CSE 3010 – Data Structures & Algorithms

Lecture #37-38

What will be covered today

- Deleting a node in a BST … Contd.
- Adding parent node for every child node
- Can a binary search tree be represented using an array?
- Announcement
  - Guided practice session on BST and its use on Friday – 28th February 2020

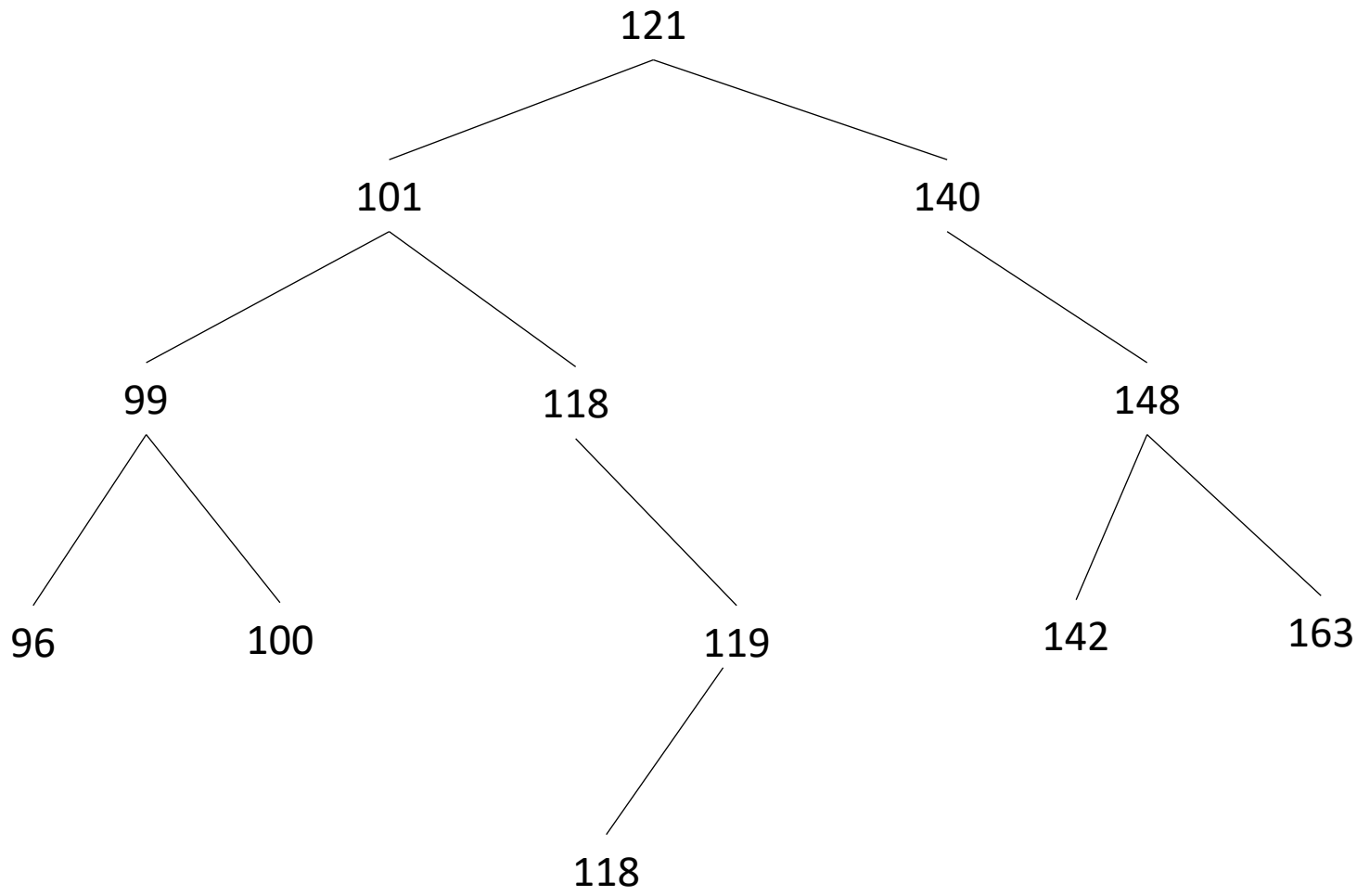
Finding the node with the minimum key

```
// Find the node with the minimum key in the
binary search tree
BSTNODE* findMin(BSTNODE* root) {
    if (root == NULL)
        return NULL;
    else
        if (root->left == NULL)
            return root;
        else
            return findMin(root->left);
}
```

