;
if (key < root->key) root->left = deleteNode(root->left, key);
else if (key > root->key) root->right = deleteNode(root->right, key);
else{// node with only one child or no child
 if (root->left == NULL){
  struct node* temp = root->right;
  delete[] root; return temp;}
 else if (root->right == NULL) {
  struct node* temp = root->left;
  delete[] root; return temp;}
 struct node* temp = minValueNode(root->right);
// Copy the inorder successor's content to this node
 root->key = temp->key;
// Delete the inorder successor
 root->right = deleteNode(root->right, temp->key);}
 return root;}

void dijkstra(int graph[9][9], int src){
 int dist[9];The output array.  dist[i] will hold the shortest distance from src to i
 bool sptSet[9]; // sptSet[i] true nếu I là đường đi ngắn nhất
 for (int i = 0; i < 9; i++) dist[i] = INT_MAX, sptSet[i] = false;
 dist[src] = 0;// khoảng cách từ nó đến chính nó bằng 0
 // Find shortest path for all vertices
 for (int count = 0; count < 9 - 1; count++) {
 // chọn đỉnh có khoảng cách tối thiểu chưa được xử lý
 int u = minDistance(dist, sptSet);
 sptSet[u] = true; // cập nhật
 for (int v = 0; v < 9; v++)
  if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
  && dist[u] + graph[u][v] < dist[v])
  dist[v] = dist[u] + graph[u][v];}
 printSolution(dist);// print the constructed distance array }
```



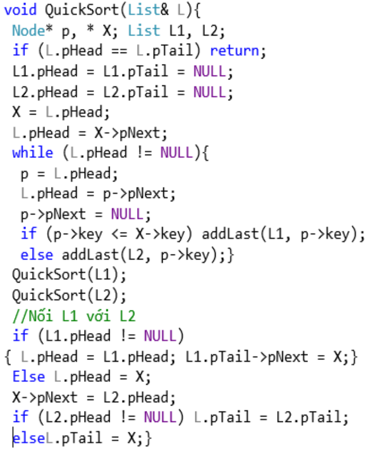
```
void flatten(TreeNode* root) {
 if (!root)
   return;
 if (root->left) flatten(root->left);
 if (root->right) flatten(root->right);
  if(root->left) {
      TreeNode* temp = root->left;
     root->left = nullptr;
   if(!root->right)
        root->right = temp;
    else {
  while(root->right)
      root = root->right;
  root->right = temp;
  }|
```



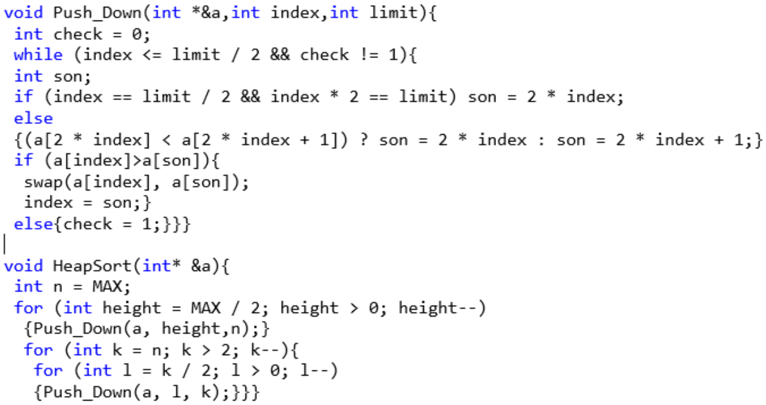
```
bool isValidateBST(Node* root)
{
if (!root)
        return false;
stack<Node*> st;
int inorder = INT_MIN;
bool first = false;
if (root->val == INT_MIN)|
        first = true;
while (true)
{
if (root == NULL && st.empty())
        break;
while (root)
        {
        st.push(root);
        root = root->left;
}
root = st.top();
st.pop();
if (root->val <= inorder && !first)
return false;
inorder = root->val;
root = root->right;
}
return true;
```



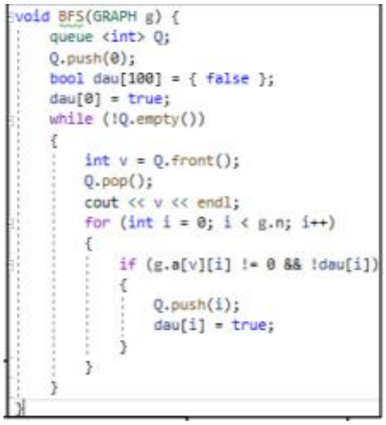
```
int util(TreeNode* root, bool& result)
{
if (root == nullptr)
        return 0;
if (result)
{
        int l = util(root->left, result);
        int r = util(root->right, result);
        if (abs(l - r) > 1) result = false;
        return max(1 + l, 1 + r);
}
else
        return 0;
}|
bool isBalanced(TreeNode* root) {
        bool result = true;
        util(root, result);
        return result;
}
```



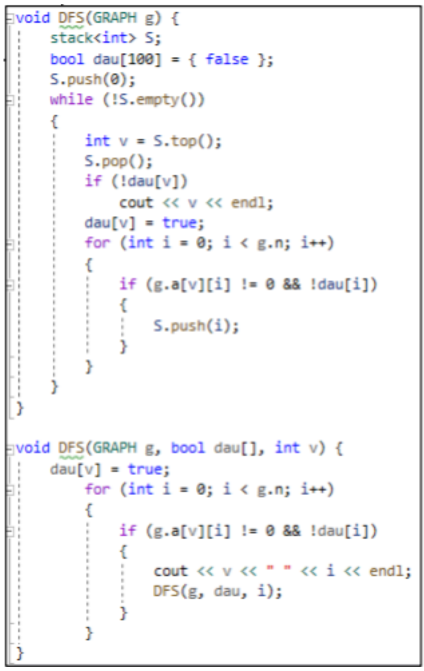
```
void QuickSort(List& L){
 Node* p, * X; List L1, L2;
 if (L.pHead == L.pTail) return;
 L1.pHead = L1.pTail = NULL;
 L2.pHead = L2.pTail = NULL;
 X = L.pHead;
 L.pHead = X->pNext;
 while (L.pHead != NULL){
  p = L.pHead;
  L.pHead = p->pNext;
  p->pNext = NULL;
  if (p->key <= X->key) addLast(L1, p->key);
  else addLast(L2, p->key);}
 QuickSort(L1);
 QuickSort(L2);
 //Nối L1 với L2
 if (L1.pHead != NULL)
{ L.pHead = L1.pHead; L1.pTail->pNext = X;}
 Else L.pHead = X;
 X->pNext = L2.pHead;
 if (L2.pHead != NULL) L.pTail = L2.pTail;
 elseL.pTail = X;}
```



