CENG2001- 2020 – MIDTERM - Report

12-Aralık-2020

Name : Hasan Ali Özkan
ID : 180709020

Challenge: Linked List Merge Sort

Time Complexity: O(nlogn)

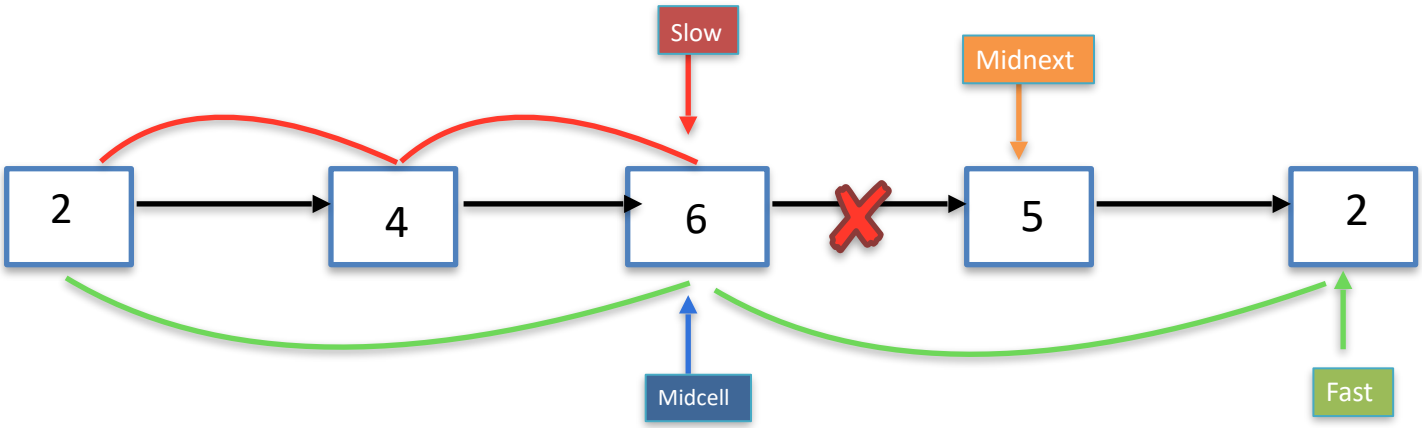
Example Input: 2 4 6 5 2

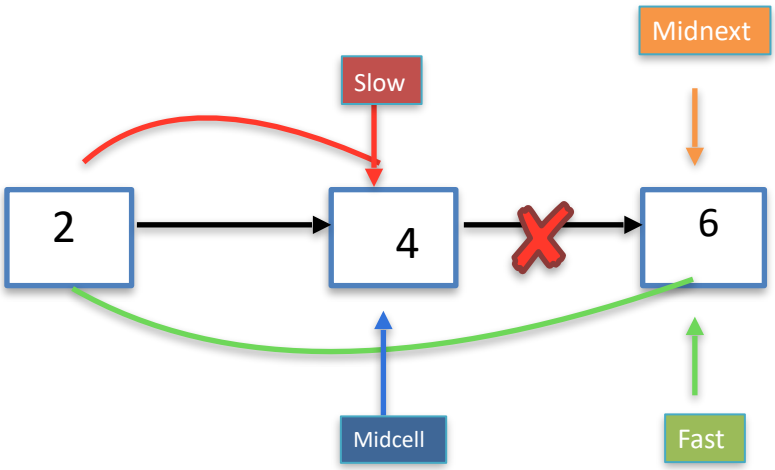
Solution Code Trace:

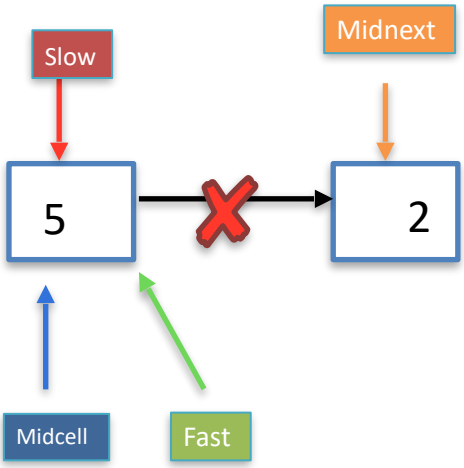
```
69
70      private static void sort(Solution.LinkedList<Integer> llist) {
71          llist.head = mergeSort(llist.head);
72      }
73
74      public static Solution.ListCell<Integer> mergeSort(Solution.ListCell<Integer> head) {
75          if(head == null || head.next == null)
76              return head;
77
78          Solution.ListCell<Integer> slower =head;
79          Solution.ListCell<Integer> faster = head;
80          while (faster != null && faster.next != null){
81              faster = faster.next;
82              if(faster.next != null){
83                  faster = faster.next;
84                  slower = slower.next;
85              }
86
87
88
89          }
90          Solution.ListCell<Integer> midCell =slower;
91          Solution.ListCell<Integer> midNext = midCell.next;
92          midCell.next = null;
93          Solution.ListCell<Integer> leftPart = mergeSort(head);
94          Solution.ListCell<Integer> rightPart = mergeSort(midNext);
95          Solution.ListCell<Integer> sorted = merge(leftPart,rightPart);
96          return sorted;
97      }
98
99
100     public static Solution.ListCell<Integer> merge(Solution.ListCell<Integer> leftPart, Solution.ListCell<Integer>
    rightPart) {
101         Solution.ListCell<Integer> cell = null;
102         if (leftPart == null){
103             return rightPart;
104         }
105         if (rightPart == null){
106             return leftPart;
107         }
108         if (leftPart.datum <= rightPart.datum){
109              cell = leftPart;
110              cell.next = merge(leftPart.next,rightPart);
111         }
112
113         else {
114             cell = rightPart;
115             cell.next = merge(leftPart,rightPart.next);
116         }
117
118
119         return cell;
120
121
122     }
```

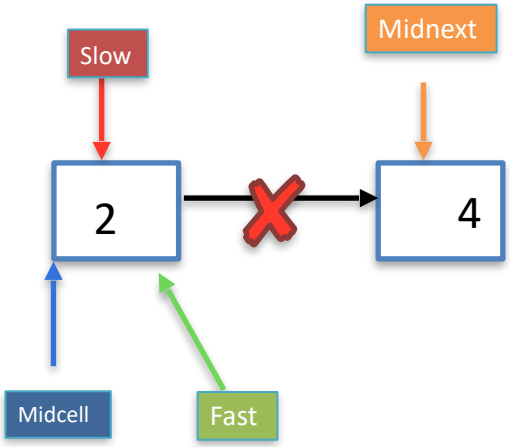
The Linked List look like this:

For divide part my code is working like that:

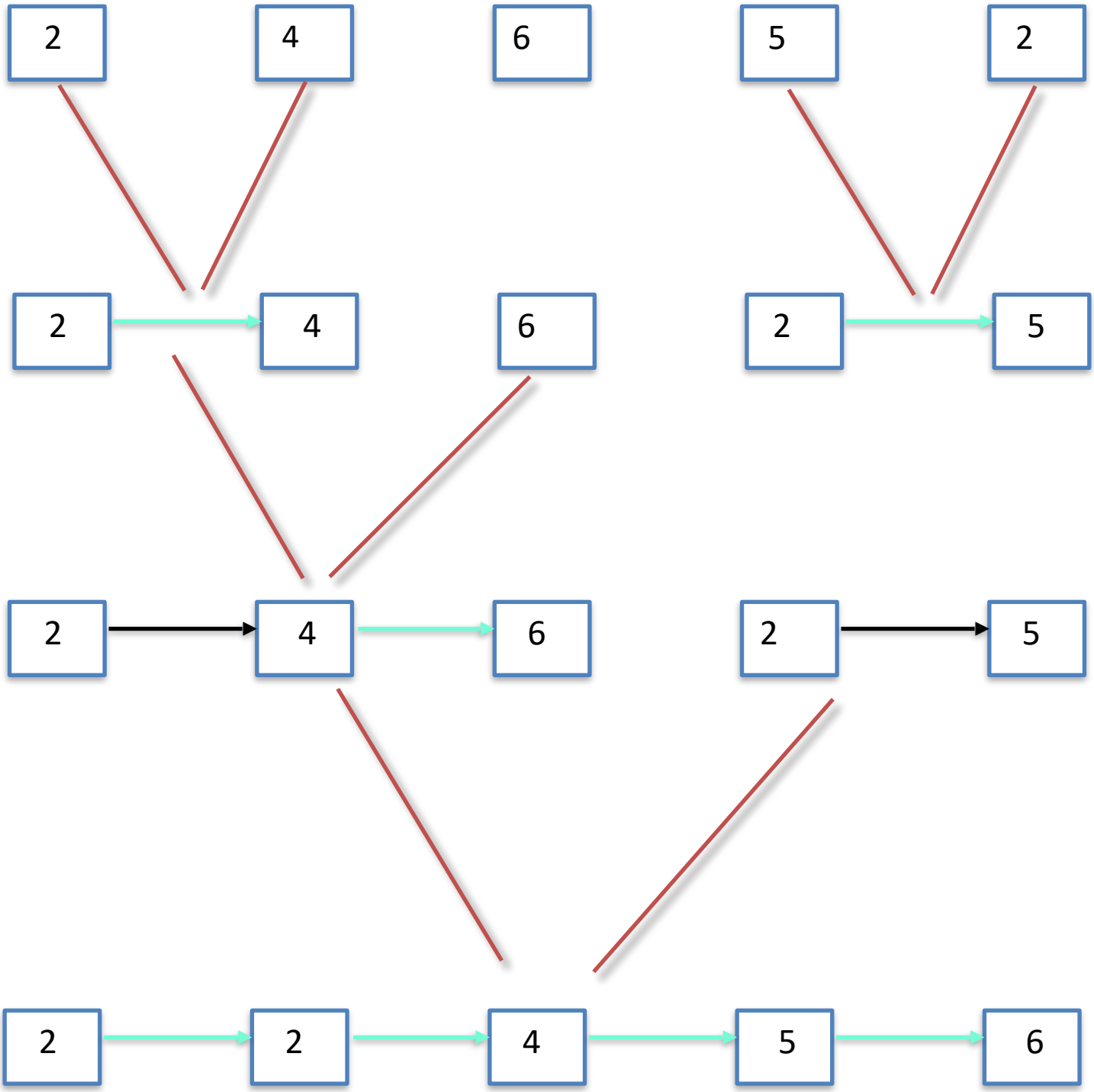








For conquer part my code is working like that:



Actually I used just an ordinary merge-sort in my code.This algorithm based on a divide and conquer approach. In order to find the middle of the linked-list I used rabbit-turtle algorithm. In order to use this method I get some help from the internet.While the turtle moves x speed the rabbit moves the 2x speed just like in my code. I realize much later there is a method which gives to us the size of the linked-list I also can this method but I struggled with this question two days. When I first submit it passed only two test cases because I was making a mistake calling my method.

Before:

```
 private static void sort(Solution.LinkedList<Integer> llist) {
    mergeSort(llist.head);
}
```

After:

```
private static void sort(Solution.LinkedList<Integer> llist) {
    llist.head = mergeSort(llist.head);
}
```

If the head-datum of the linked-list is the smallest then the first calling is work. If the location of the head is changed then the code isn't working. I noticed much later and fixed it this small mistake took me two days.