Deleting a node

```
BSTNODE* deleteNode(BSTNODE* root, ITEM key) {
    BSTNODE *temp;
    if (root == NULL)
        return NULL;
    if (key < root->key)
        root->left = deleteNode(root->left,key);
    else
        if (key > root->key)
            root->right = deleteNode(root->right,key);
        else {// This is the node to be deleted
            if (root->left == NULL) {
                temp = root->right;
                free(root);
                return temp;
            }
            else if (root->right == NULL) {
                temp = root->left;
                free(root);
                return temp;
            }
            temp = findMin(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
        return root;
}
```

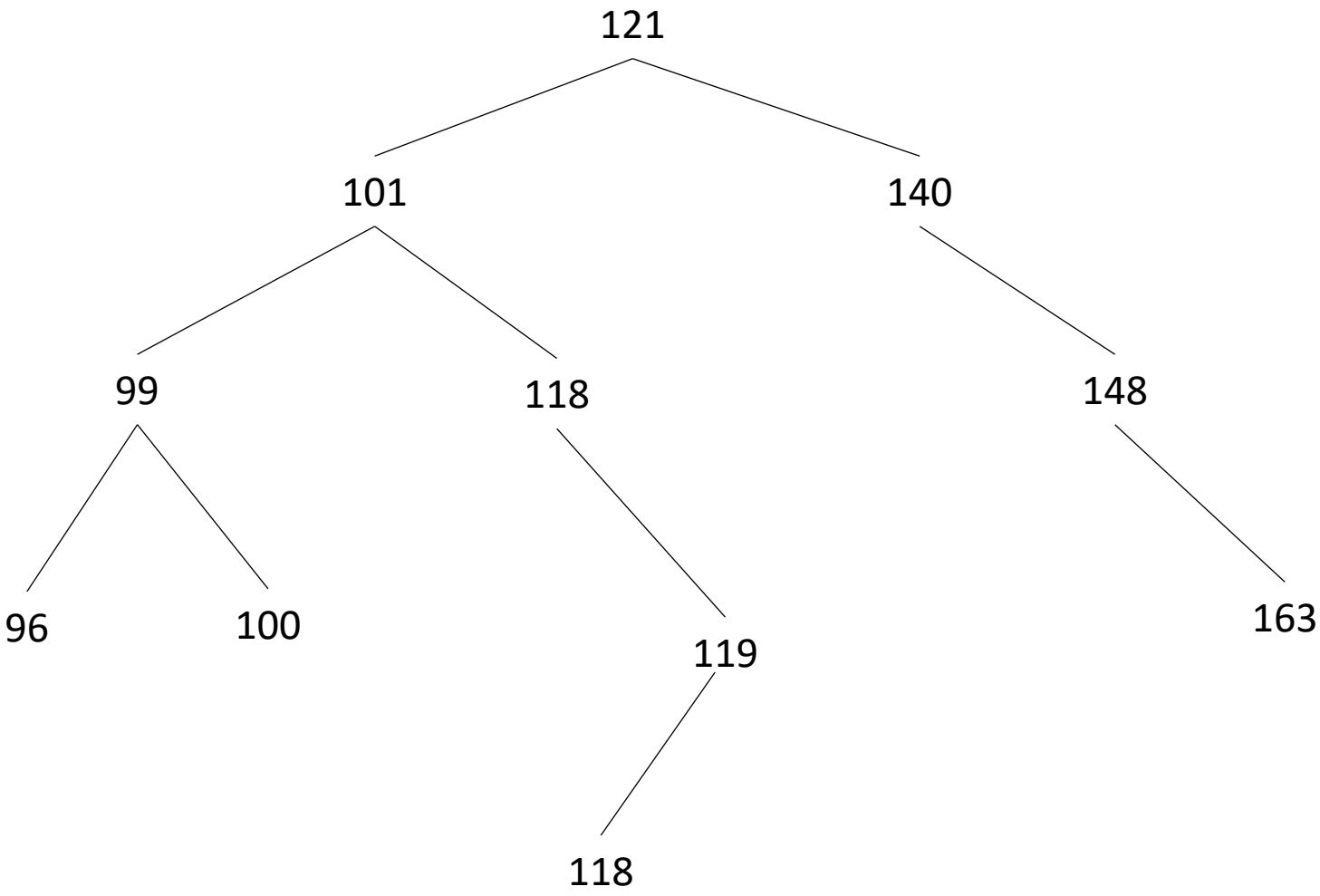
Example tree for reference



Delete node in a binary search tree

deleteNode([121], 142]				
root	**root->left**	**root->right**	**key == root->key**	**temp**
[121]	[101]	[140] =		
[140]	[NULL]	[148] =		
[148]	[142] =	[163]		
[142]	[NULL]	[NULL]	true	= [142]->right = [NULL]
[148]	[NULL]	[163]		
[140]	[NULL]	[148]		
[121]	[101]	[140]		

