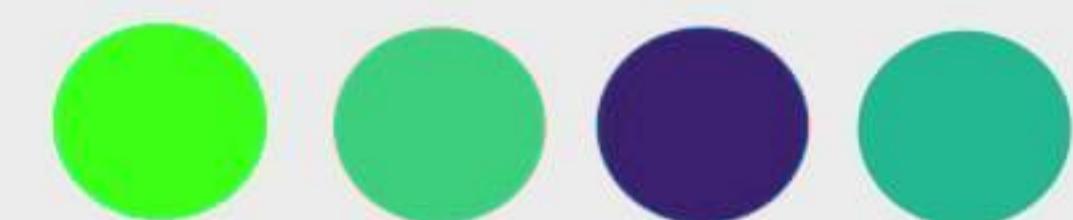




נושאים متقدمים בשפת C



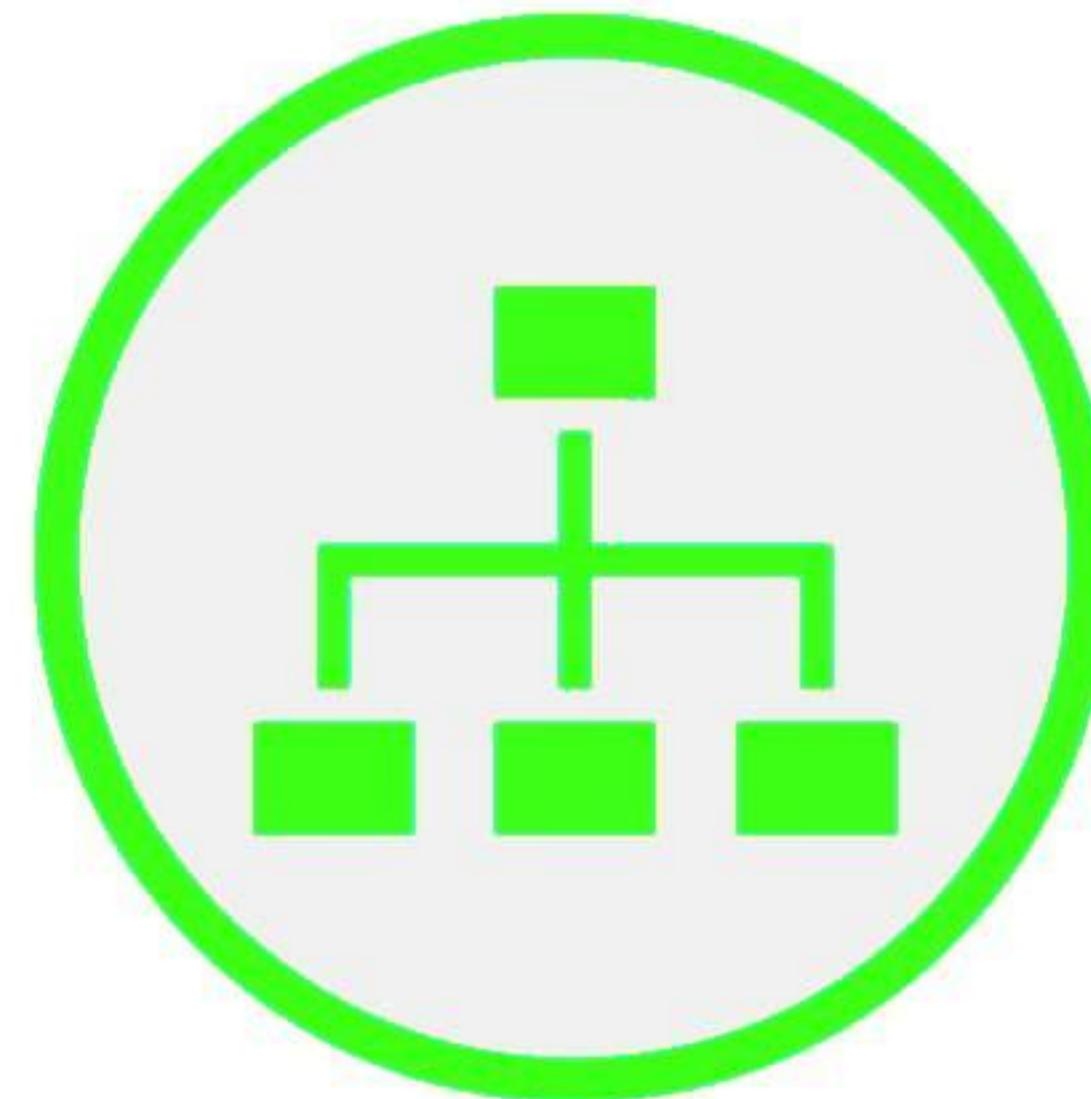
הרצאה 11

מרצה: יורם רענן
yoram.ruppin@gmail.com

פירוט נושאי הקורס וההרצאות



נושאי הרצאה



מבנה מנהלי



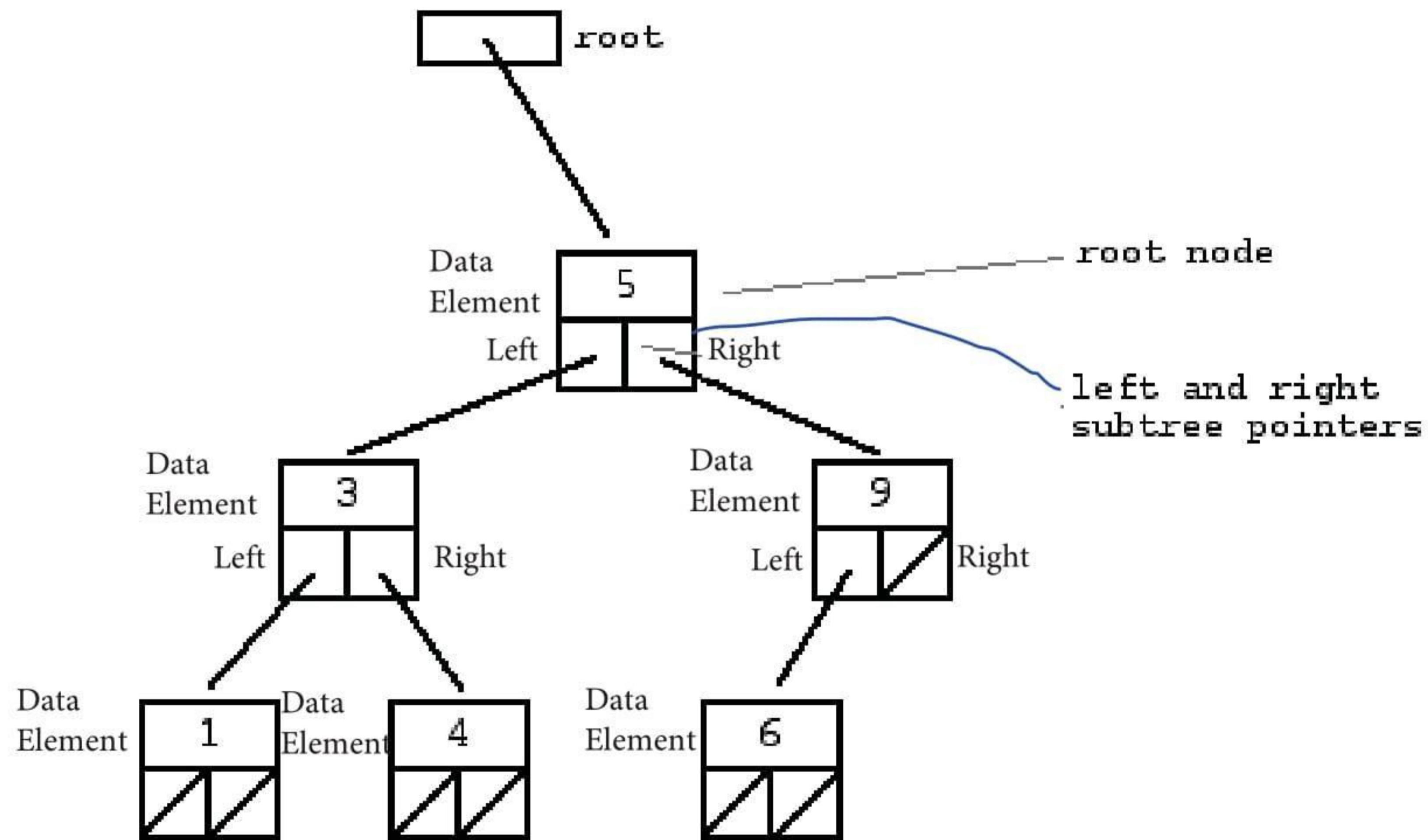
עץ חיפוש בינהי



עץ בינהי (Binary Tree)

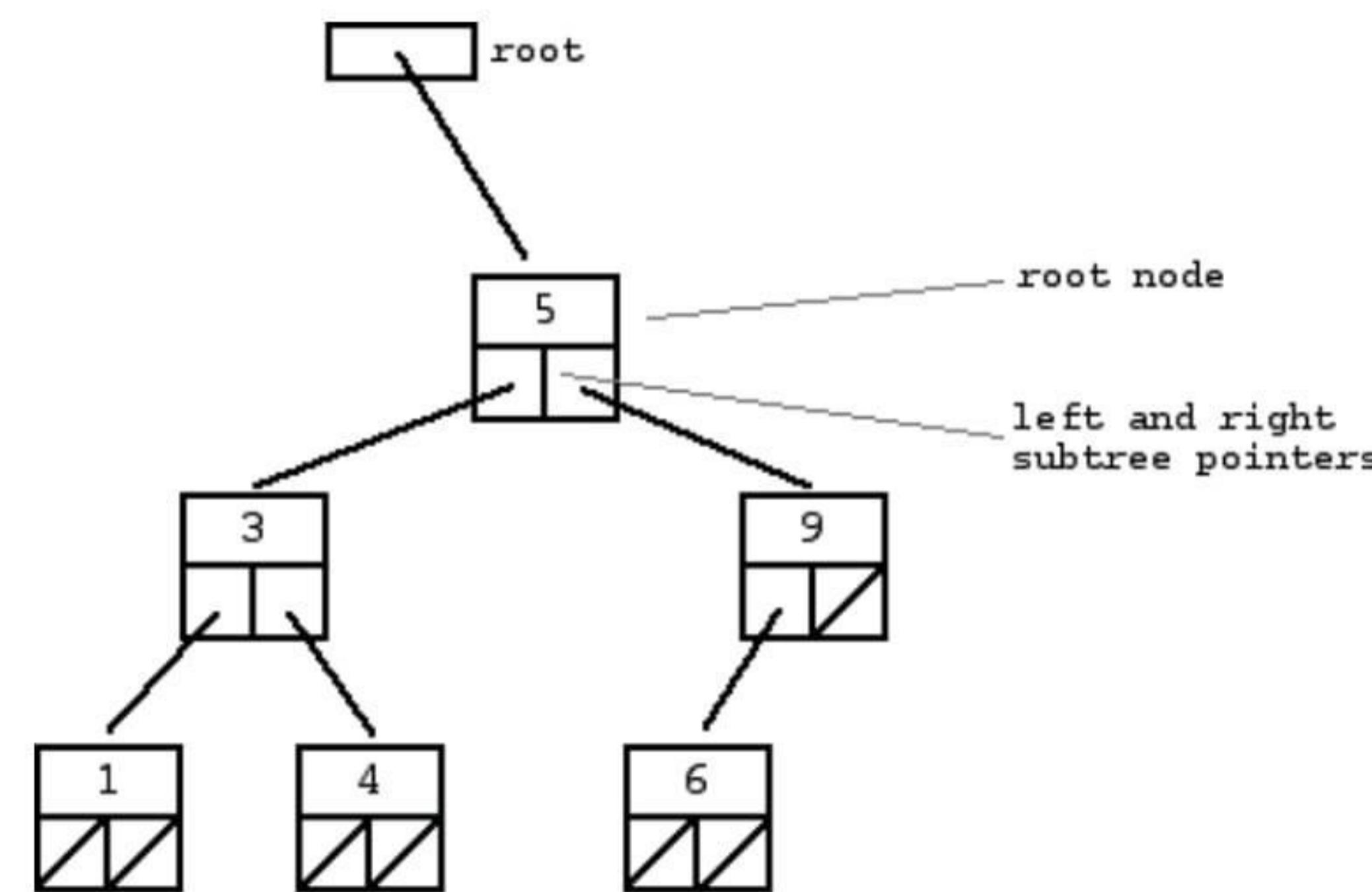
עץ ביארי

- ✓ A binary tree is made of **nodes**, where each node contains a "**left**" pointer, a "**right**" pointer, and a **data element**.
- ✓ The "**root**" pointer points to the topmost node in the tree.



עץ ביארי (המשר)

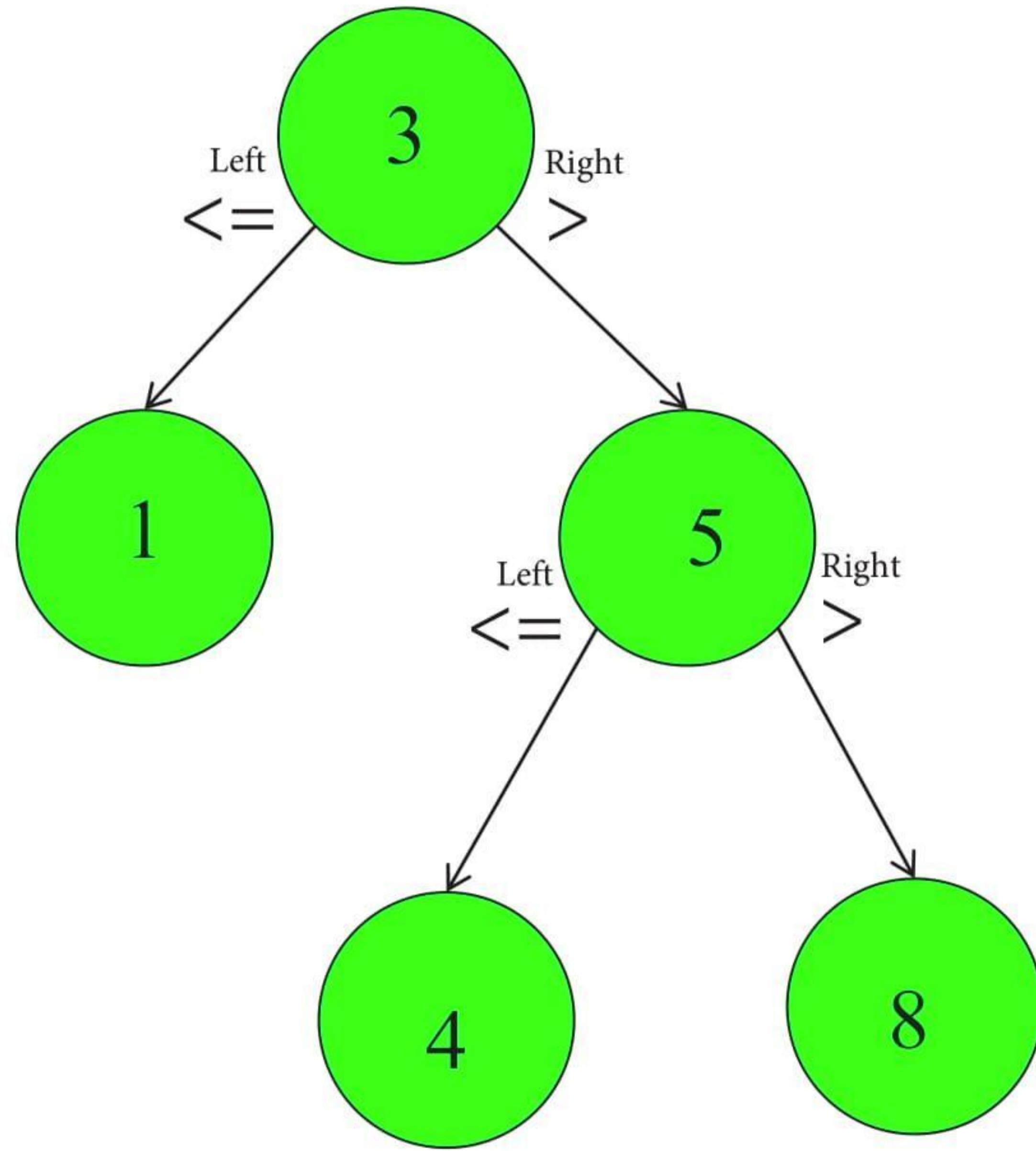
- ✓ The left and right pointers *recursively point* to smaller "subtrees" on either side. A *null pointer* represents a binary tree with no elements -- *the empty tree*.
- ✓ The formal **recursive definition** is:
 - ❖ a **binary tree** is either *empty* (represented by a null pointer),
 - ❖ or is made of a *single node*, where the left and right pointers (recursive definition ahead) each point to a **binary tree**.