Challenge: Trees: Construct Binary Tree from Traversals

Time Complexity:O(n)

Example Input:

0 1 11 9 13 3 6 10
11 1 9 13 0 6 3 10

Solution Code Trace:

```
private static ArrayList<Integer> getPostOrder(ArrayList<Integer> preOrder, ArrayList<Integer> inOrder) {
       Map<Integer,Integer> treeMap = new HashMap<>();
       for (int i = 0; i <inOrder.size() ; i++) {
           treeMap.put(inOrder.get(i),i);
       }
       Stack<TreeCell<Integer>> treeStack = new Stack<>();
       TreeCell<Integer> root = new TreeCell<>(preOrder.get(0));
       treeStack.push(root);
       for (int i = 1; i <preOrder.size() ; i++) {
           int  datum = preOrder.get(i);
           TreeCell<Integer> cell = new TreeCell<>(datum);
           if (treeMap.get(datum)<treeMap.get(treeStack.peek().datum)){
               treeStack.peek().left = cell;
           }
           else{
               TreeCell<Integer> parent = null;
               while (!treeStack.isEmpty() && treeMap.get(datum) > treeMap.get(treeStack.peek().datum)) {
                   parent = treeStack.pop();
               }
               parent.right = cell;
           }
           treeStack.push(cell);
       }
       ArrayList<Integer> postOrder = new ArrayList<Integer>();
       Stack<TreeCell<Integer>> stack1 = new Stack<TreeCell<Integer>>();
       Stack<TreeCell<Integer>> stack2 = new Stack<TreeCell<Integer>>();
       stack1.push(root);
       while (!stack1.isEmpty()){
           TreeCell<Integer> treeCell = stack1.pop();
           stack2.push(treeCell);
           if (treeCell.left!=null){
               stack1.push(treeCell.left);
           }
           if (treeCell.right!=null){
               stack1.push(treeCell.right);
           }
       }
       while (!stack2.empty()){
           TreeCell<Integer> tempCell = stack2.pop();
           postOrder.add(tempCell.datum);
       }
       return postOrder;
   }


```

treeMap = {(11,0),(1,1),(9,2),(13,3),(0,4),(6,5),(3,6),(10,7)}

Preorder (Root-Left-Right)

Inorder (Left-Root-Right)

To create the tree:

root=>(0) and push the root into the stack.

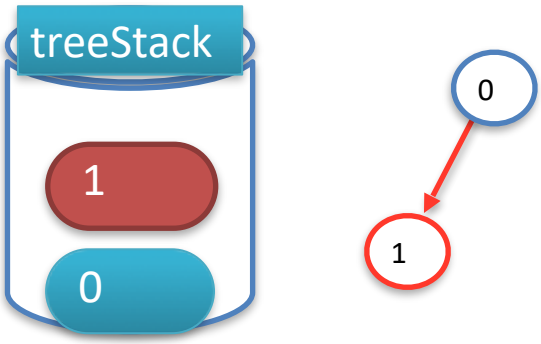
i = 1{

datum = 1

cell => (1)

1<4 = true -> then;

The position of the cell is to the left of the topmost element in the stack.



Push the cell in to the stack.

}

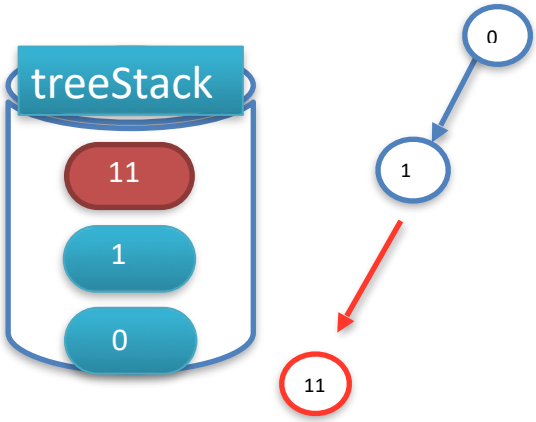
i = 2{

datum = 11

cell => (11)

0<1 = true -> then;

The position of the cell is to the left of the topmost element in the stack.



Push the cell into the stack.

}

İ = 3{

datum = 9

cell => (9)

2<0 = false -> then;

treeStack is not empty and 2>0 = true -> then;

parent => (11)

treeStack is not empty and 2>1 true -> then;

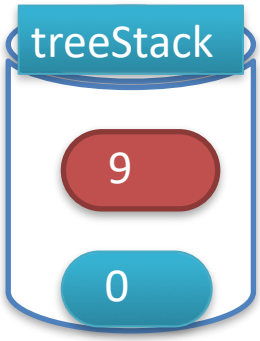
parent => (1)

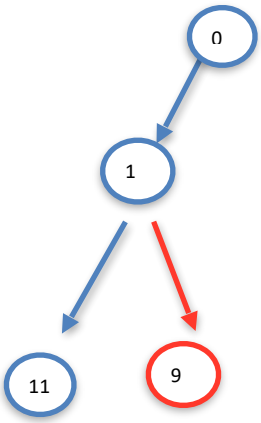
treeStack is not empty and 2>4 = false -> then;

The position of the cell is right of the parent cell.

