**اشکان شکیبا (9931030)**

**سوال اول**

شبه کد:

**void** postOrderIterative(Node root) {

**declare** Stack<Node> stack

**while** (**true**) {

**while** (root ≠ **null**) {

stack.push(root)

stack.push(root)

root ← root.leftChild

}

**if** (stack.empty()) {

**return**

}

root ← stack.pop()

**if** (!stack.empty() **and** stack.peek() = root) {

root ← root.rightChild

}

**else** {

print(root.value)

root ← **null**

}

}

}

روش دوم، با استفاده از دو Stack:

**void** postOrderIterative(Node root) {

**declare** Node node

**declare** Stack<Node> stack1, stack2

**while** (~stack1.empty()) {

node ← stack1.pop()

stack2.push(node)

stack1.push(node.leftChild)

stack1.push(node.rightChild)

}

**while** (~stack2.empty()) {

node ← stack2.pop()

print(node.value)

}

}

تحلیل زمانی:

با توجه به اینکه حلقه ها به تعداد اعضای Stack ها تکرار میشوند، میتوان نتیجه گرفت پیچیدگی زمانی این الگوریتم از مرتبه n (تعداد اعضا) است.

**سوال دوم**

شبه کد:

**declare** Node prev

**boolean** isBST() {

prev ← **null**

**return** isBST(root)

}

**boolean** isBST(Node node) {

**if** (node ≠ **null**) {

**if** (~isBST(node.leftChild)) {

**return** **false**

}

**if** (prev ≠ **null** **and** node.value <= prev.value) {

**return** **false**

}

prev ← node

**return** isBST(node.rightChild)

}

**return** **true**

}

برای بررسی درخت، کافی ست متد isBST() فراخوانی شود.

**سوال سوم**

الف) پیاده سازی در جاوا:

class Node {  
 int value;  
 Node left, right, nextRight;  
  
 Node(int value) {  
 this.value = value;  
 left = right = nextRight = null;  
 }  
}  
  
class BinaryTree {  
 Node root1,root2;  
  
 boolean areIdentical(Node root1, Node root2) {  
 if (root1 == null && root2 == null)  
 return true;  
  
 if (root1 == null || root2 == null)  
 return false;  
  
 return (root1.value == root2.value && areIdentical(root1.left, root2.left) && areIdentical(root1.right, root2.right));  
 }  
  
 boolean isSubtree(Node T, Node S) {  
 if (S == null)  
 return true;  
  
 if (T == null)  
 return false;  
  
 if (areIdentical(T, S))  
 return true;  
  
 return isSubtree(T.left, S) || isSubtree(T.right, S);  
 }  
}

ب) پیاده سازی در جاوا:

class Node {  
  
 char value;  
 Node left, right;  
  
 Node(char value) {  
 this.value = value;  
 left = right = null;  
 }  
}  
  
class IMN {  
 int i;  
 int m = 0;  
 int n = 0;  
}  
  
class BinaryTree {  
 static Node *root*;  
 IMN a = new IMN();  
  
 String strstr(String haystack, String needle) {  
 if (haystack == null || needle == null) {  
 return null;  
 }  
 int hLength = haystack.length();  
 int nLength = needle.length();  
 if (hLength < nLength) {  
 return null;  
 }  
 if (nLength == 0) {  
 return haystack;  
 }  
 for (int i = 0; i <= hLength - nLength; i++) {  
 if (haystack.charAt(i) == needle.charAt(0)) {  
 int j = 0;  
 for (; j < nLength; j++) {  
 if (haystack.charAt(i + j) != needle.charAt(j)) {  
 break;  
 }  
 }  
 if (j == nLength) {  
 return haystack.substring(i);  
 }  
 }  
 }  
 return null;  
 }  
  
 void storeInorder(Node node, char[] arr, IMN i) {  
 if (node == null) {  
 arr[i.i++] = '$';  
 return;  
 }  
 storeInorder(node.left, arr, i);  
 arr[i.i++] = node.value;  
 storeInorder(node.right, arr, i);  
 }  
   
 void storePreOrder(Node node, char[] arr, IMN i) {  
 if (node == null) {  
 arr[i.i++] = '$';  
 return;  
 }  
 arr[i.i++] = node.value;  
 storePreOrder(node.left, arr, i);  
 storePreOrder(node.right, arr, i);  
 }  
   
 boolean isSubtree(Node T, Node S) {  
 if (S == null) {  
 return true;  
 }  
 if (T == null) {  
 return false;  
 }  
  
 char[] inT = new char[100];  
 String op1 = String.*valueOf*(inT);  
 char[] inS = new char[100];  
 String op2 = String.*valueOf*(inS);  
 storeInorder(T, inT, a);  
 storeInorder(S, inS, a);  
 inT[a.m] = '\0';  
 inS[a.m] = '\0';  
  
 if (strstr(op1, op2) != null) {  
 return false;  
 }  
  
 a.m = 0;  
 a.n = 0;  
 char[] preT = new char[100];  
 char[] preS = new char[100];  
 String op3 = String.*valueOf*(preT);  
 String op4 = String.*valueOf*(preS);  
 storePreOrder(T, preT, a);  
 storePreOrder(S, preS, a);  
 preT[a.m] = '\0';  
 preS[a.n] = '\0';  
   
 return (strstr(op3, op4) != null);  
 }  
}

**سوال چهارم**

الف) متد HeapDelete

ب) متد HeapInsert

ج) متد HeapChangeKey

**void** HeapInsert (Heap A, Element value) {

A.length ← A.length + 1

A[A.length] ← -∞

HeapChangeKey(A, A.length, value)

}

**void** HeapChangeKey (Heap A, NodeIndex i, Element value) {

A[i] ← value

**while** (i > 1 and A[i] > A[floor(i/2)]) {

swap(A[i], A[floor(i/2)])

i ← floor(i/2)

}

}

**void** HeapDelete (Heap A, NodeIndex i) {

A[i] ← A[A.length]

A.length ← A.length - 1

**while** (A[floor(i/2)] < A[i] **and** i > 1) {

swap(A[i], A[floor(i/2)])

i ← floor(i/2)

}

heapify(i)

}