עץ חיפוש בינארי

BST הוא עץ מסודר - שמאלו קטן שווה וימינו גדול

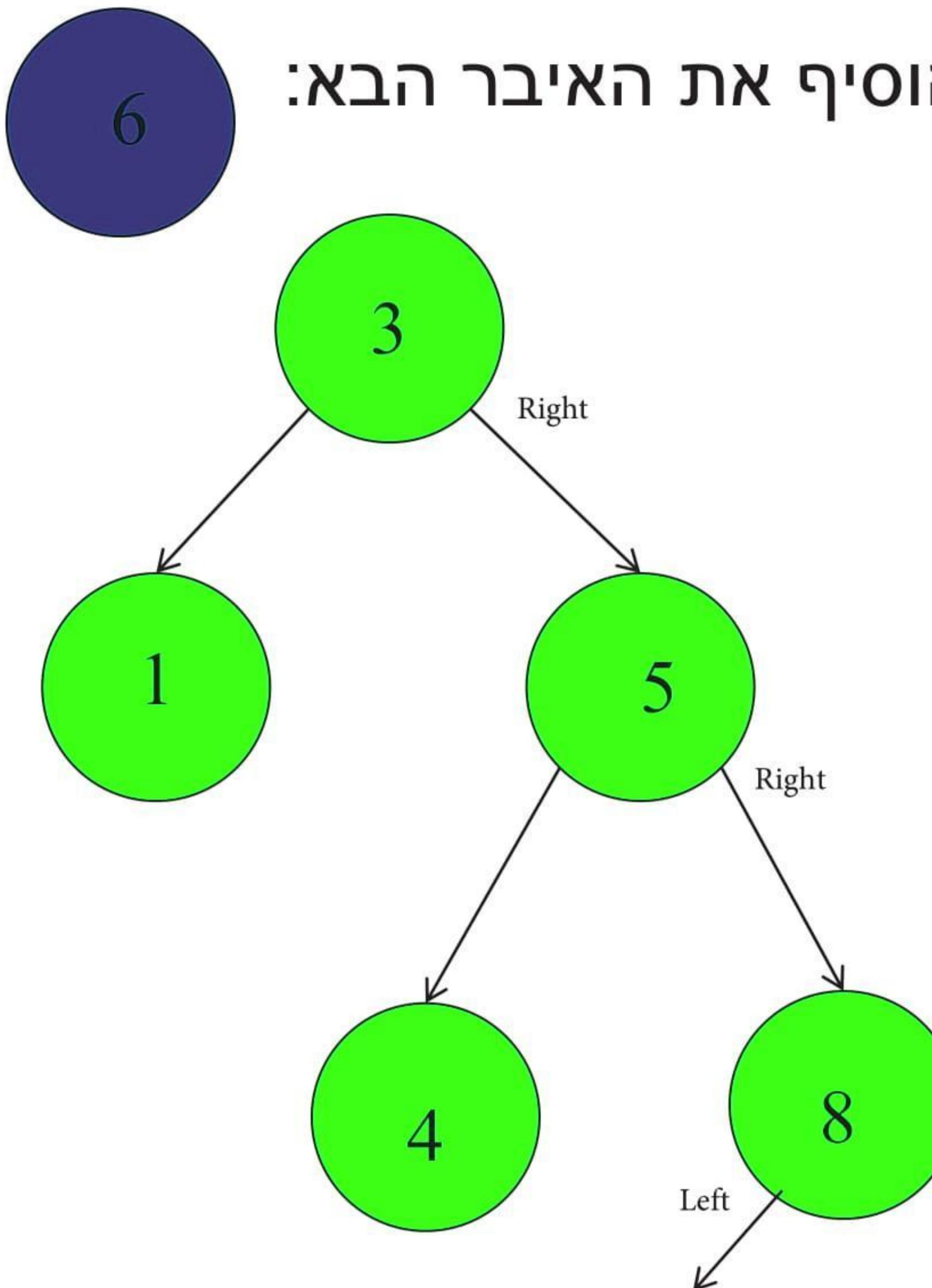
- ✓ A "binary search tree" (BST) or "ordered binary tree" is a type of binary tree where the nodes are arranged in order:
 - ❖ for each node, all elements in its **left subtree** are *less-or-equal to the node* (\leq), and all the elements in its **right subtree** are *greater than the node* ($>$).



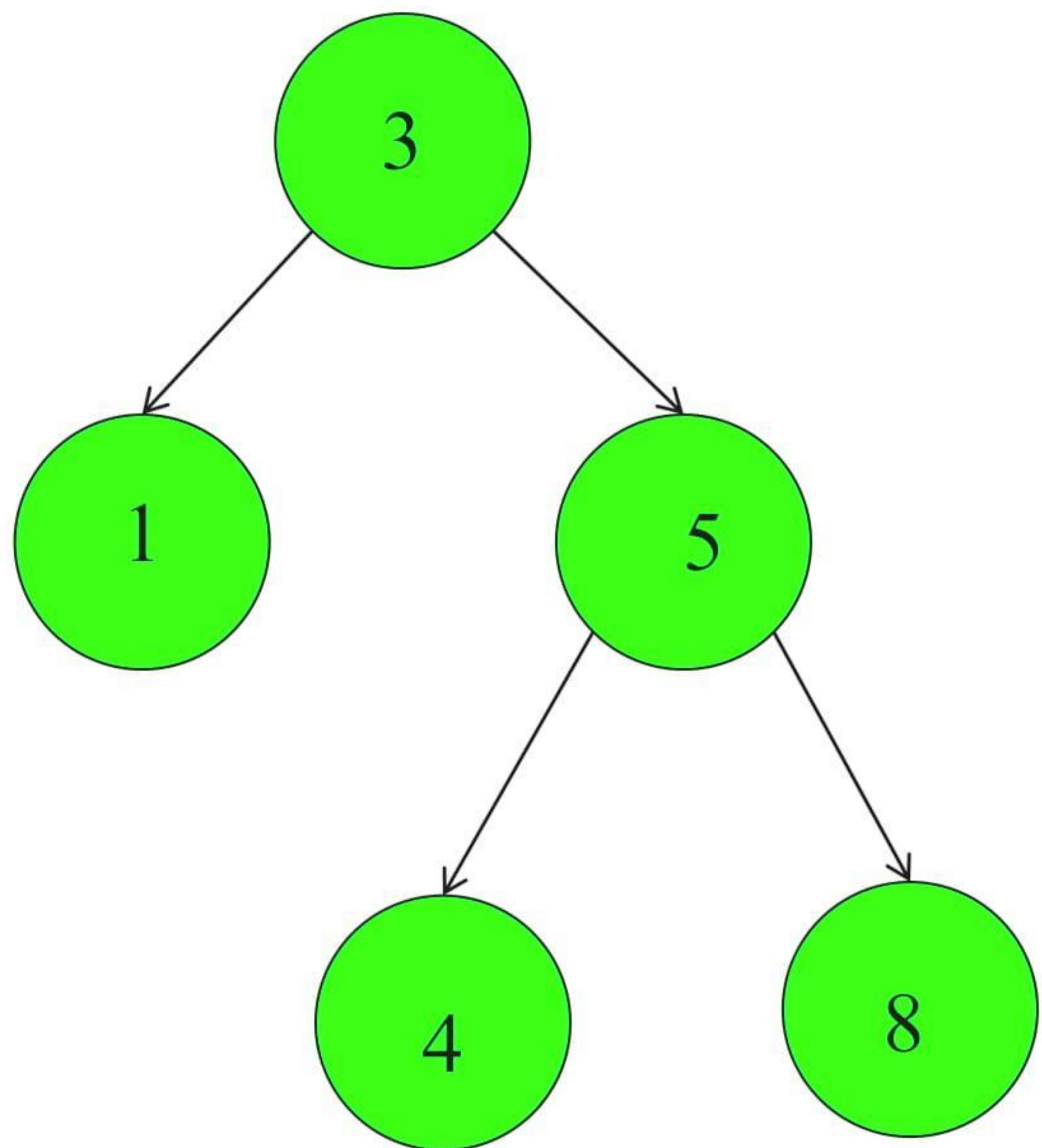
עץ חיפוש בינארי – מושטר הוסיף

נרצה להוסיף את האיבר הבא:

- 6>3 ✓^{Root} נתחילה משורש העץ.
- 6>5 ✓^{Right} נחלחל לעץ הימני.
- 6<=8 ✓^{Right} נחלחל לעץ הימני.
- נוסף את האיבר לענף השמאלי ✓^{Left}



הגדרת המבנה לעץ חיפוש ביארי



איבר בודד בעץ נראה:

```
struct node  
{  
    int data;  
    struct node* left;  
    struct node* right;  
};  
typedef struct node node;
```

פעולות על עצץ חיפוש בינארי

- ✓ יצירת איבר בודד
- ✓ הכנסת איבר למקוםו בעץ – פעולה זו **שומרת על העץ כעץ חיפוש בינארי** (עפ"י ההגדרה הרקורסיבית של BST שראינו קודם)
- ✓ חיפוש איבר בעץ
- ✓ סריקה של העץ (**Traversal** ^{מעבר} עפ"י) סדר – ישן מספר גישות לסדר הסריקה:
preorder, postorder, inorder 

יצירת איבר בודד

- ✓ איבר חדש תמיד נכנס כעלה (leaf) – כאמור שני תת-העצים שלו ריקים!
- ✓ פונקציה מייצרת איבר ללא תוכן
 - מזקה לו מקום
 - משתמש NULL לשני תת-העצים של האיבר החדש

```
node *createNode() {  
    node *temp;  
    temp= (node *) malloc( sizeof (node));  
    temp->left=NULL; // could use calloc instead  
    temp->right=NULL; // could use calloc instead  
    return temp;  
}
```

הfonקציות malloc ו-() calloc משמשות להקצת זיכרון דינמית בשפת התכנות C. ההבדל העיקרי בין הום הוא ש-() calloc תמיד דורשת רק אחד. הfonקציה () calloc מזקה כמות מסוימת של זיכרון ומאתחלת אותו לאפס. ניתן לקבוע את הfonקציה לסוג הרצוי כאשר היא בפועל מוחזרה מצביע ריק null למקום הזיכרון.

ركурсיה - תזכורת

```
int f1(int i){  
    int x=0;  
if (i>0)  
    x=f1(i-1);  
x = x+2;  
    return (x);  
}
```

01 מחושבים הפרמטרים לשילחה (אם יש)-
במקרה זה: 1-i

02 סביבת העבודה (הנתונים) של הפונקציה הקוראת מושהית ונשמרת בזיכרון:

- הנקודה בה ביצוע הקוד הופסק
- ערכם של המשתנים המקומיים

המחשב מתחילה שוב לבצע את הפונקציה מהתחלת עם משתנים מקומיים
חדים ו不同于 המקוריים! שם המשתנים זהה לנסיבות אחרות אבל
התוכן שונה

03 עם סיום החישוב נחזיר להשלים את הפונקציה מהמקום בו הופסקה

הוופת איבר לעץ

```
void insert(node *root, node *newNode)
{
    if(newNode->data <= root->data) //left subtree (<=)
        הגענו לסוף העץ
        if(root->left == NULL)
            root->left = newNode;
        else
            insert(root->left, newNode);

    if(newNode->data > root->data) //right subtree (>)
        הגענו לסוף העץ
        if(root->right == NULL)
            root->right = newNode;
        else
            insert(root->right, newNode);
}
```

- ✓ ביצורה רקורסיבית מATTRים את תת-העץ הנכון.
- ✓ מכניםים את האיבר.

תרגיל

✓ יש להכניס לעץ חיפוש ביןארי את הצמתים עם התוכן הבא (לפי הסדר משמל לימין):

31, 16, 45, 24, 7, 19, 29 ✓



✓ יש לצייר כל אחד מן השלבים.

31, 16, 45, 24, 7, 19, 29

תרגיל (המשר)

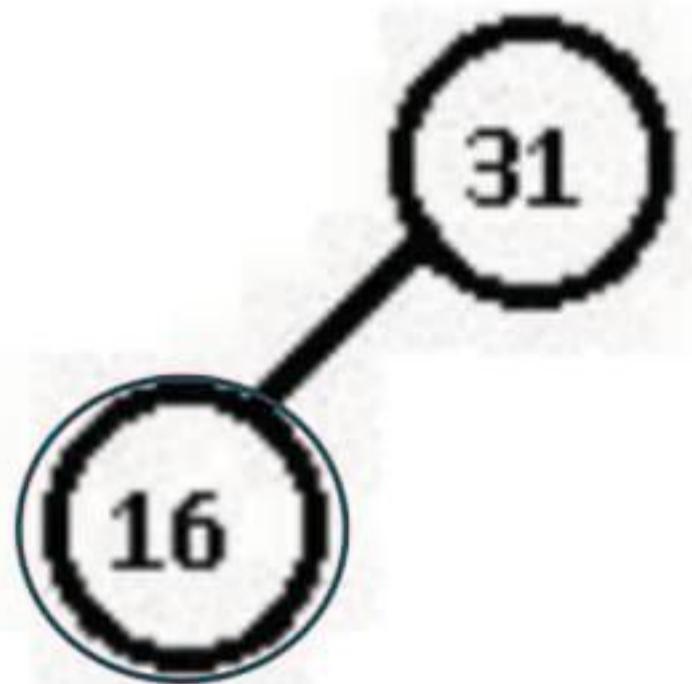
31

Insert 31

31, 16, 45, 24, 7, 19, 29



תרגיל (המשר)



Insert 31

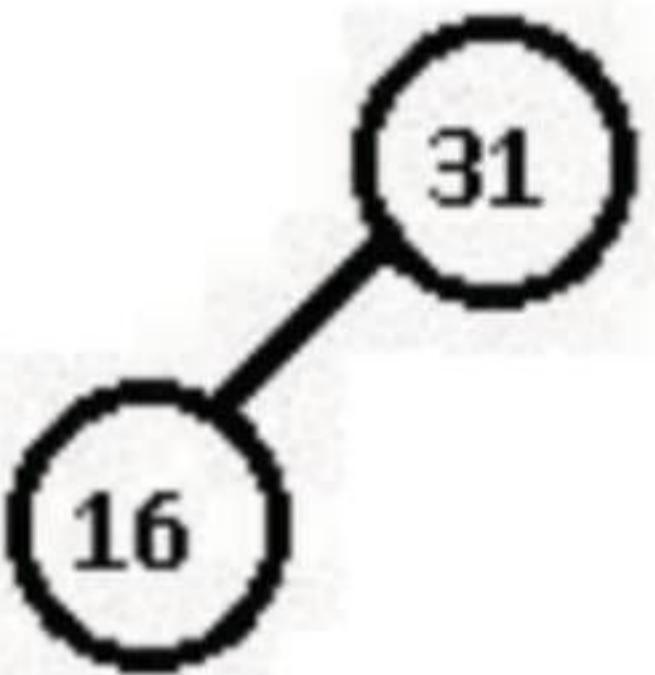
Insert 16

31, 16, 45, 24, 7, 19, 29

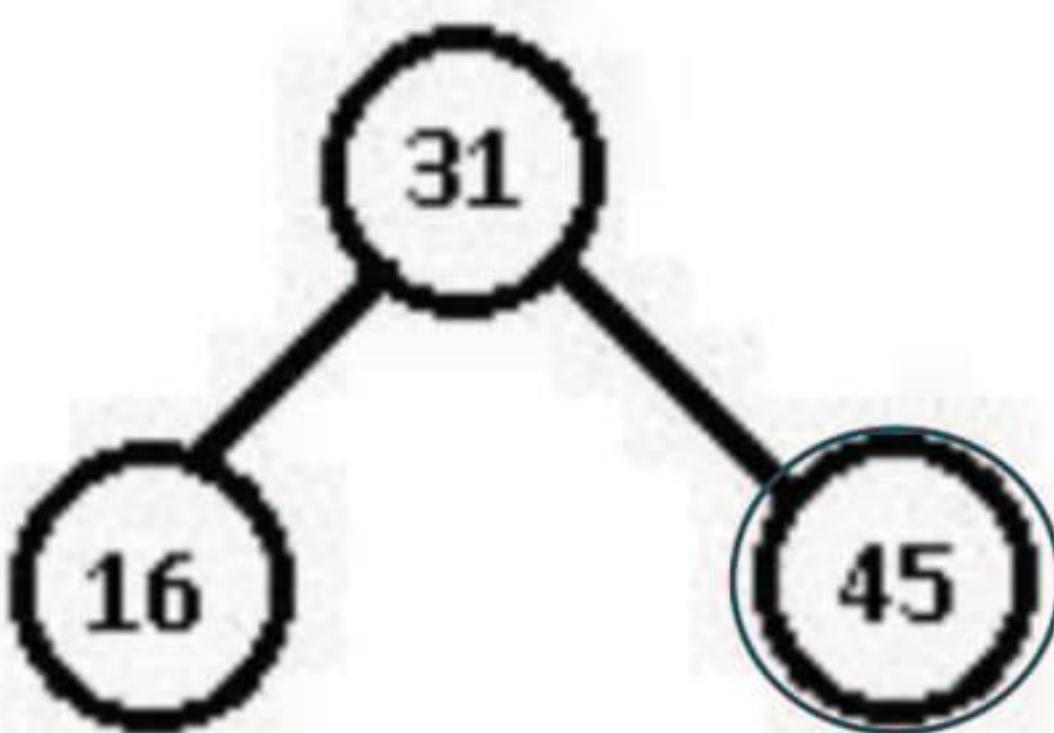
תרגיל (המשר)