Push the cell into the stack.

}





i = 4{

datum = 13

cell => (13)

3<2 = false -> then;

treeStack is not empty and 3>2 = true -> then;

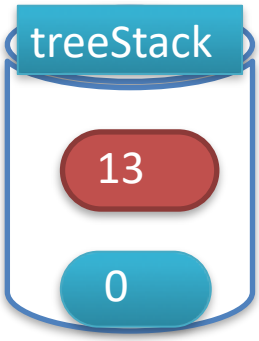
parent =>(13)

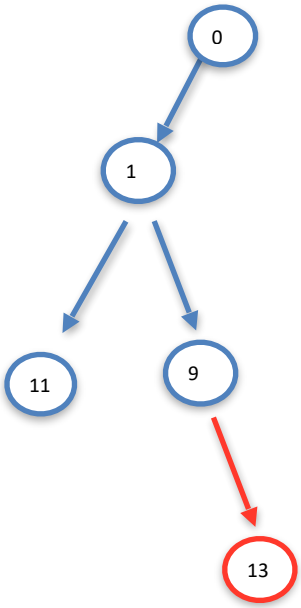
treeStack is not empty and 3>4 = false -> then;

The position of the cell is right of the parent cell.

Push the cell into the stack.

}





i = 5 {

datum = 3

cell=>(3)

6<3 = false -> then;

treeStack is not empty and 6 > 3 = true -> then;

parent => (13)

treeStack is not empty and 6>4 = true -> then;

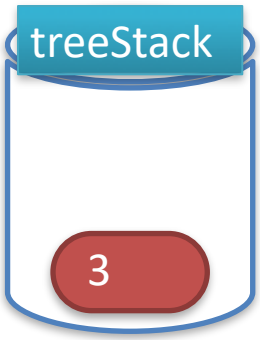
parent =>(0)

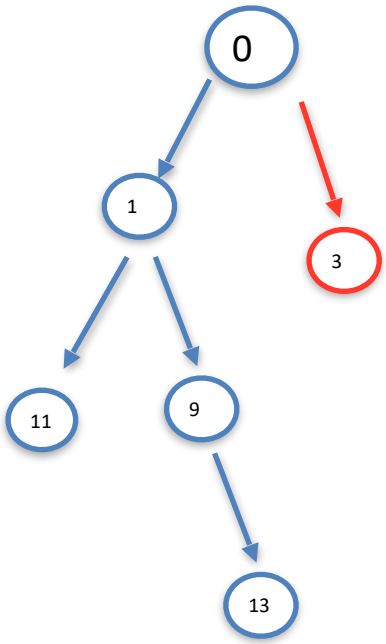
treeStack is empty then->;

The position of the cell is right of the parent cell.

Push the cell into stack.

}





i = 6 {

datum = 6

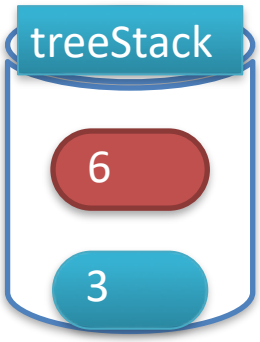
cell=>(6)