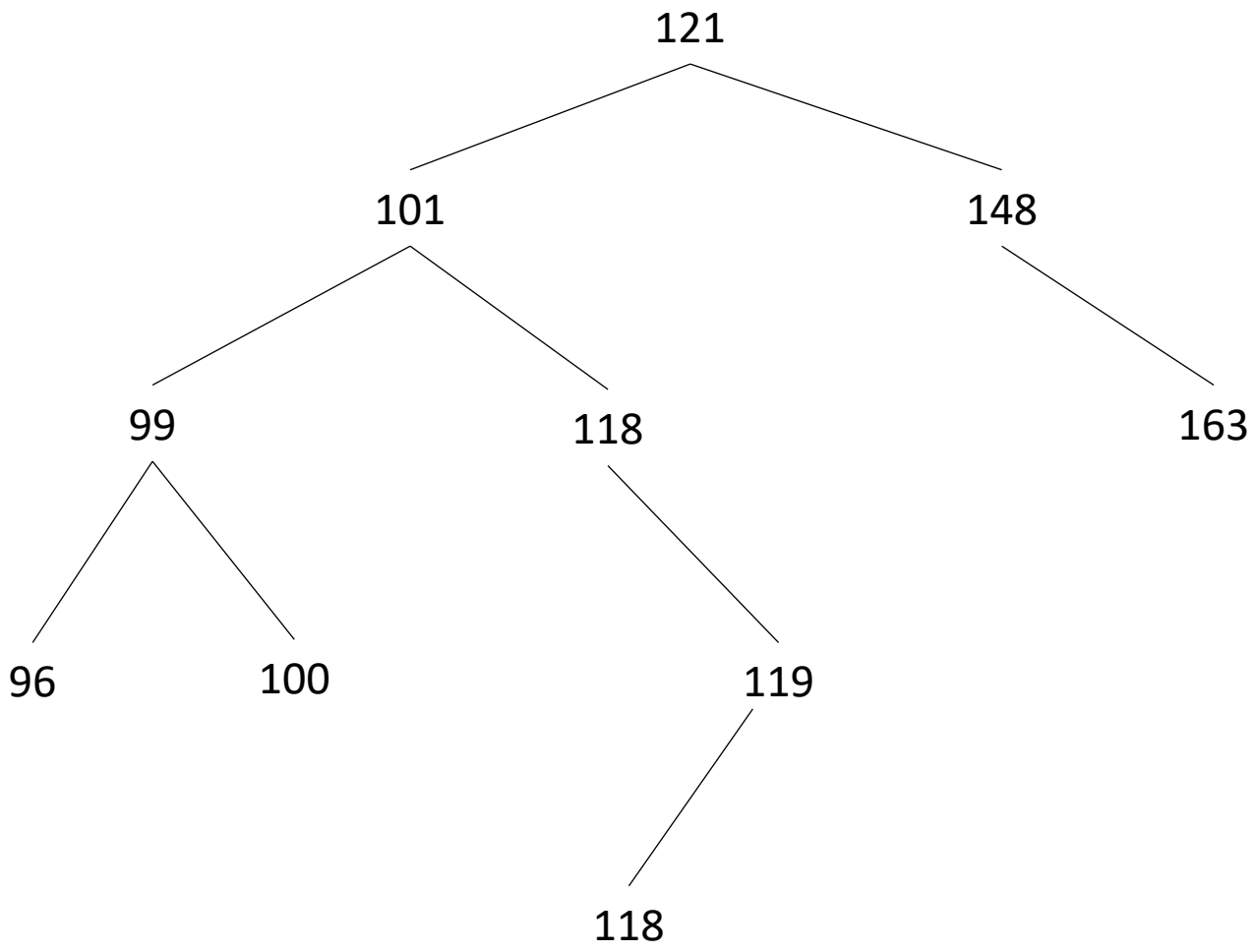
After deleting node with key 142



Delete node in a binary search tree

deleteNode([121], 140]				
root	**root->left**	**root->right**	**key == root->key**	**temp**
[121]	[101]	[140] =		
[140]	[NULL]	[148]	true	= [140]->right = [148]
[121]	[101]	[148]		