Insert 31



Insert 16



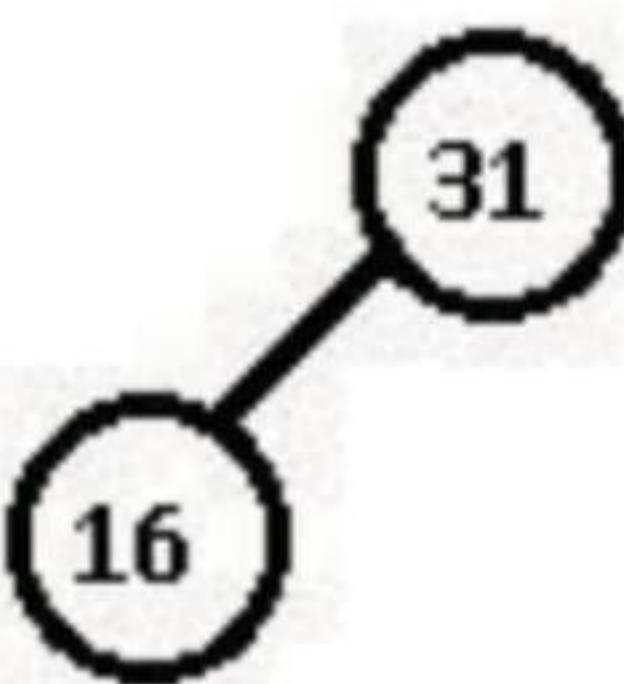
Insert 45

31, 16, 45, 24, 7, 19, 29

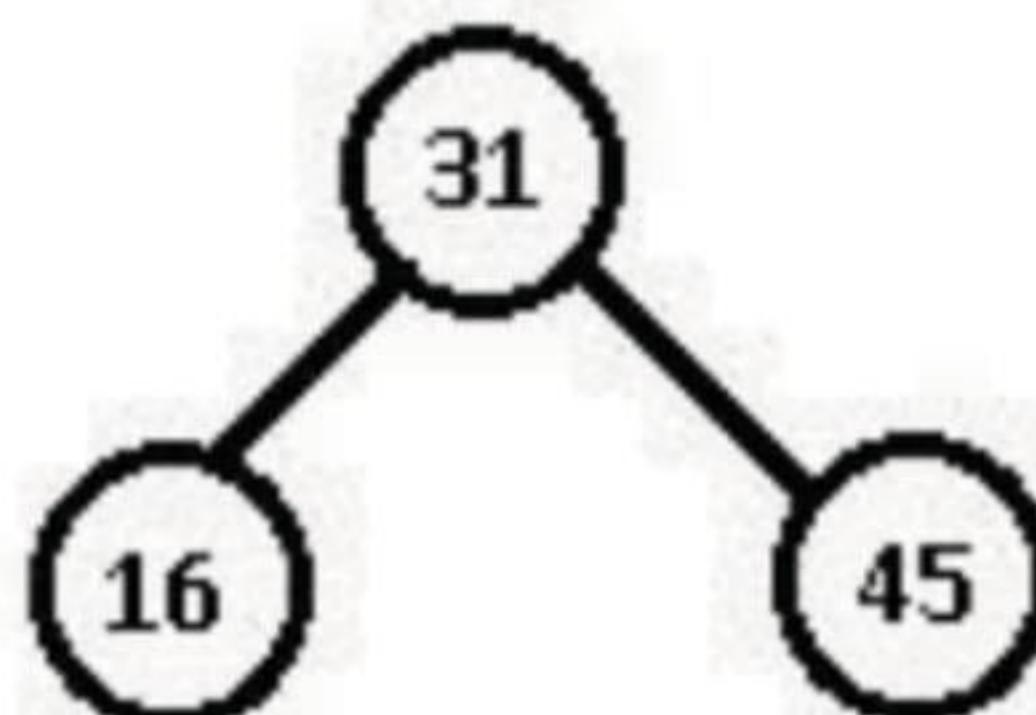
תרגיל (המשר)



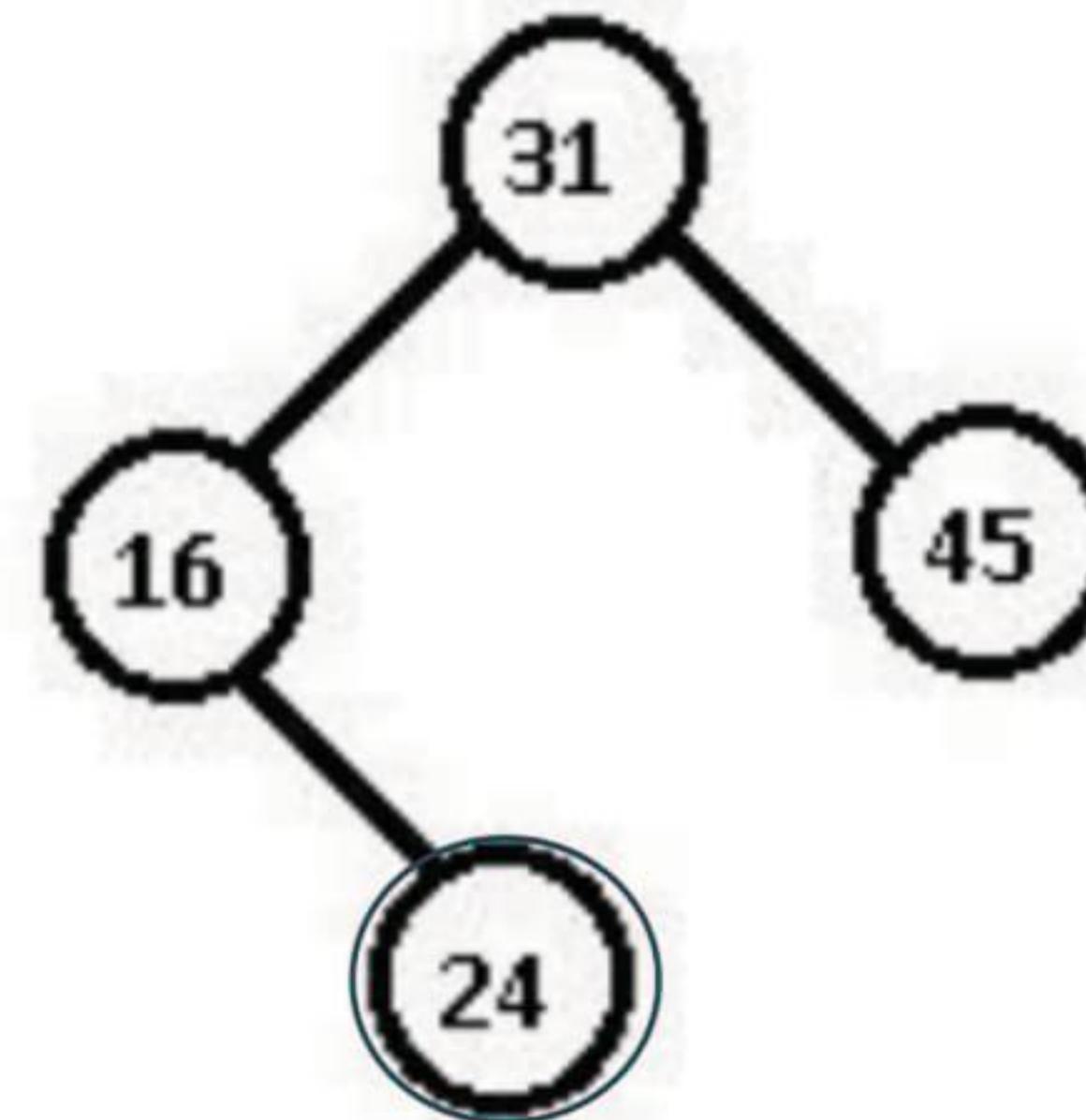
Insert 31



Insert 16



Insert 45



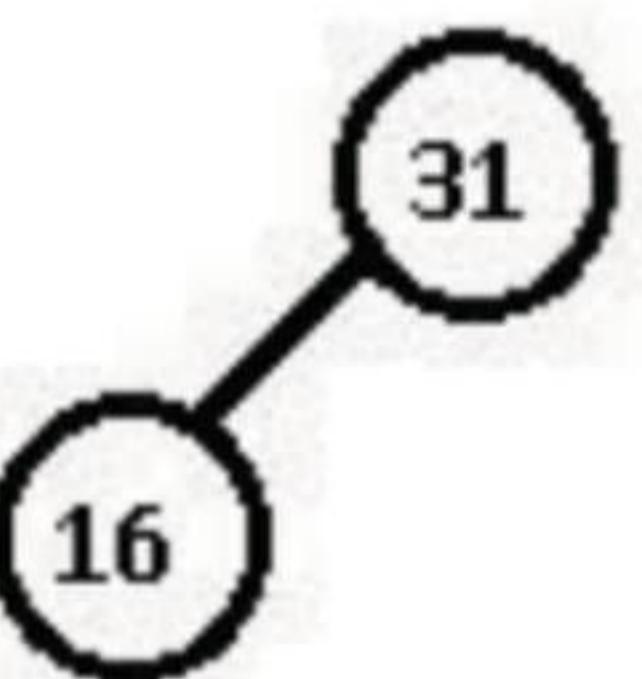
Insert 24

31, 16, 45, 24, 7, 19, 29

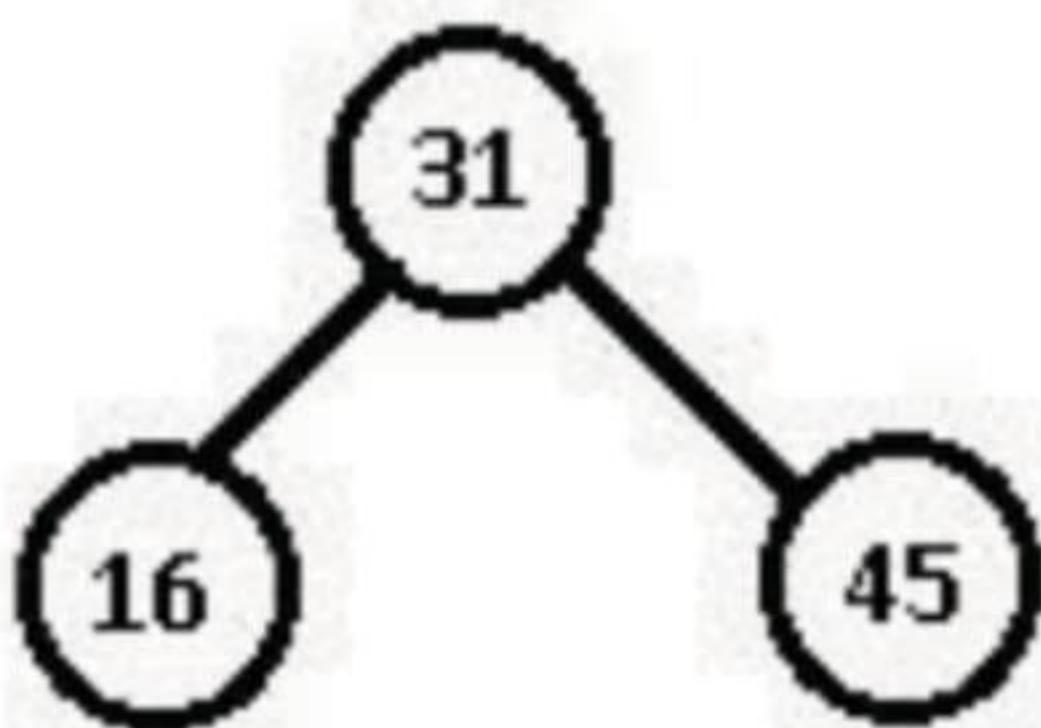
תרגיל (המשר)