```
void Push_Down(int *&a,int index,int limit){
 int check = 0;
 while (index <= limit / 2 && check != 1){
 int son;
 if (index == limit / 2 && index * 2 == limit) son = 2 * index;
 else
 {(a[2 * index] < a[2 * index + 1]) ? son = 2 * index : son = 2 * index + 1;}
 if (a[index]>a[son]){
  swap(a[index], a[son]);
  index = son;}
 else{check = 1;}}}

void HeapSort(int* &a){
 int n = MAX;
 for (int height = MAX / 2; height > 0; height--)
  {Push_Down(a, height,n);}
  for (int k = n; k > 2; k--){
   for (int l = k / 2; l > 0; l--)
   {Push_Down(a, l, k);}}}
```



```
void BFS(GRAPH g) {
    queue <int> Q;
    Q.push(0);
    bool dau[100] = { false };
    dau[0] = true;
    while (!Q.empty())
    {
        int v = Q.front();
        Q.pop();
        cout << v << endl;
        for (int i = 0; i < g.n; i++)
        {
            if (g.a[v][i] != 0 && !dau[i])
            {
                Q.push(i);
                dau[i] = true;
            }
        }
    }
}
```



```
void DFS(GRAPH g) {
    stack<int> S;
    bool dau[100] = { false };
    S.push(0);
    while (!S.empty())
    {
        int v = S.top();
        S.pop();
        if (!dau[v])
            cout << v << endl;
        dau[v] = true;
        for (int i = 0; i < g.n; i++)
        {
            if (g.a[v][i] != 0 && !dau[i])
            {
                S.push(i);
            }
        }
    }
}

void DFS(GRAPH g, bool dau[], int v) {
    dau[v] = true;
        for (int i = 0; i < g.n; i++)
        {
            if (g.a[v][i] != 0 && !dau[i])
            {
                cout << v << " " << i << endl;
                DFS(g, dau, i);
            }
        }
}
```