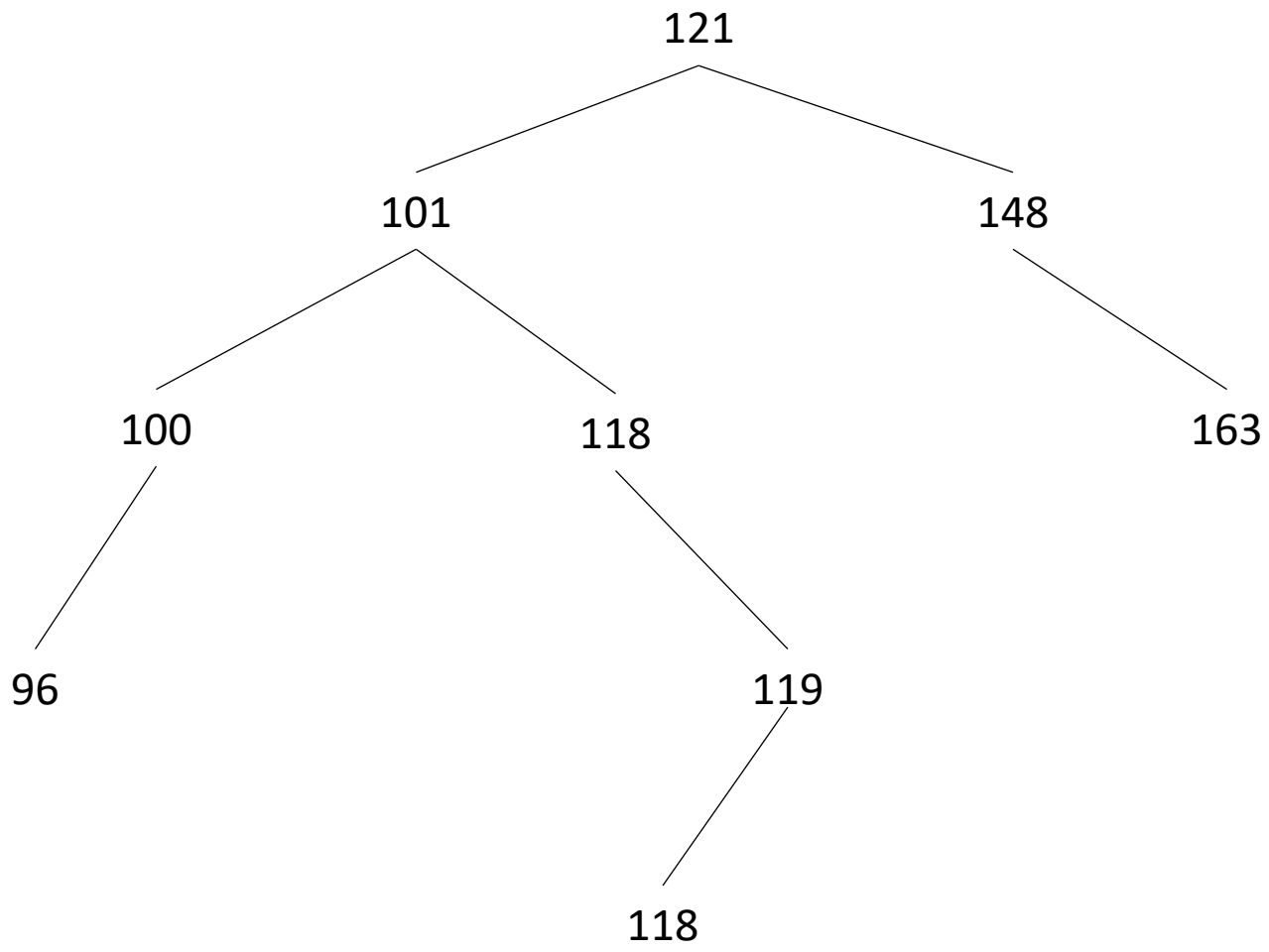
After deleting node with key 140



Delete node in a binary search tree

deleteNode([121], 99]				
root	**root->left**	**root->right**	**key == root->key**	**temp**
[121]	[101] =	[148]		
[101]	[99] =	[118]		
[99] [100]	[96]	[100] =	true	= findMin([99]->right) = [100] [99]->key = 100
deleteNode([100], 100]				
[100]	NULL	NULL	true	= [100]->right = [NULL]
[100]	[96]	NULL		
[101]	[100]	[118]		
[121]	[101]	[148]		