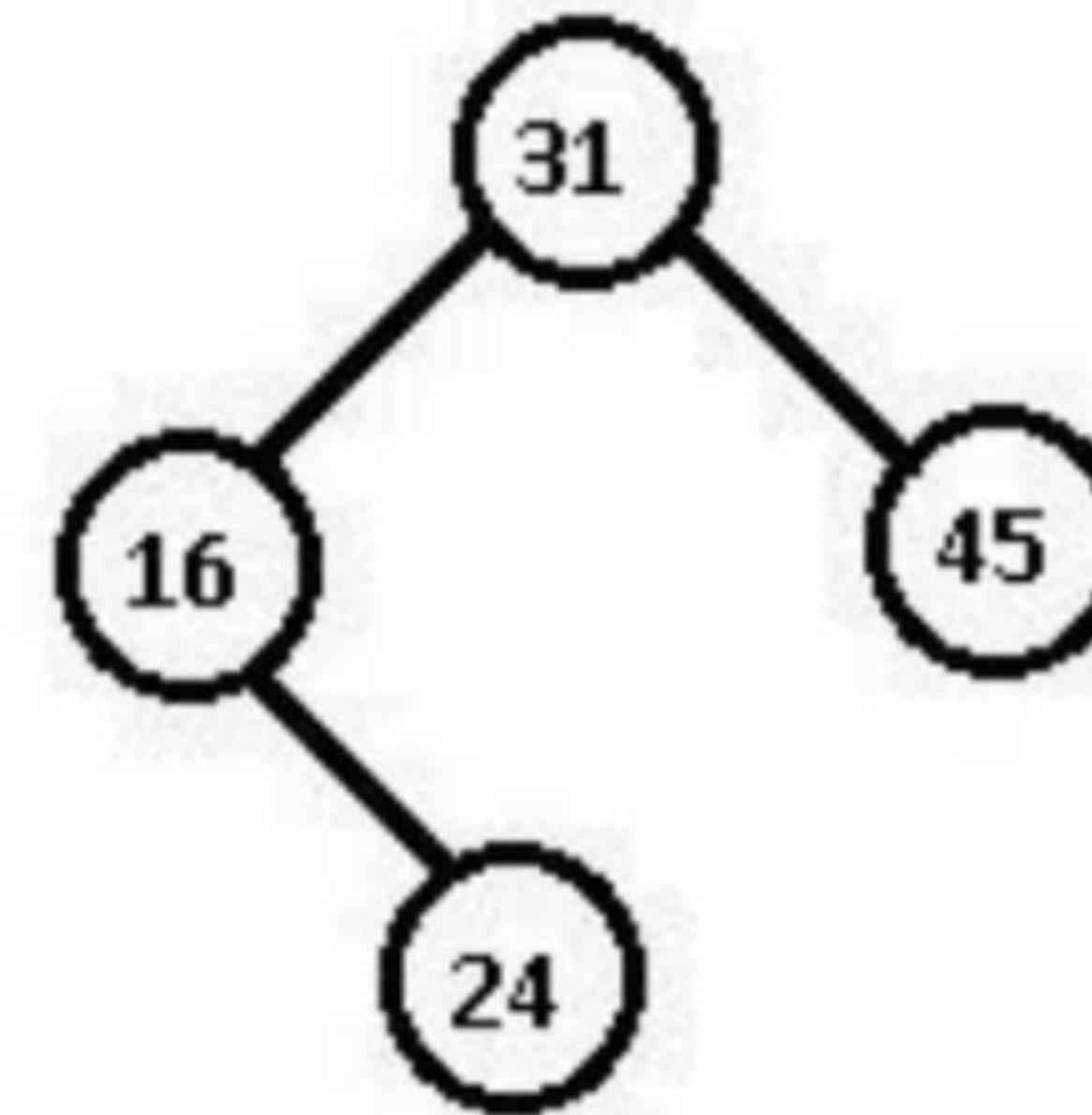
Insert 31



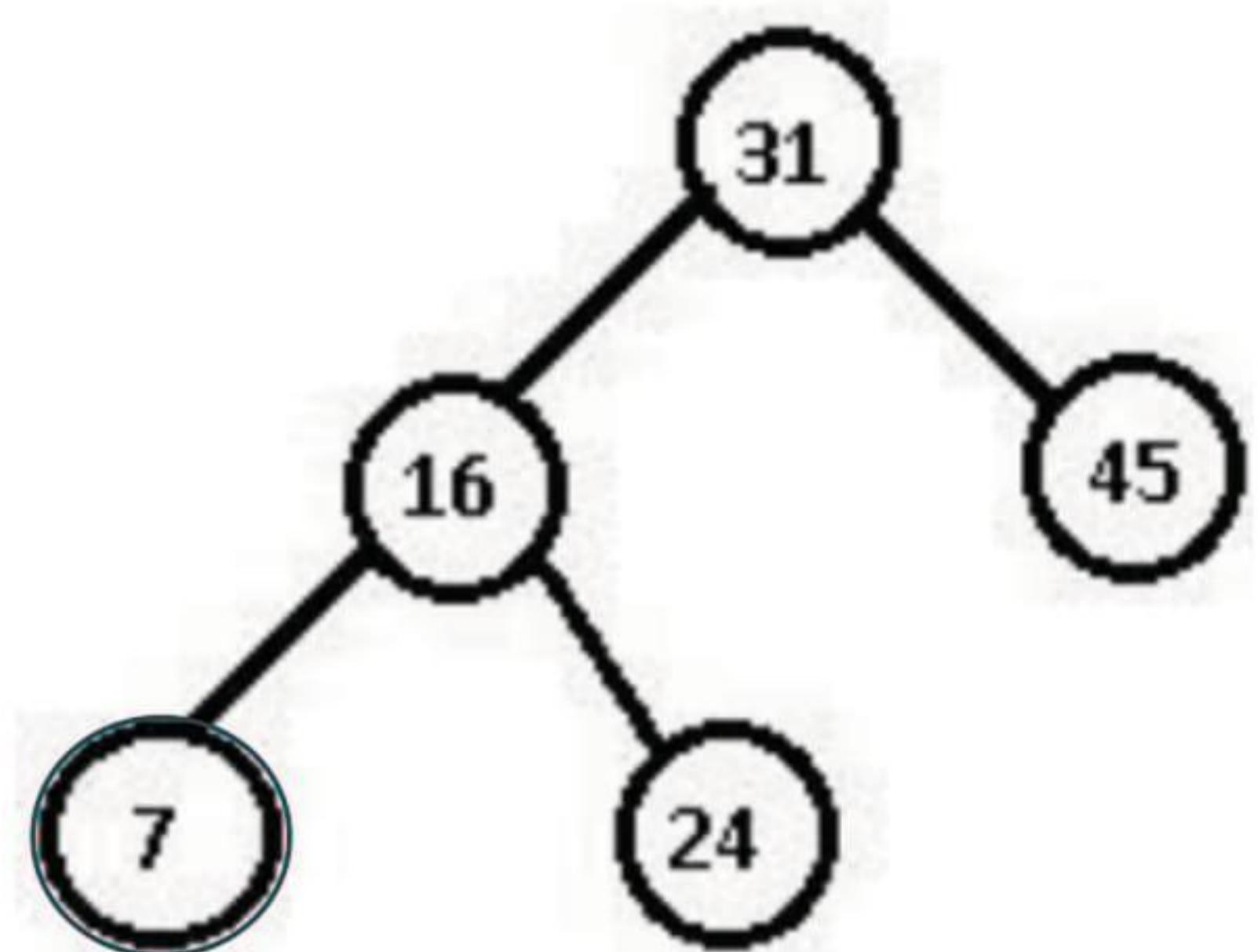
Insert 16



Insert 45



Insert 24



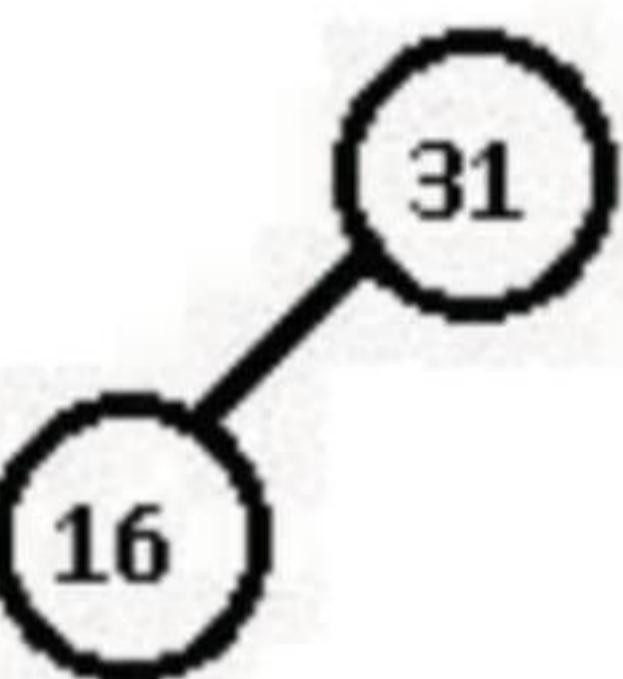
Insert 7

31, 16, 45, 24, 7, 19, 29

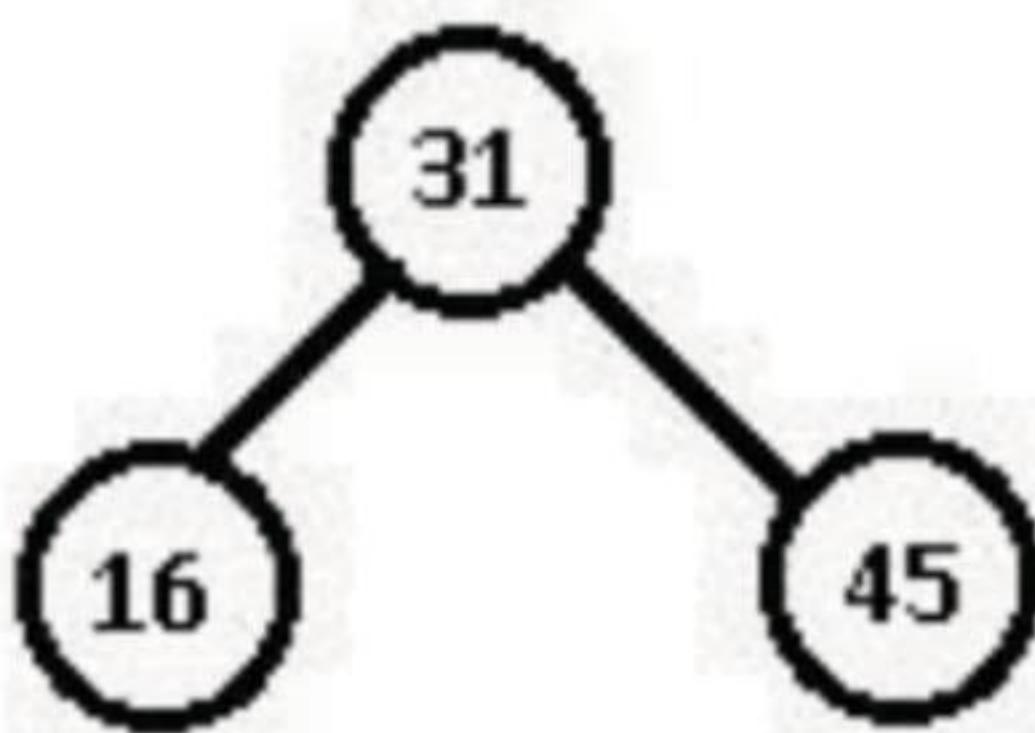
תרגיל (המשר)