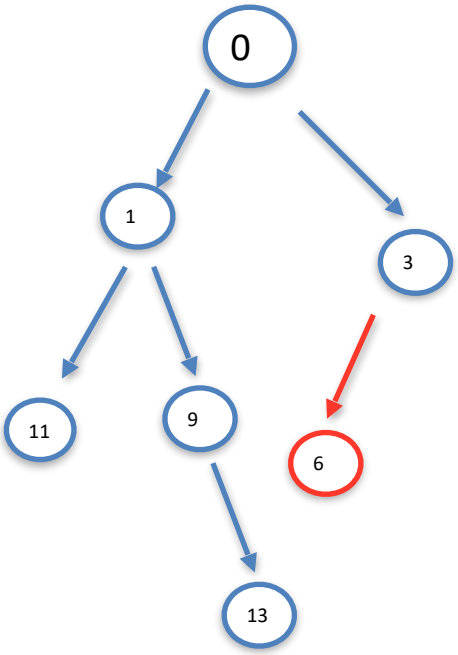
5<6 = true -> then;

The position of the cell is to the left of the topmost element in the stack.

Push the cell into stack.

}





i = 7{

datum = 10

cell=>(10)

7<5 = false -> then;
treeStack is not empty and 7>5 = true -> then;

parent=>(6)

treeStack is not empty and 7>6 = true -> then;

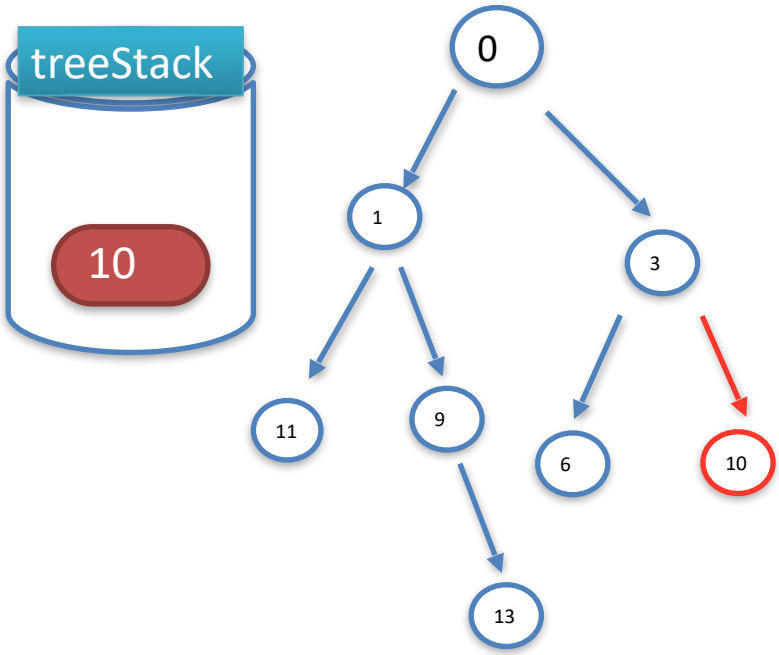
parent=>(3)

treeStack is empty -> then;

The position of the cell is right of the parent cell.

Push cell into the stack.

}



To find the post-order array(Left-Right-Root)

stack1 is not empty then{

treeCell =>(0)

Push the treeCell into the stack2.

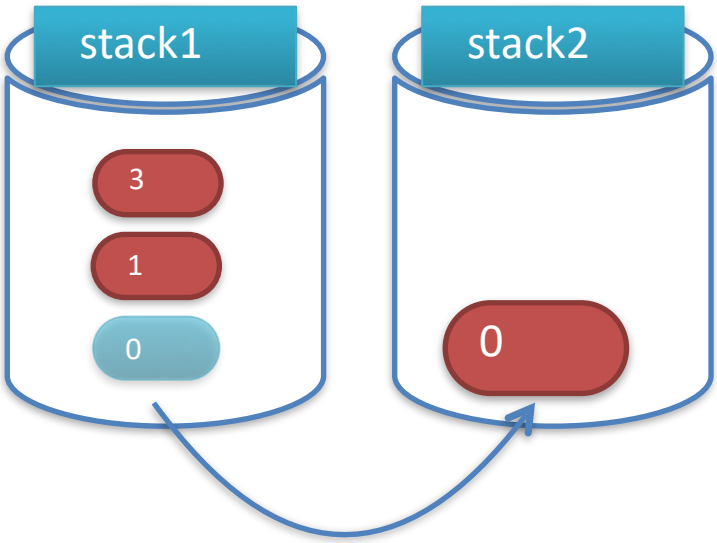
Left of the treeCell is not null then;

Push the left of the treeCell into stack1.