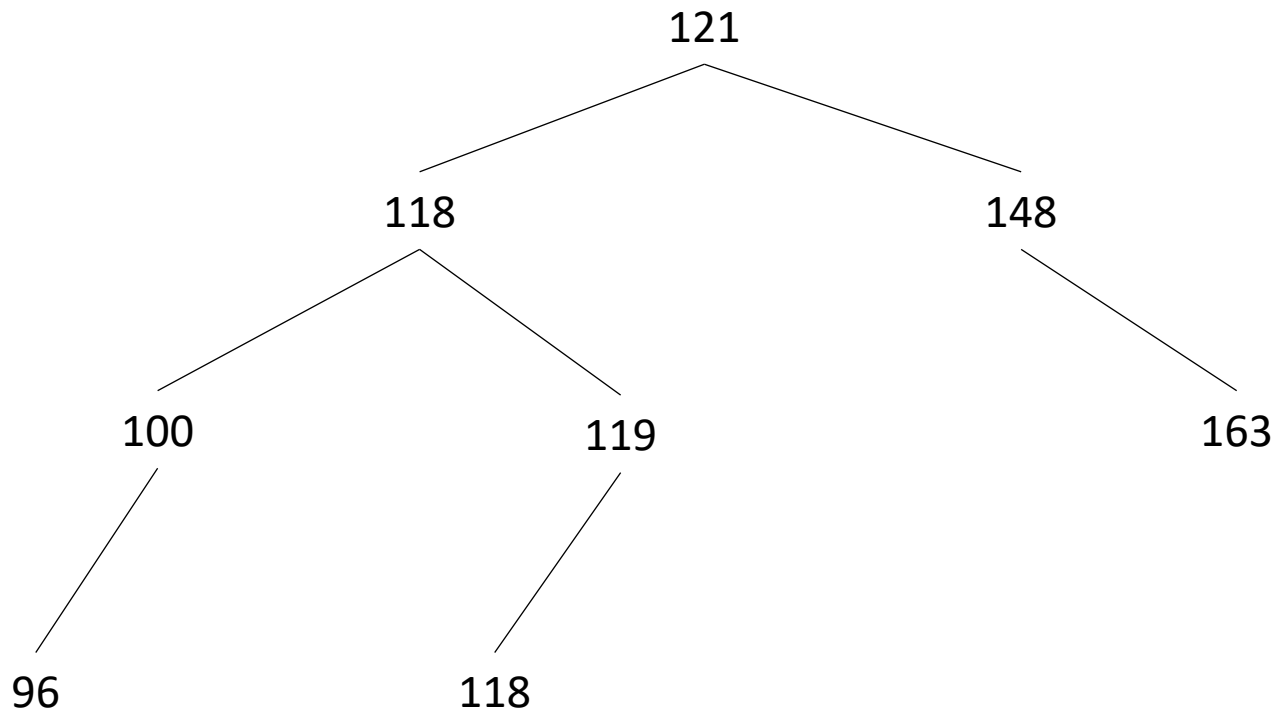
After deleting node with key 99



Delete node in a binary search tree

deleteNode([121], 101]				
root	**root->left**	**root->right**	**key == root->key**	**temp**
[121]	[101] =	[148]		
[101] [118]	[100]	[118] =	true	= findMin([101]->right) = [118] [101]->key = 118
deleteNode([118], 118]				
[118]	NULL	[119] =	true	= [118]->right = [119]
[118]	[100]	[119]		
[121]	[118]	[148]		

After deleting node with key 101



Inserting when parent node is included in every child node

```
// Node in a BST with a pointer to the parent
typedef int ITEM;
typedef struct node {
    ITEM key;
    struct node *left;
    struct node *right;
    struct node *parent;
} BSTNODE;

// Insert an item in the tree
BSTNODE* insertItem(BSTNODE *root, ITEM key) {
    if (root == NULL)
        root = createNode(key);
    else {
        if (key <= root->key) {
            root->left = insertItem(root->left, key);
            root->left->parent = root; // Parent added to the node
        }
        else {
            root->right = insertItem(root->right, key);
            root->right->parent = root; // Parent added to the node
        }
    }
    return root;
}
```

Representation of a BST using an array

- `BSTArray[0]` is the root of the tree
- Initialize the array to a value outside the range of input dataset
- `(2 * i + 1)` will be the left child of node `i`
- `(2 * i + 2)` will be the right child of node `I`
- No left child if initialized-value found at `(2 * i + 1)`
- No right child if initialized-value found at `(2 * i + 2)`