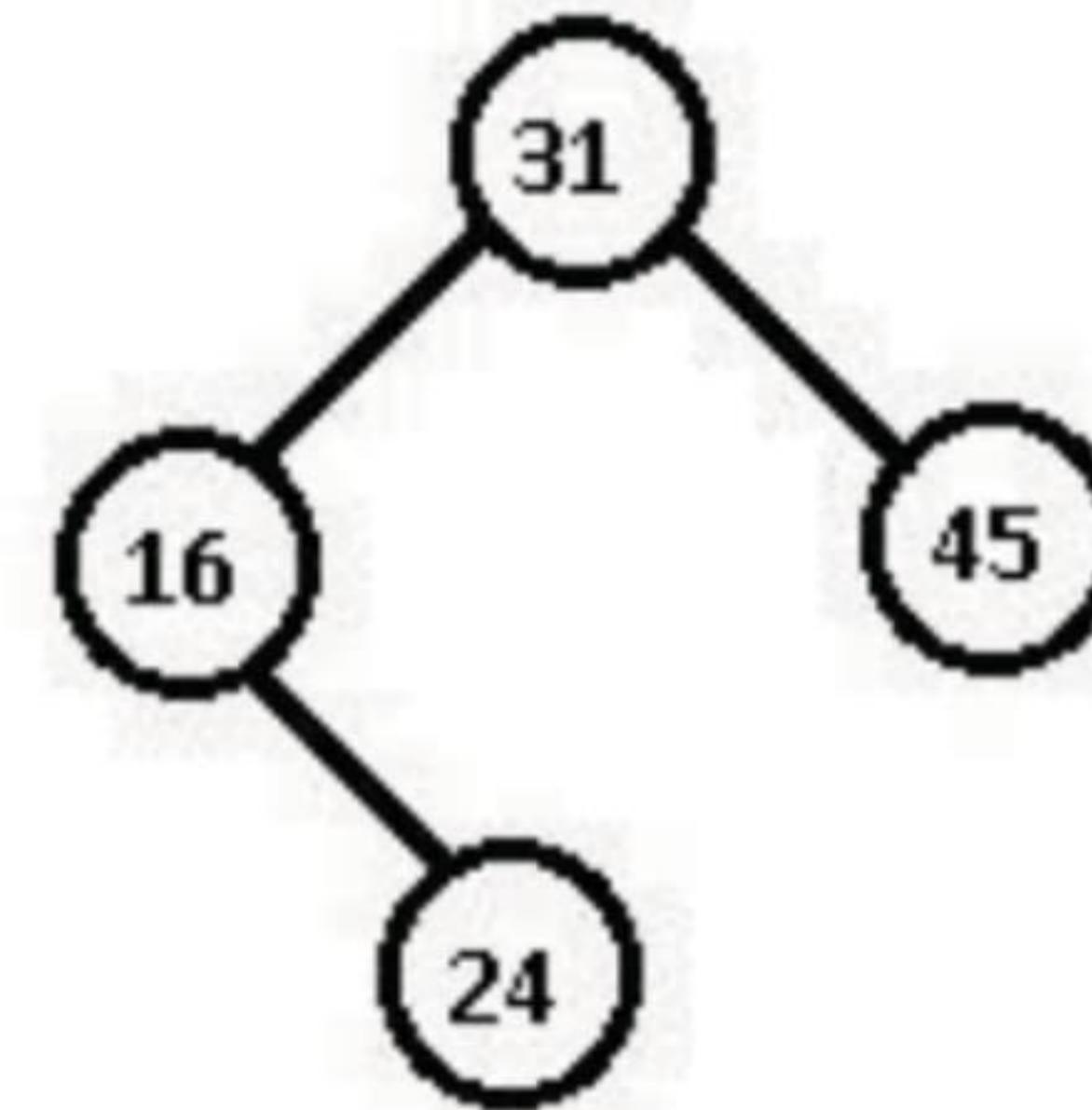
Insert 31



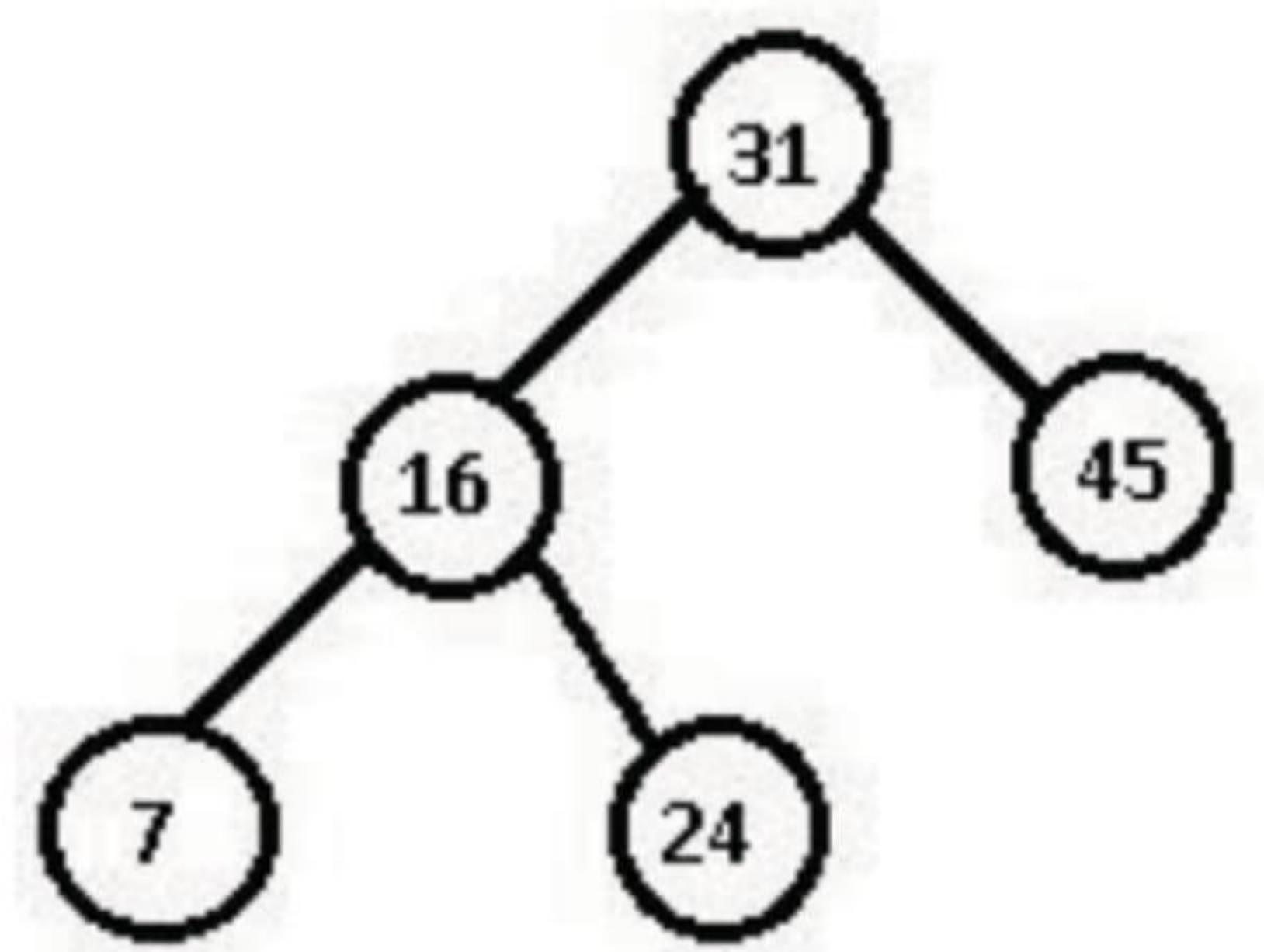
Insert 16



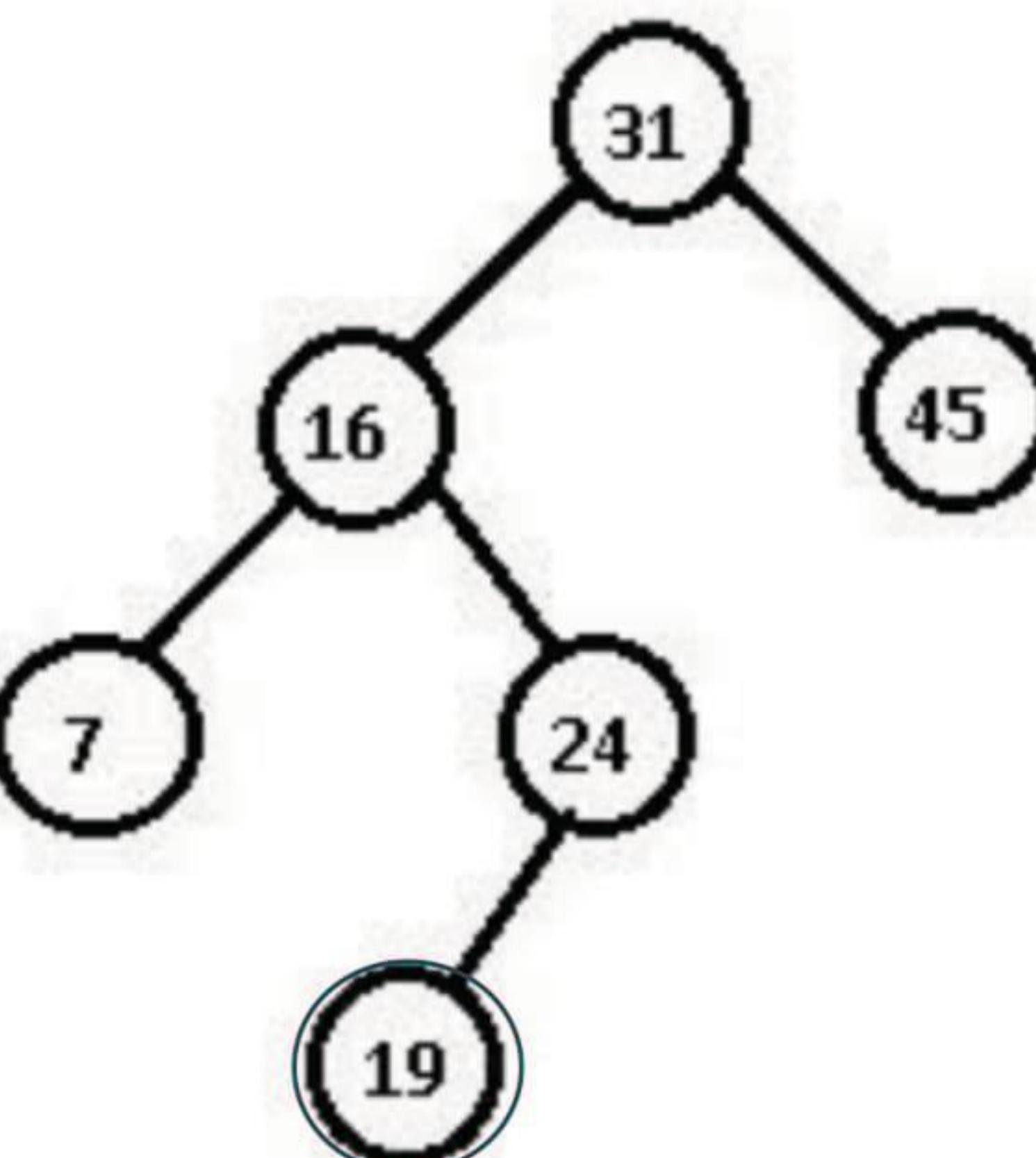
Insert 45



Insert 24



Insert 7



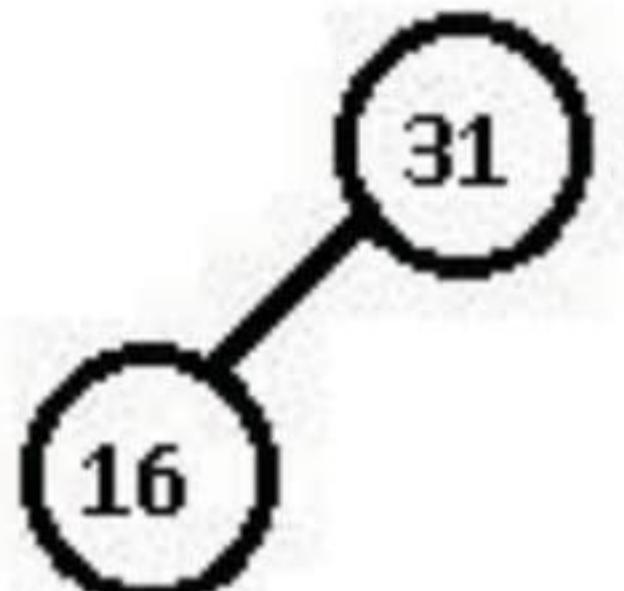
Insert 19

31, 16, 45, 24, 7, 19, 29

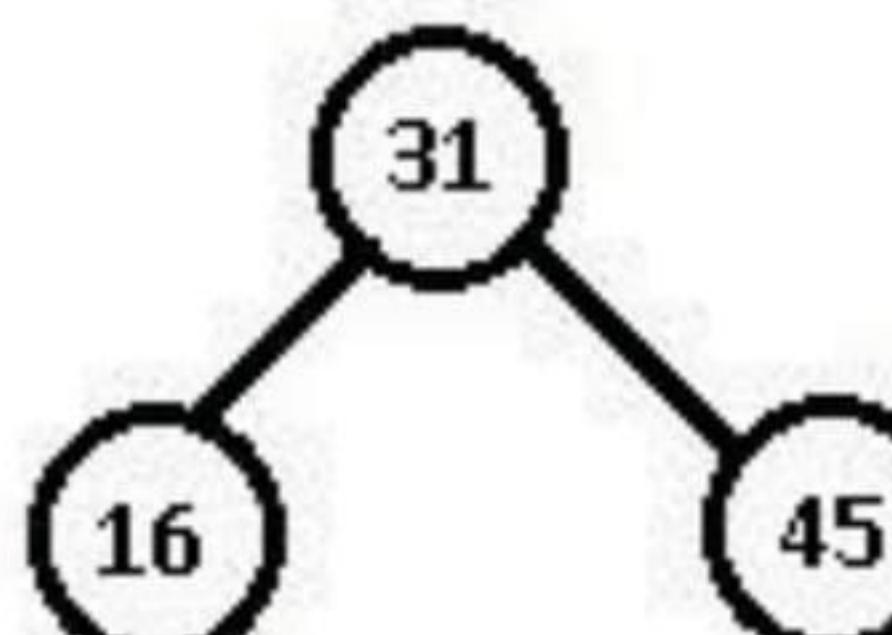
תרגיל (המשר)



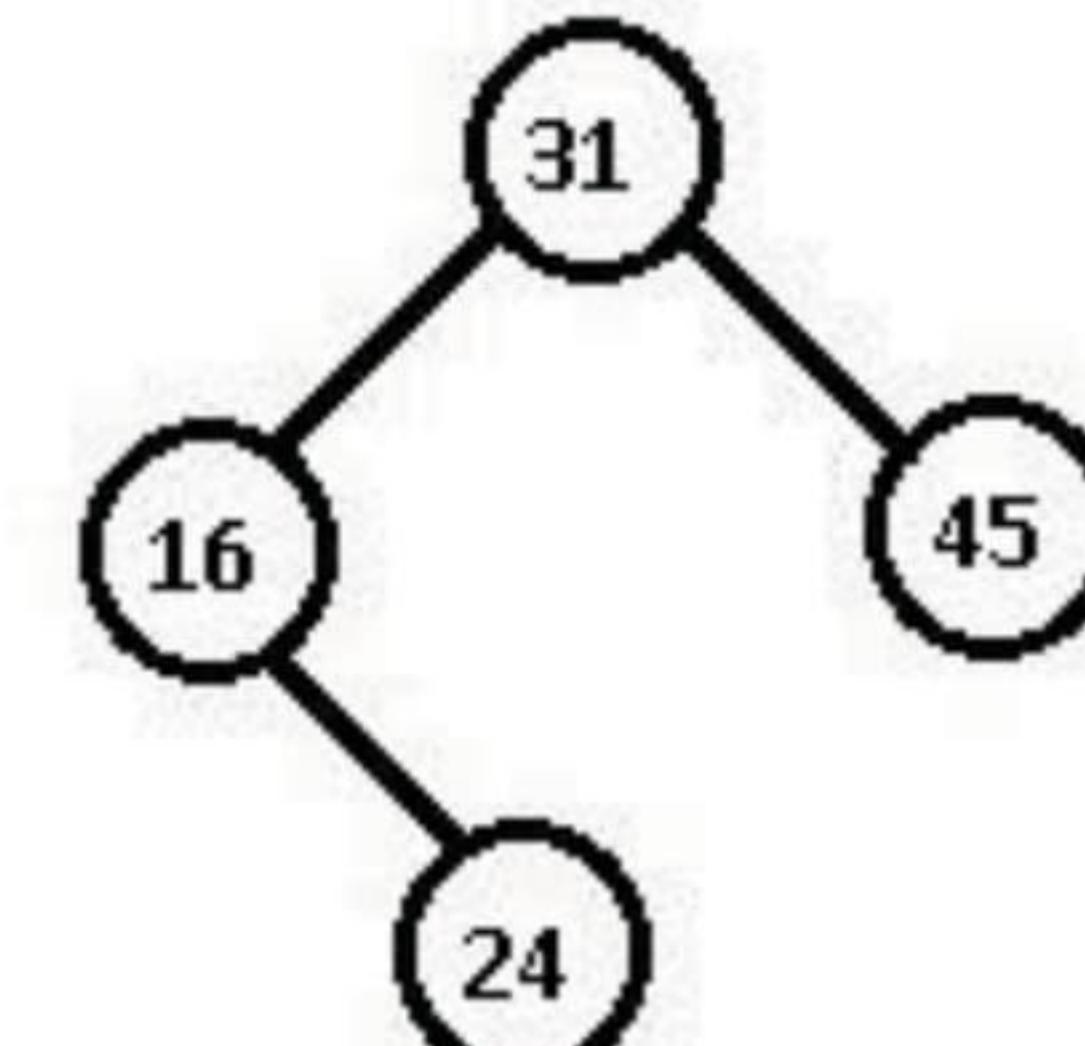
Insert 31



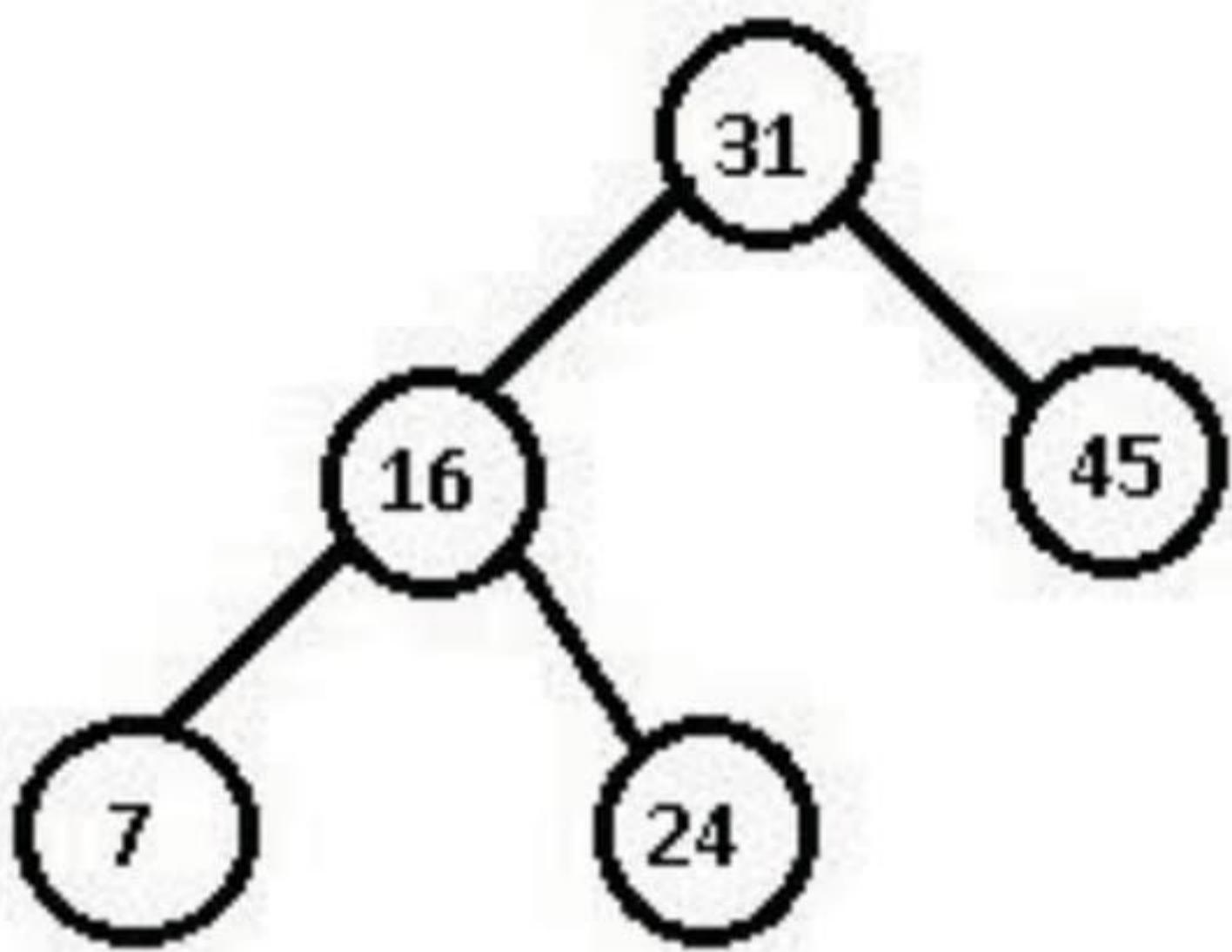
Insert 16



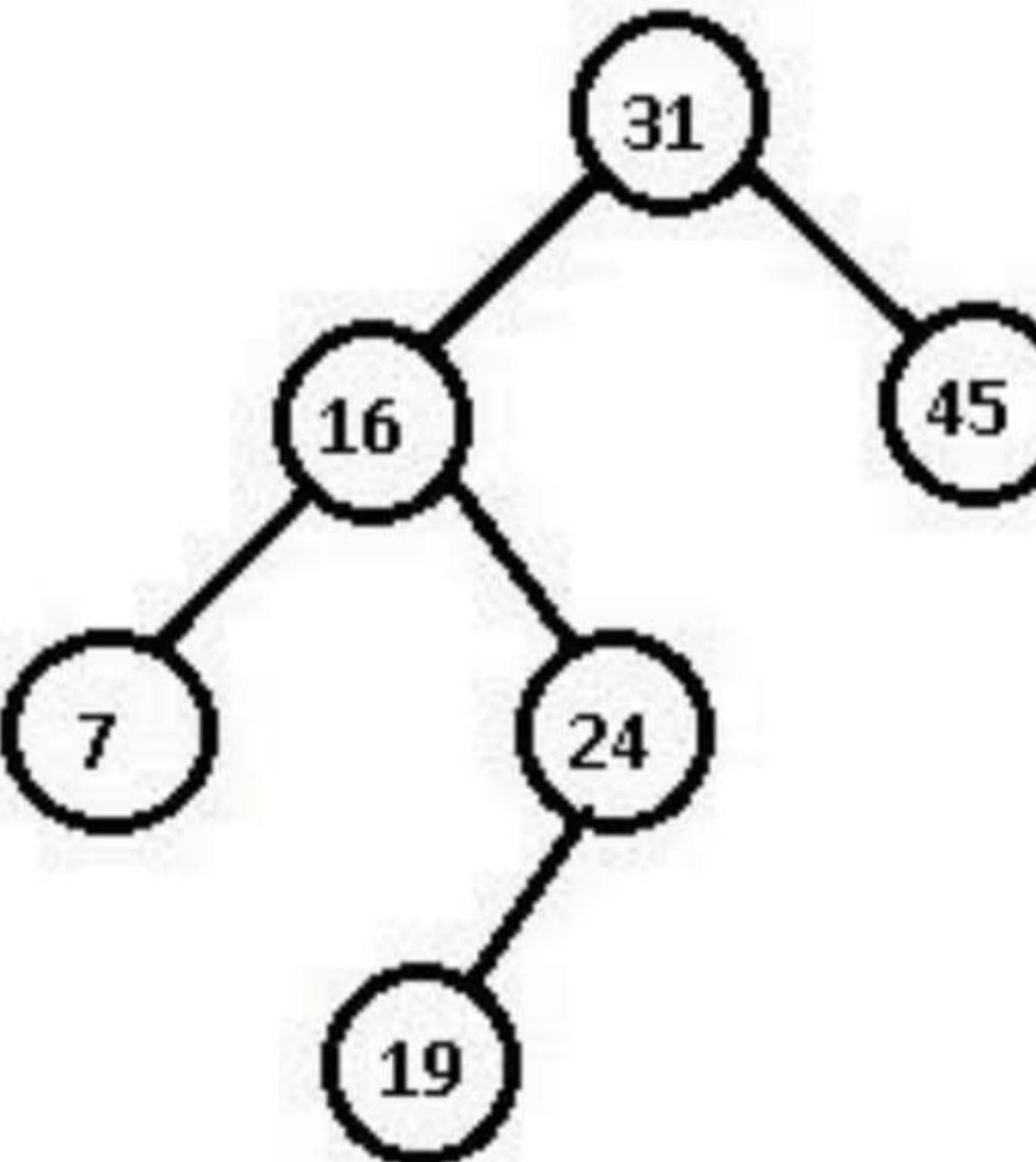
Insert 45



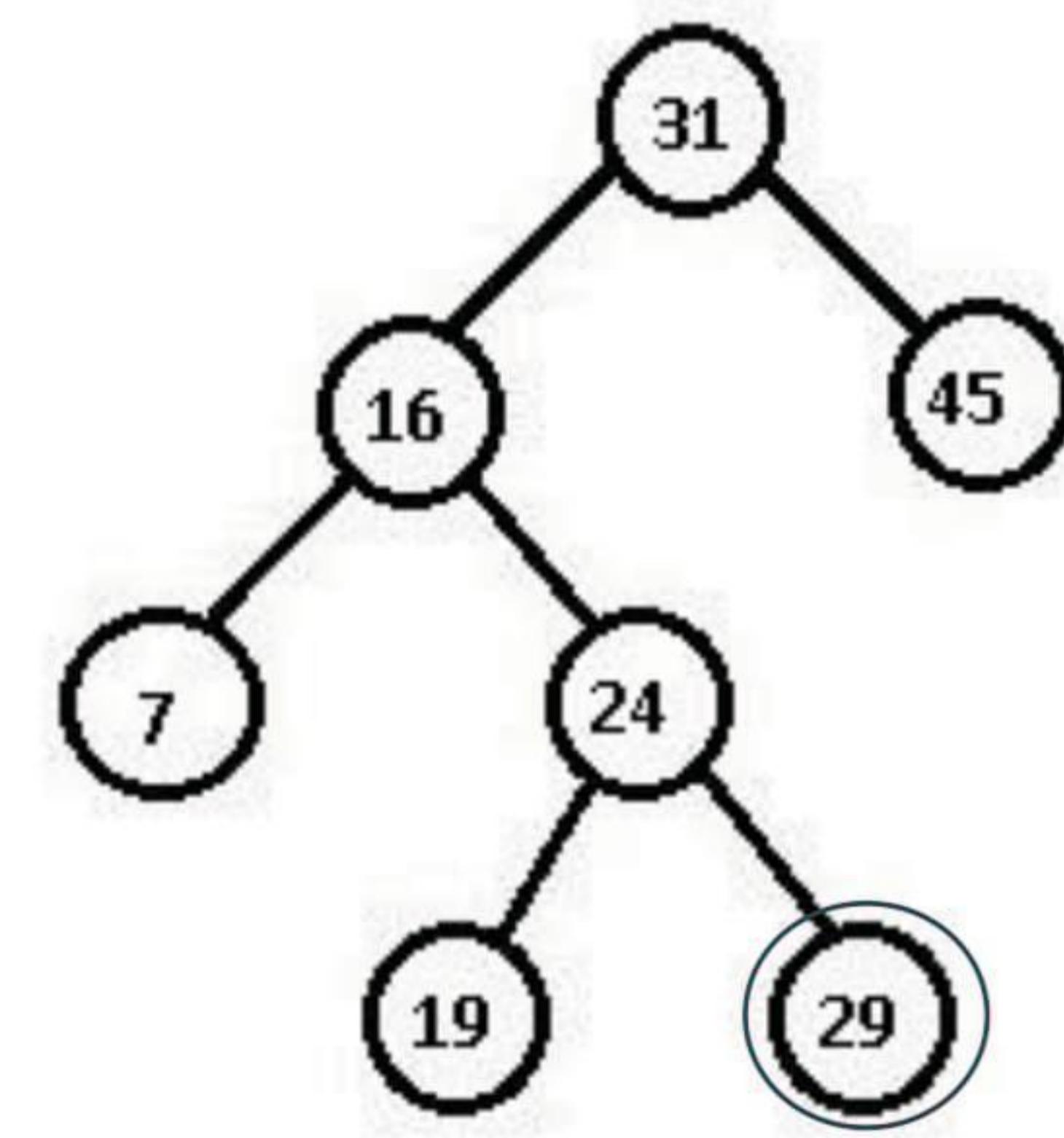
Insert 24



Insert 7



Insert 19



Insert 29

חיפוש איבר בעץ (פתרון איטרטיבי)

```
node *search(node *root, int key)
{
    node *temp = root;
    while(temp != NULL)
    {
        if(temp->data==key)
        {
            printf("\n The %d Element is Present ",temp->data);
            return temp;
        }
        if(key > temp->data)
            temp=temp->right;
        else
            temp=temp->left;
    }
    return NULL;
}
```

חיפוש איבר בעץ (פתרון רקוריובי)

```
node *search(node *root, int key)
{
    // Implement this!!!!
}
```

Inorder של העץ - סריקה

✓ בצורה רקורסיבית סורקים את העץ ומדפיסים את האיברים

❖ קודם תת-עץ שמאל

❖ אחר כר שורש

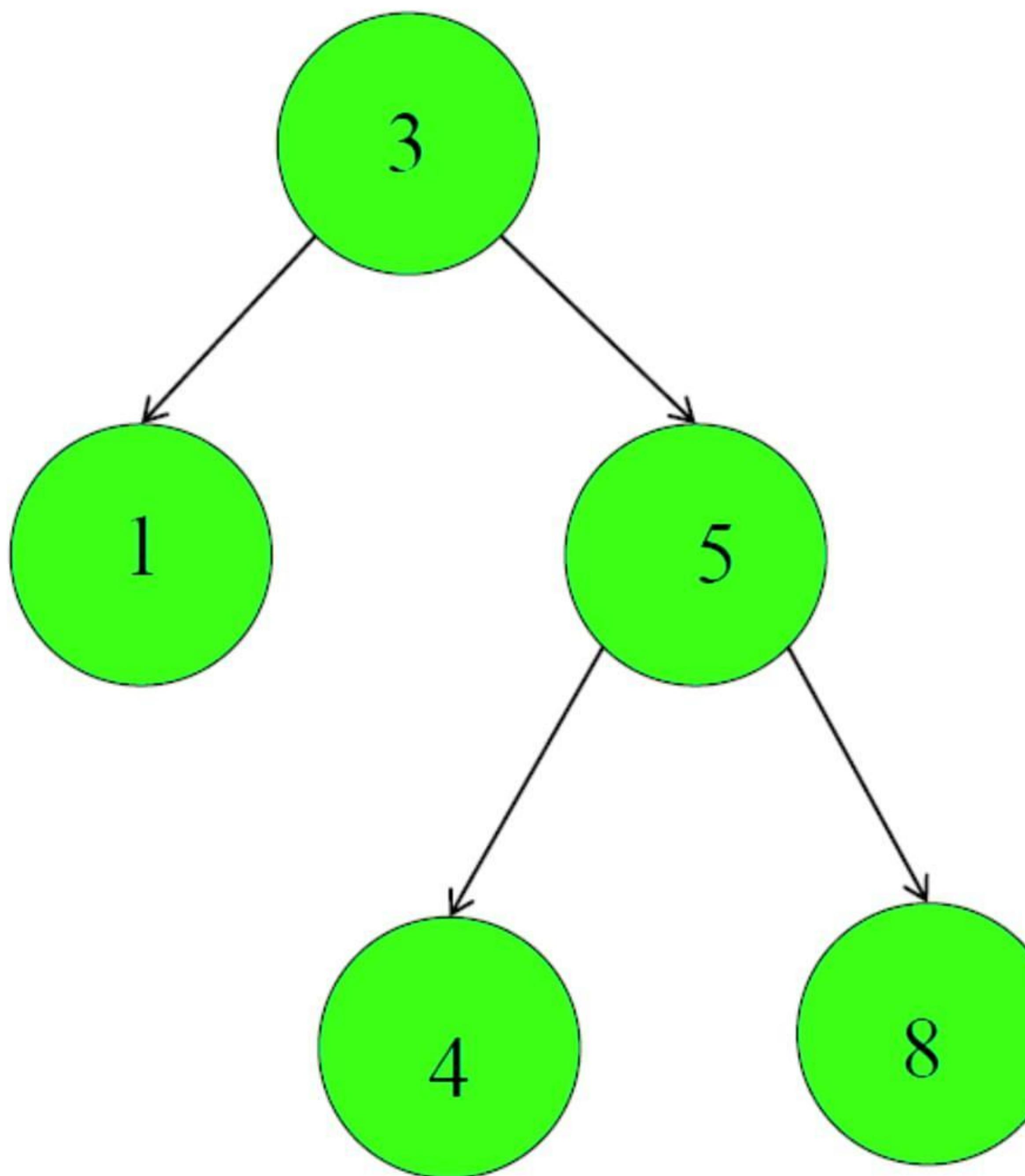
❖ ולאחר מכן תת-עץ ימני.

```
void inorder(node *root)
{
    if(root != NULL)
    {
        inorder(root->left);
        printf("%d", root->data);
        inorder(root->right);
    }
}
```

תרגיל

- ❖ קודם לת-עץ שמאל
- ❖ אחר כר שורש
- ❖ ולאחר מכן לת-עץ ימני.

✓ הפעילו את אלגוריתם הסריקה לפי הסדר inorder על העץ הבא:

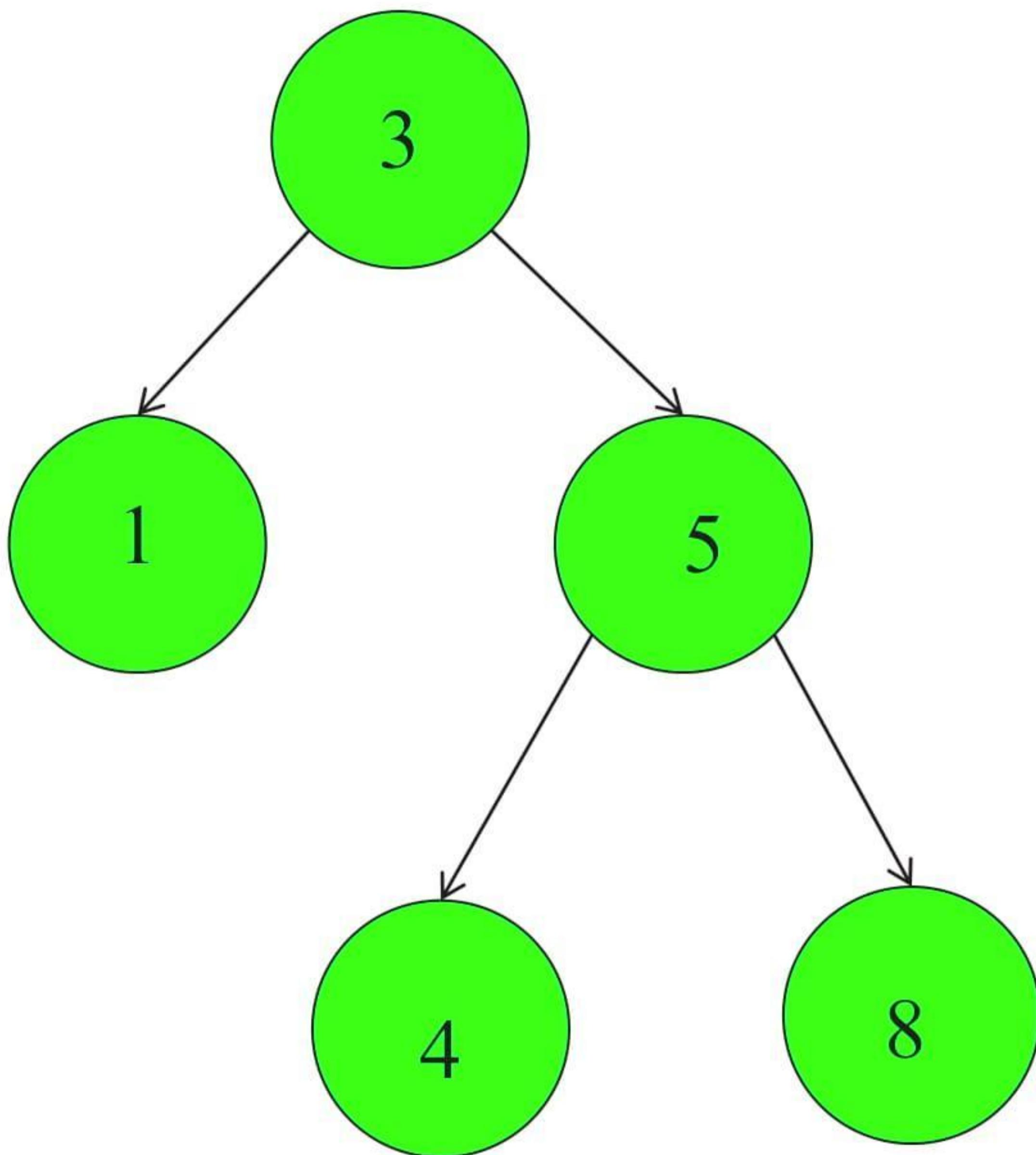


```
void inorder(node *root)
{
    if(root != NULL)
    {
        inorder(root->left);
        printf("%d", root->data);
        inorder(root->right);
    }
}
```

תרגיל

- ❖ קודם לת-עץ שמאל
- ❖ אחר כר שורש
- ❖ ולאחר מכן לת-עץ ימני.

✓ הפעילו את אלגוריתם הסריקה
לפי הסדר `inorder` על העץ הבא:



```
void inorder(node *root)
{
    if(root != NULL)
    {
        inorder(root->left);
        printf("%d", root->data);
        inorder(root->right);
    }
}
```

1,3,4,5,8

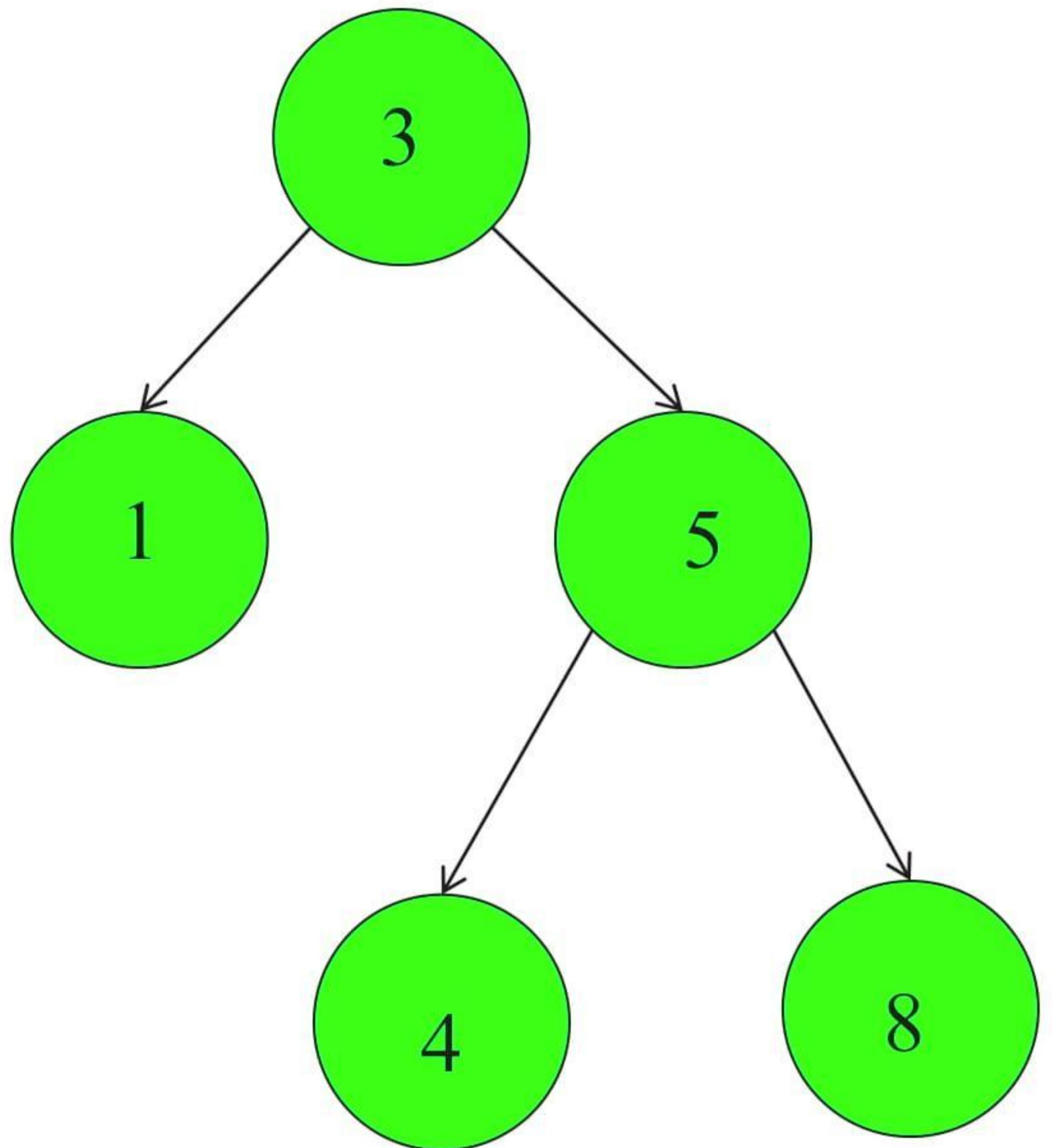
שאלה

- ✓ כתבו פונקציה רקורסיבית הסורקת את העץ בצורת preorder
- ✓ קודם שורש ואז תתי-עצים (תת-עץ שמאל ולאחר מכן תת-עץ ימני)

```
void preorder(node *root)
{
    if(root != NULL)
    {
        printf("%d", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
```

תרגיל

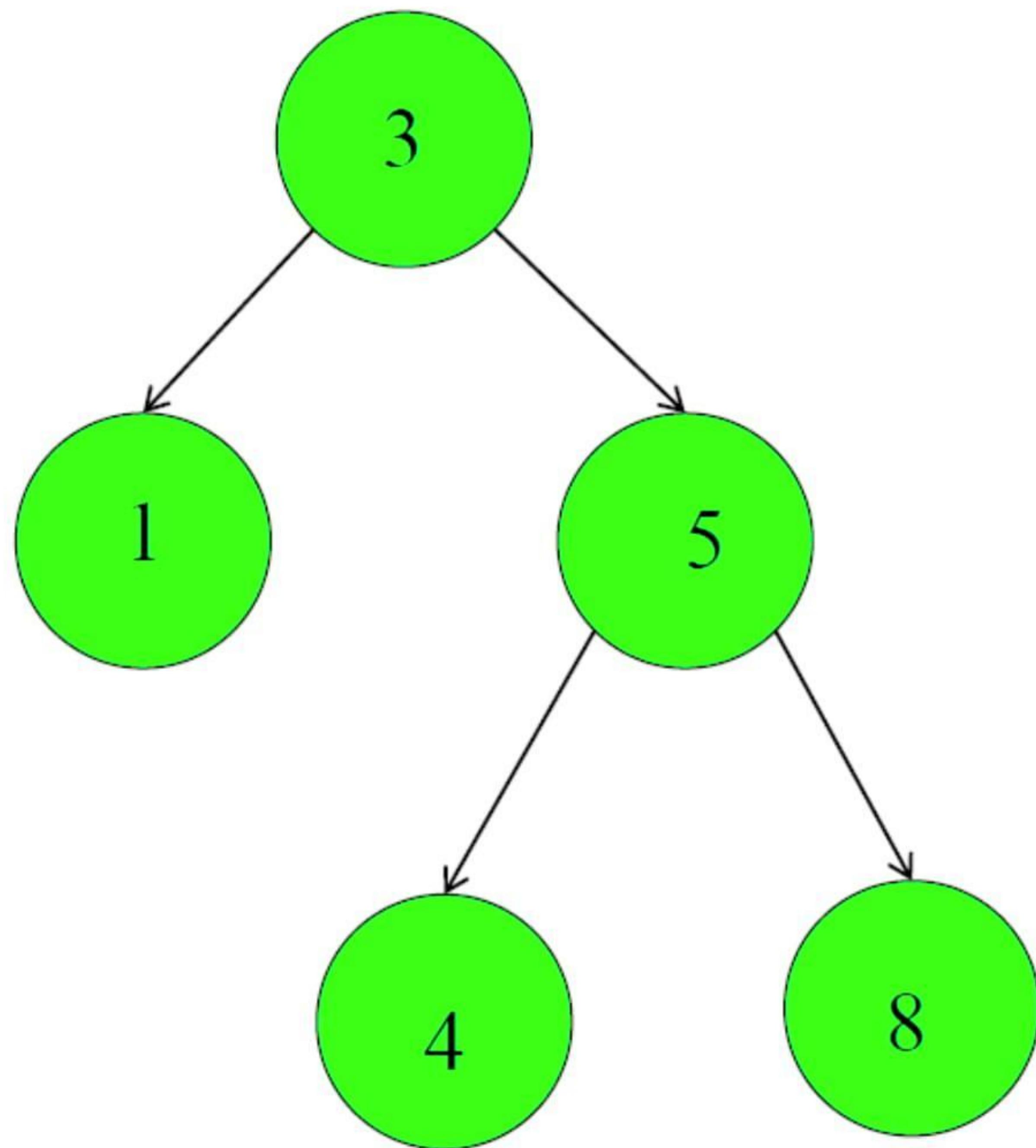
✓ הפעילו את אלגוריתם הסריקה
לפי הסדר preorder על העץ
הבא:



```
void preorder(node *root)
{
    if(root != NULL)
    {
        printf("%d", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
```

תרגיל

✓ הפעילו את אלגוריתם הסריקה
לפי הסדר preorder על העץ
הבא:



```
void preorder(node *root)
{
    if(root != NULL)
    {
        printf("%d", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
```

3,1,5,4,8

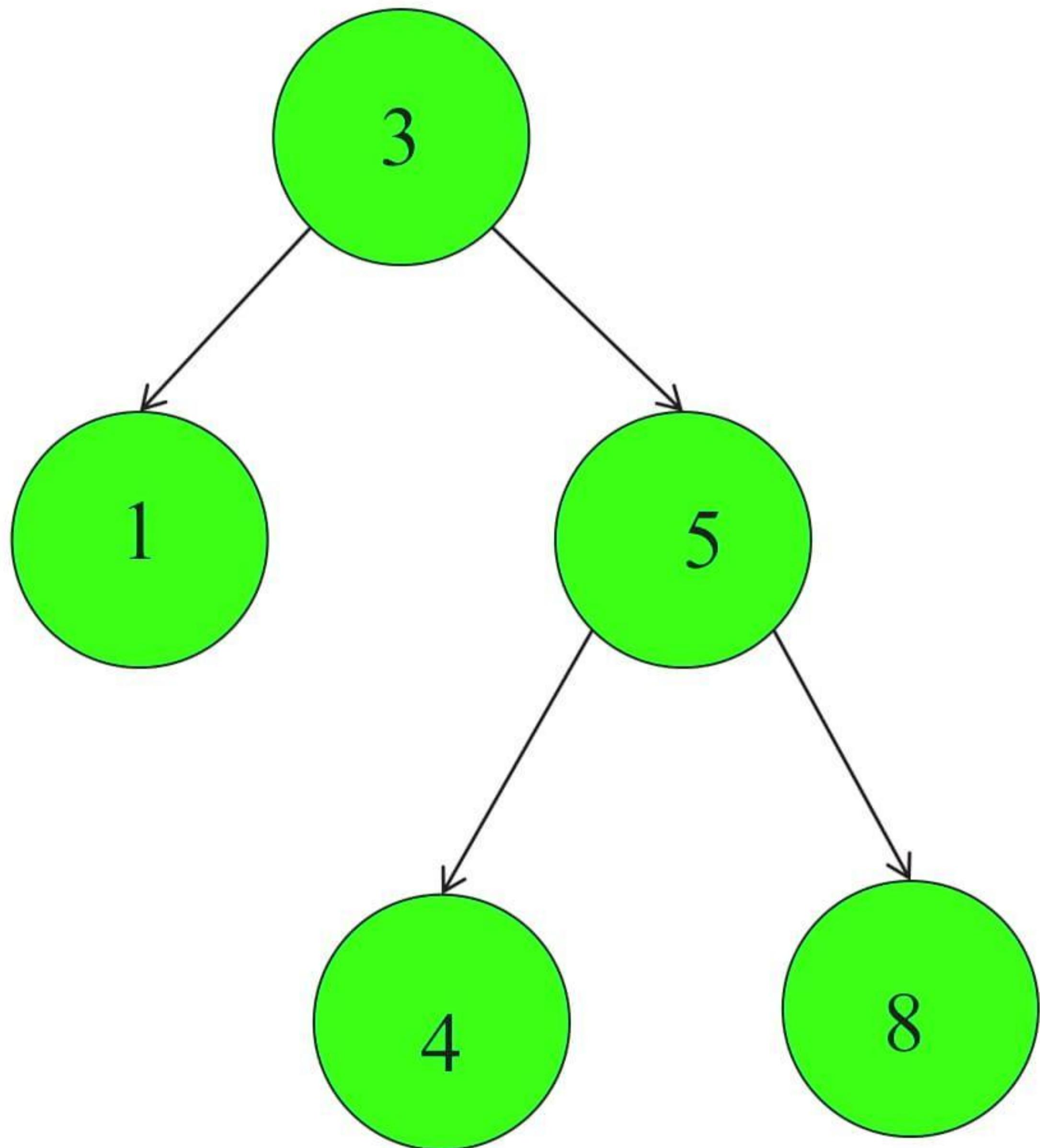
שאלה

- ✓ כתבו פונקציה רקורסיבית הソורקמת את העץ בצורת postorder
- ✓ קודם תת-עצים (תת-עץ שמאלית ולאחר מכן תת-עץ ימנית) ולאחר מכן שורש

```
void postorder(node *root)
{
    if(root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d", root->data);
    }
}
```

תרגיל

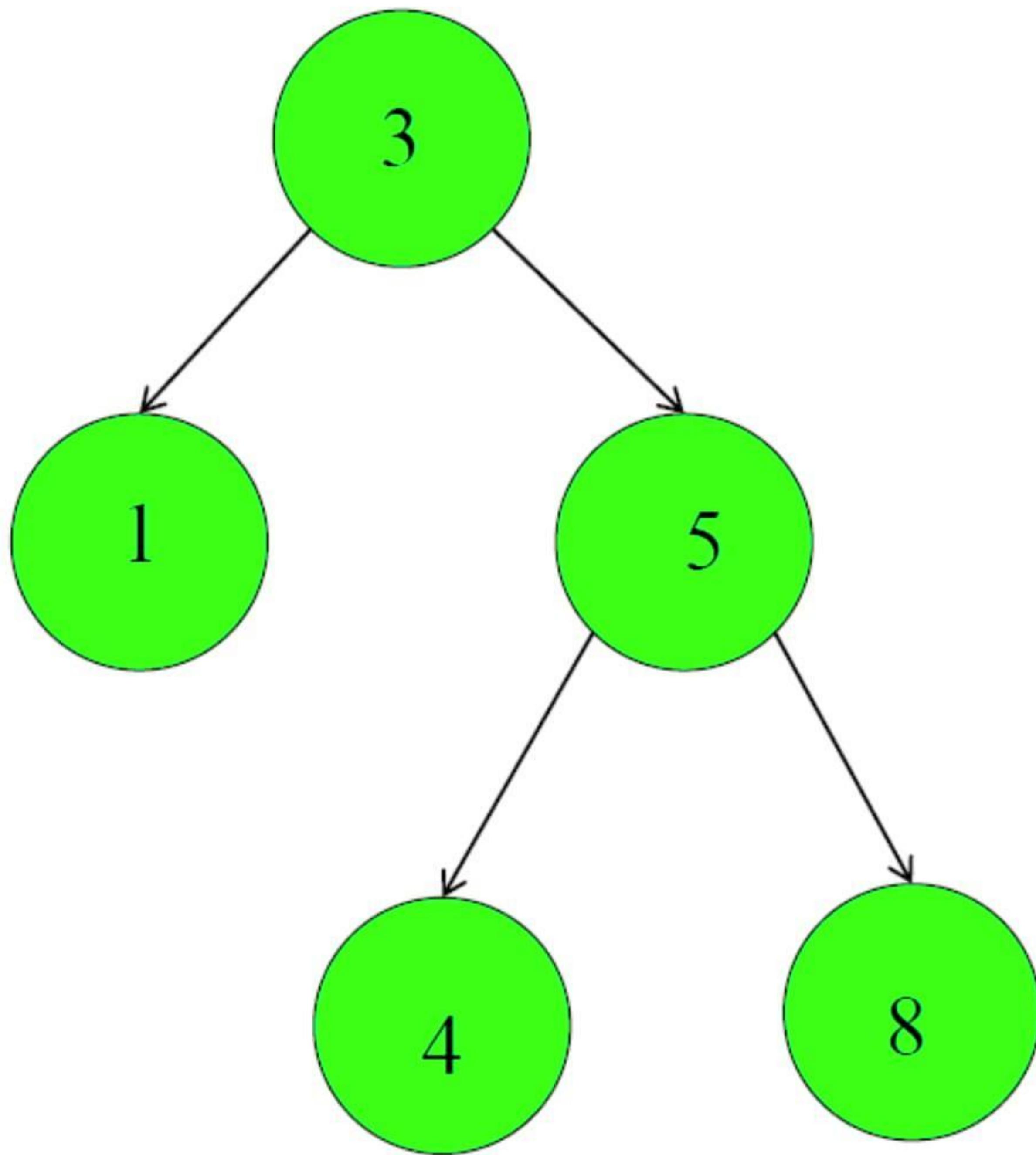
✓ הפעילו את אלגוריתם הסריקה לפי הסדר postorder על העץ הבא:



```
void postorder(node *root)
{
    if(root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d", root->data);
    }
}
```

תרגיל

✓ הפעילו את אלגוריתם הסריקה
לפי הסדר postorder על העץ
הבא:



```
void postorder(node *root)
{
    if(root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d", root->data);
    }
}
```

פונקציה עיקרית

```
void main()
{
    int choice;
    char ans = 'N';
    int key;
    node *newNode,*root,*tmp;
    root = NULL;
    do
    {
        printf("\n1.Create");
        printf("\n2.Search");
        printf("\n3.Recursive Traversals");
        printf("\n4.Exit");
        printf("\nEnter your choice :");
        scanf("%d", &choice);
```

```
1.Create
2.Search
3.Recursive Traversals
4.Exit
Enter your choice :
```

פונקציה עיקרית - המשך

```
switch(choice)
{
    case 1://Create
        do {
            newNode = createNode();
            printf("\nEnter The Element ");
            scanf("%d",&newNode->data);
            if(root == NULL)
                root = newNode;
            else
                insert(root, newNode);
            printf("\nWant To enter More Elements?(y/n)");
            scanf("%c",&ans);
        } while(ans == 'y');
        break;
}
```

```
1.Create
2.Search
3.Recursive Traversals
4.Exit
Enter your choice :1

Enter The Element 4

Want To enter More Elements?(y/n)y

Enter The Element 1

Want To enter More Elements?(y/n)y

Enter The Element 8

Want To enter More Elements?(y/n)y

Enter The Element 9

Want To enter More Elements?(y/n)n
```

פונקציה עיקרית – המשר

case 2: //Search

```
printf("\nEnter Element to be searched :");
scanf("%d",&key);
tmp = search(root, key);
if(tmp == NULL)
    printf("\nThe node %d was not found", key);
break;
```

```
1.Create
2.Search
3.Recursive Traversals
4.Exit
Enter your choice :3

The Inorder display : 1489
```

case 3: //Traversal

```
if(root == NULL)
    printf("\nTree Is Not Created");
else
{
    printf("\n The Inorder display : ");
    inorder(root);
}
break;

default:
    printf("Invalid Choice %d, choice);
} //end of switch block
}while(choice != 4);
} //end of main
```

```
1.Create
2.Search
3.Recursive Traversals
4.Exit
Enter your choice :2

Enter Element to be searched :4

The 4 Element is Present
1.Create
```

שאלה

- ✓ בהינתן שורש העץ ROOT ואיבר T הנמצא בעץ יש לאתר את הבא של T בעץ.
- ✓ אין להניח הנחות:
 - ❖ יתכן שהעץ ריק
 - ❖ יתכן שהאיבר לא קיים בעץ!

פתרון איטרטיבי

```
node *getFather(node *root, node *t)
{
    node *temp = root;
    if(temp == t)
        return NULL; //Father not exist

    while(temp != NULL)
    {
        if(temp->left == t || temp->right == t) //one of temp's children
            return temp;
        else
            if(t->data <= temp->data)
                temp = temp->left;
            else
                temp = temp->right;
    }
    return NULL //Not Found! Or Tree is empty!
}
```

פתרון קורסיבי

```
node * getFather(node *root, node *t)
{
    if(root == NULL || root == t )
        return NULL; //Not Found! Or Tree is empty!

    if (root->left == t || root->right == t) //one of the current node's children
        return root;

    if (t->data <= root->data)
        return father(root->left);

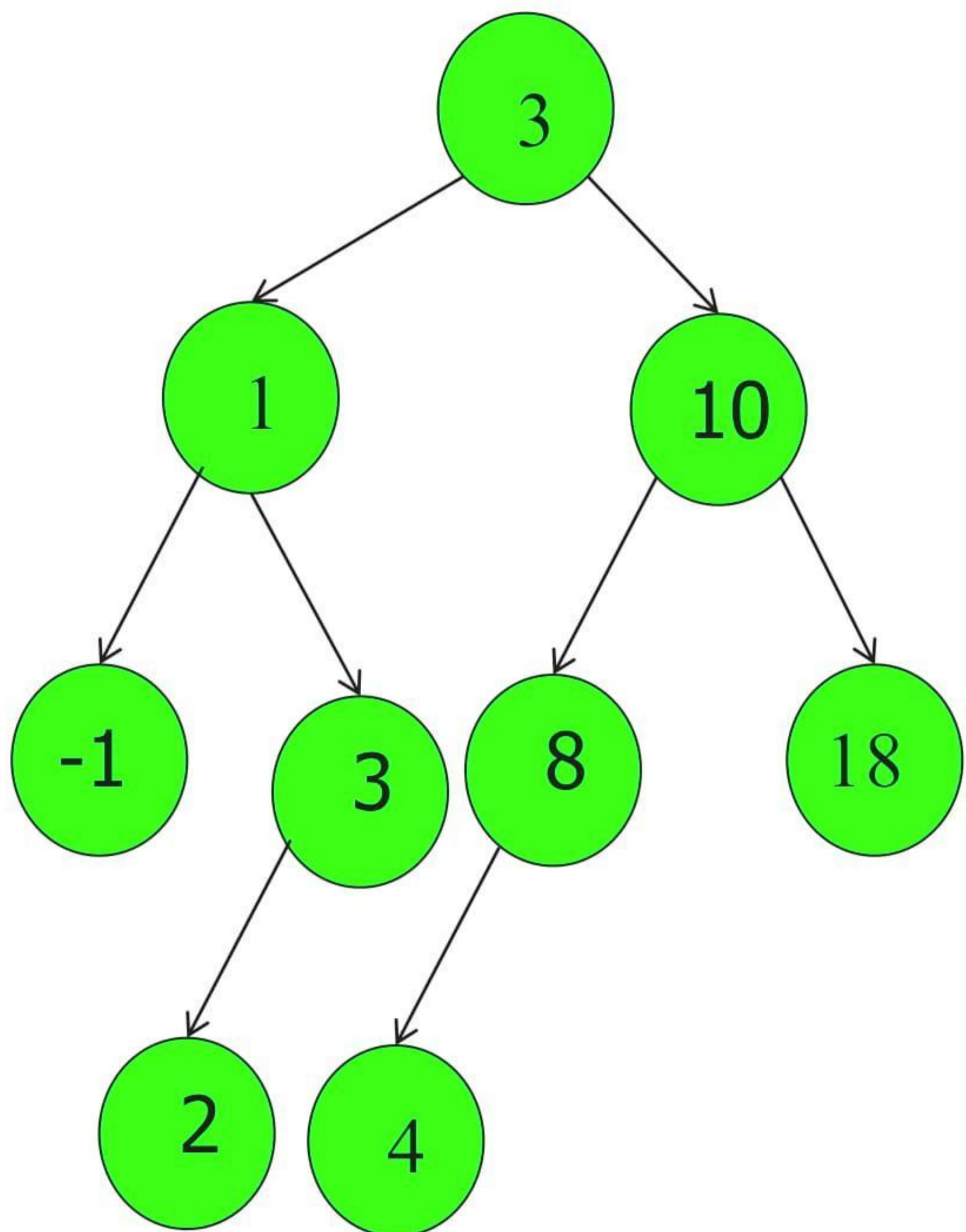
    return father(root->right);
}
```

מימוש, עץ חיפוש בינארי C ADT

```
typedef struct
{
    node *root;
} BST;

//Initializes root to NULL
void init(BST *bst)
void insertBST(BST *bst, int d);
node *searchBST(BST *bst, int key)
void traverseInorder(BST *bst);
void traversePreorder(BST *bst);
void traversePostorder(BST *bst);
int isEmpty(BST *bst); //returns 1 if tree is
//Empty, Otherwise 0.
```

מחיקת איבר



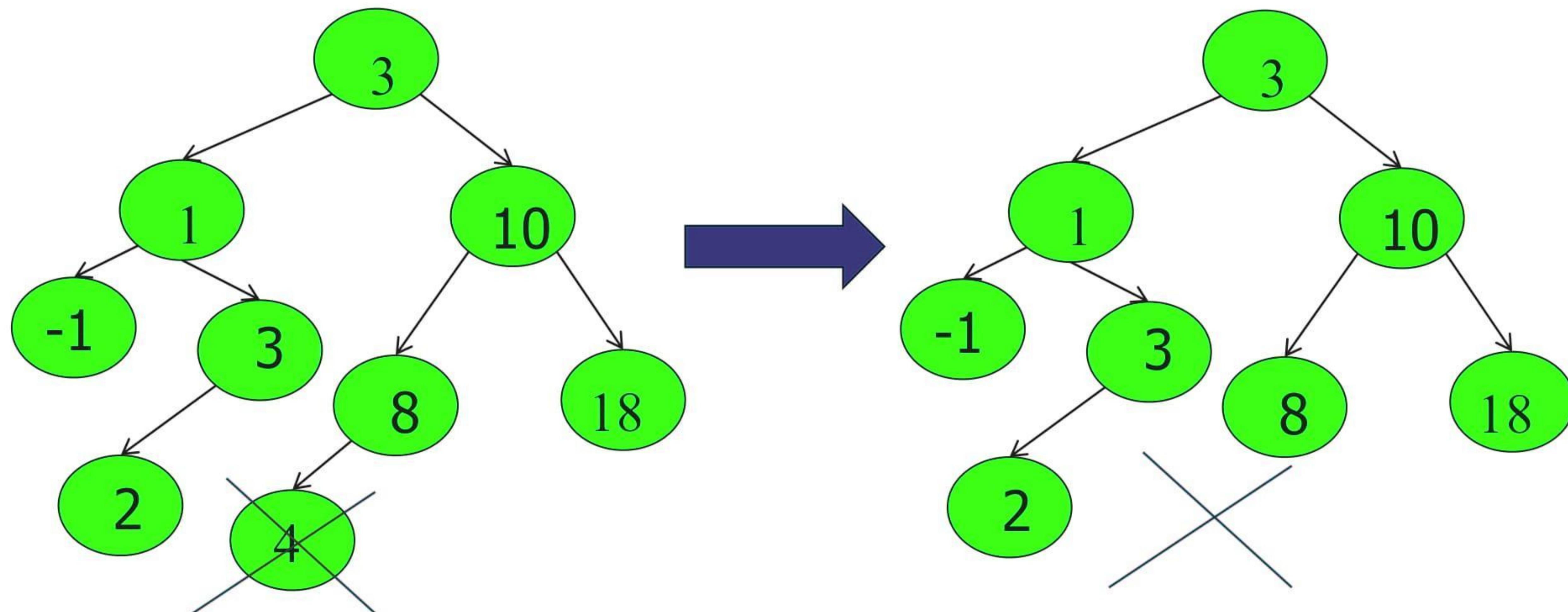
✓ תרגיל

- ✓ חישבו איך מוחקקים איבר מהעץ.
- ✓ איך מוחקקים עלה (leaf)?
- ✓ איך מוחקקים צומת פנימי כלשהו (node)?
- ✓ איך מוחקקים שורש (root)?

נשים לב כי הערך 3 מופיע פעמיים כי בעלה שמאלו מוגדר קטן שווה

מחיקת איבר

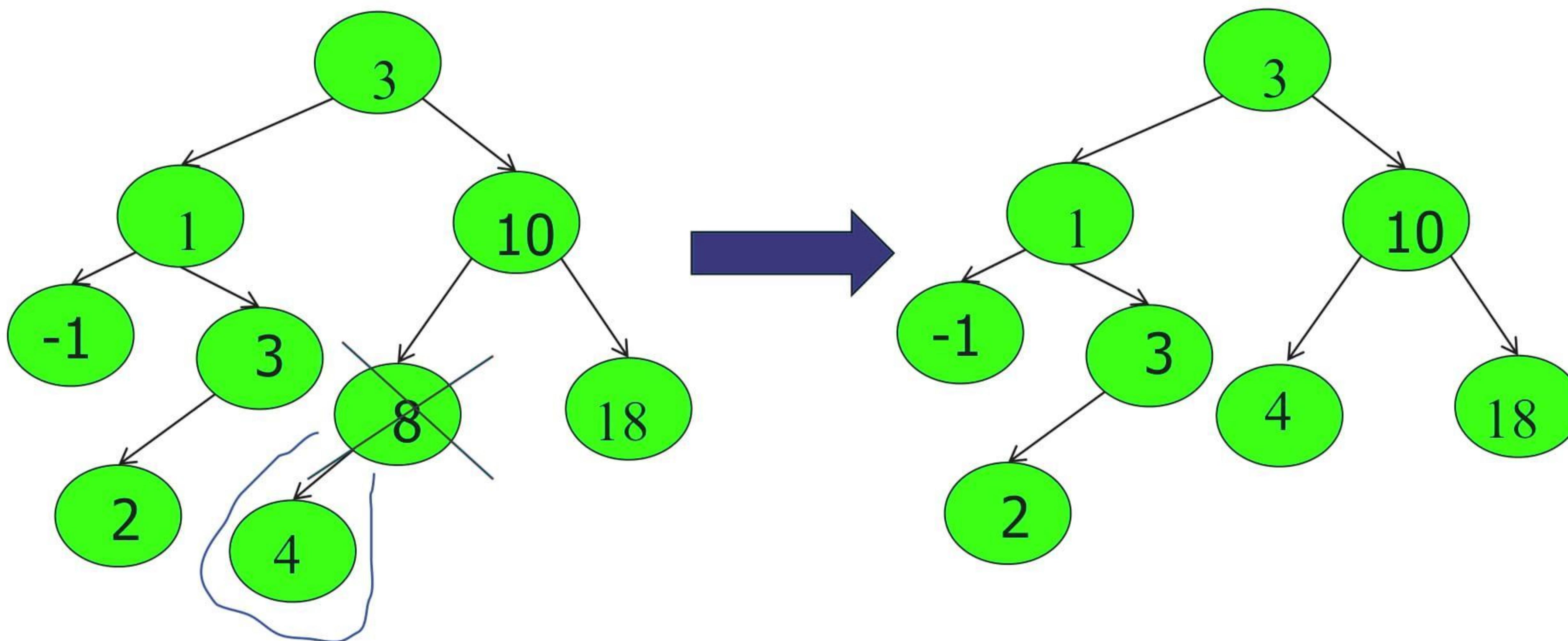
- ✓ מקרה 1: האיבר הוא עלה :
- ✓ דוגמה מחיקה של הערך "4":



מחיקת איבר

✓ מקרה 2: האיבר למחיקה בעל בן יחיד:

✓ דוגמה-מחיקה של הערך "8":

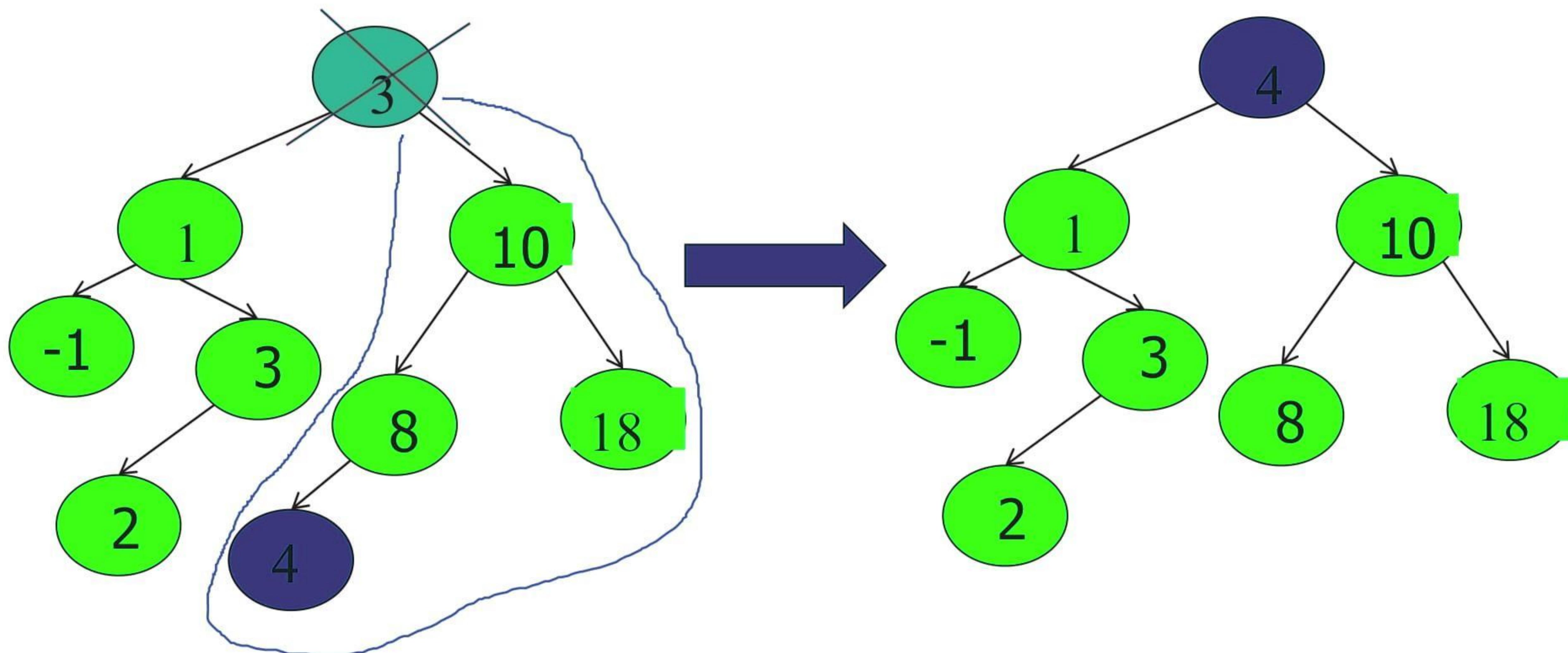


מחיקת איבר

✓ מקרה 3: האיבר למחיקה בעל שני בניים:

✓ דוגמה-מחיקה של הערך "3":

✓ טכנית – מציאת האיבר ה"עוקב" (הכי קטן בעץ ימני)



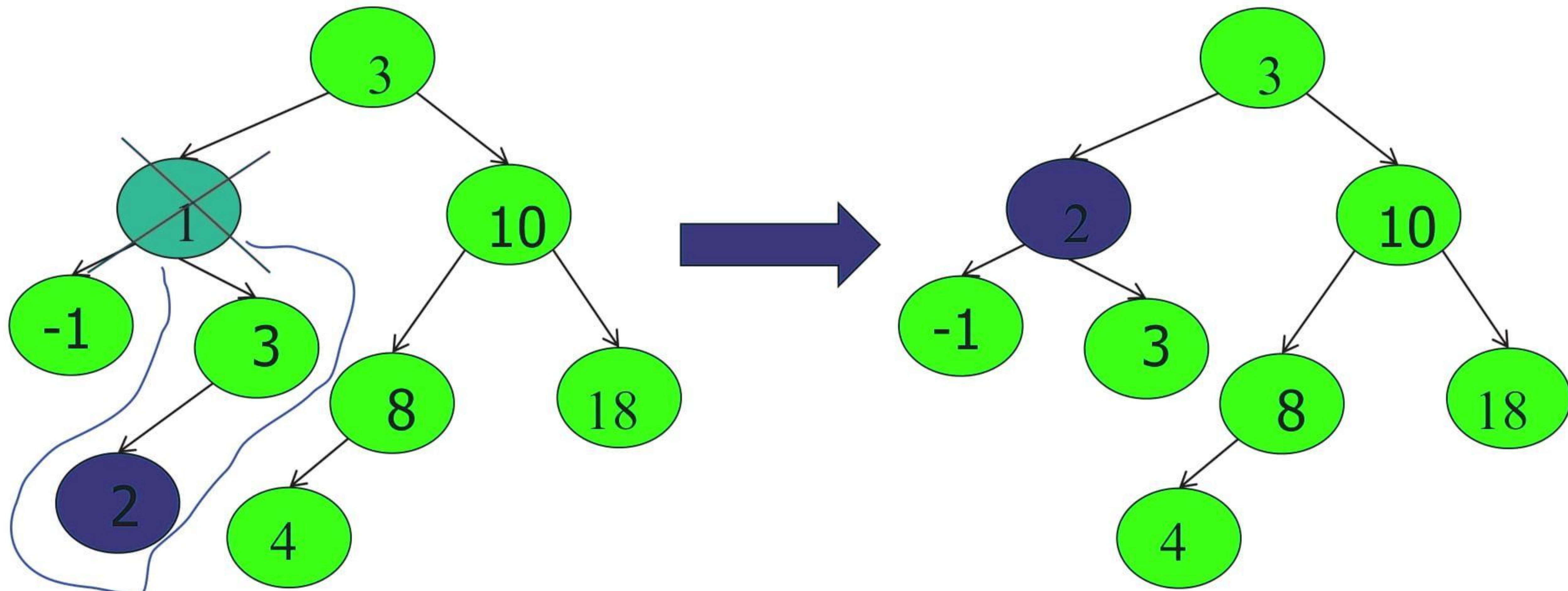
מחיקת איבר

✓ מקרה 3: האיבר למחיקה בעל שני בניים:

החל מהענץ של 1

✓ דוגמה-מחיקה של הענץ "1":

✓ טכנית – מציאת האיבר ה"עוקב" (הכי קטן בעץ ימני)

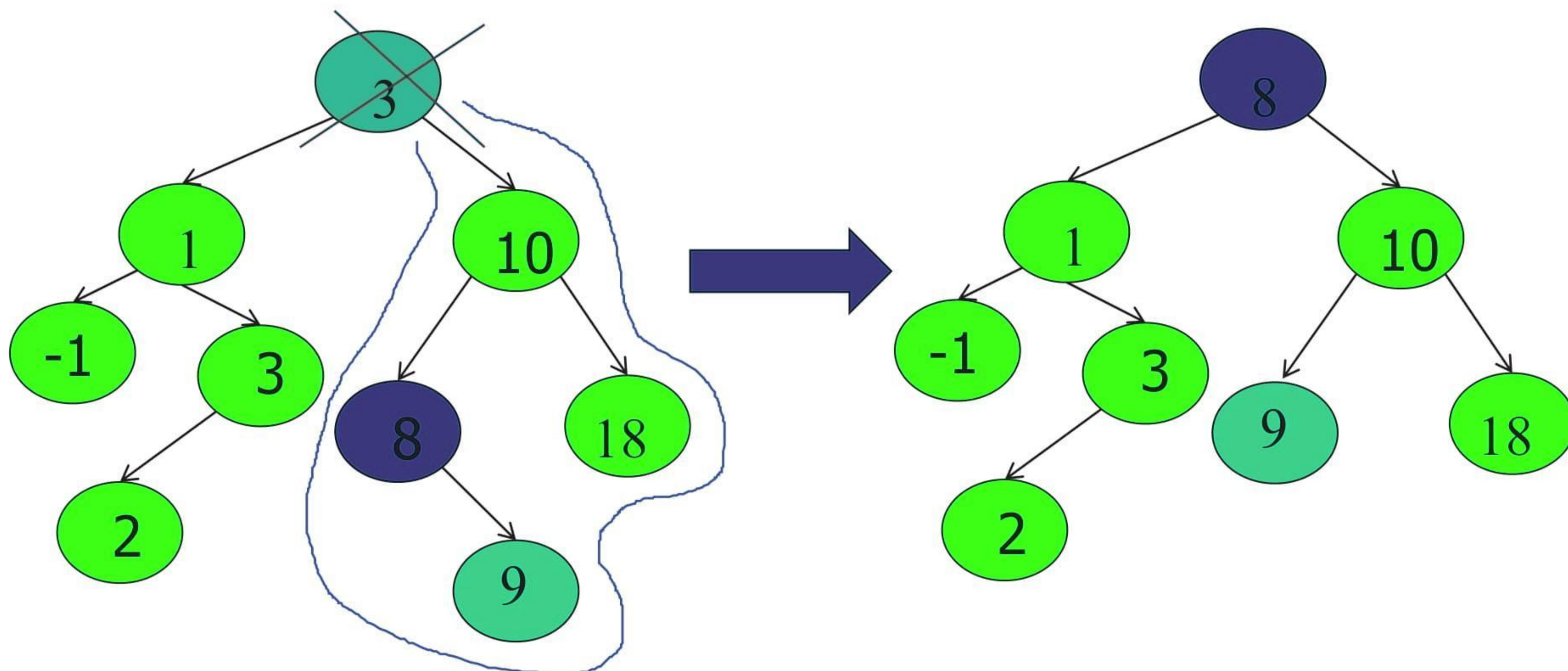


מחיקת איבר

✓ מקרה 3: האיבר למחיקה בעל שני בניים:

✓ דוגמה-מחיקה של הערך "3":

✓ טכנית – מציאת האיבר ה"עוקב" (הכי קטן בעץ הימני)



פתרונות | פונקציה למציאת ערך מינימלי בהינתן תחילת עץ או תחילת תחת עץ

```
/*
 Given a non-empty binary search tree, return the node
 with minimum key value found in that tree.
 */
struct node* minValueNode(struct node* node)
{
    struct node* current = node;

    /* loop down to find the leftmost leaf */
    while (current && current->left != NULL)
        current = current->left;

    return current;
}
```

```

struct node* deleteNode(struct node* root, int key) {
    if (root == NULL) return root; //empty tree case

    if (key < root->key) //deleted item in the left tree
        root->left = deleteNode(root->left, key);
    if (key > root->key) //deleted item in the right tree
        root->right = deleteNode(root->right, key);

    if(key == root->key) { //deleted item found
        if (root->left == NULL && root->right == NULL) {
            free(root); return NULL;
        }
        if (root->left == NULL || root->right == NULL) {
            struct node *temp = (root->left)? root->left : root->right;
            free(root);
            return temp;
        }
        else{
            struct node* temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
    }
    return root;
}

```