Right of the treeCell is not null then;

Push the right of the treeCell into stack1.

}



stack1 is not empty then{

treeCell=>(3)

Push the treeCell into stack2.

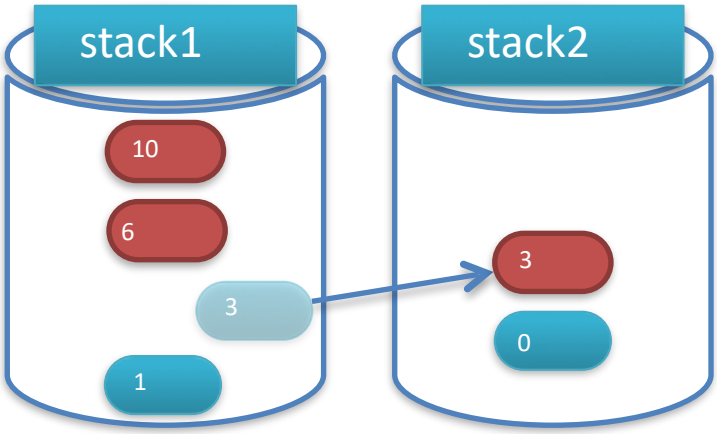
Left of the treeCell is not null then;

Push the left of the treeCell into stack1.

Right of the treeCell is not null then;

Push the right of the treeCell into stack1.

}



stack1 is not empty then{

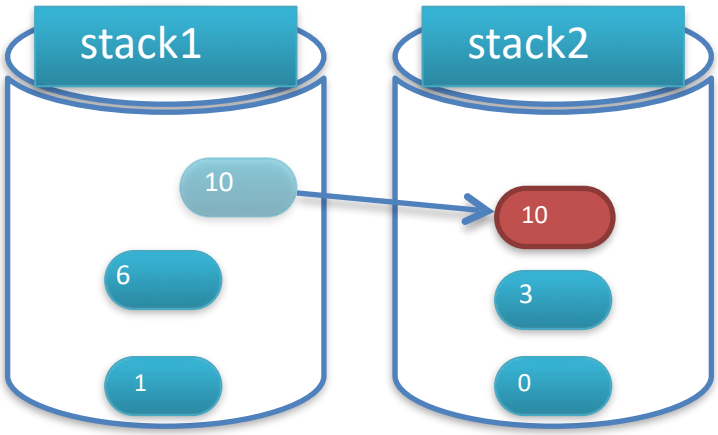
treeCell =>(10)

Push the treeCell into stack2.

Left of the treeCell is null.

Right Of the treeCell is null.

}



stack1 is not empty then{

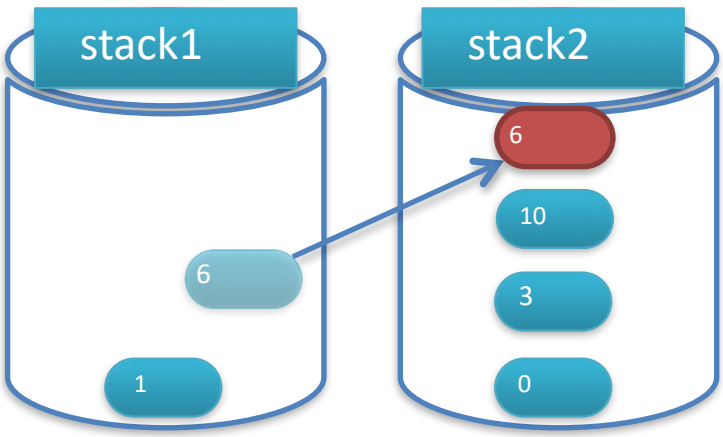
treeCell => (6)

Push the treeCell into stack.

Left of the treeCell is null.

Right Of the treeCell is null.

}



stack1 is not empty then{

treeCell =>(1)

Push the treeCell into stack2.

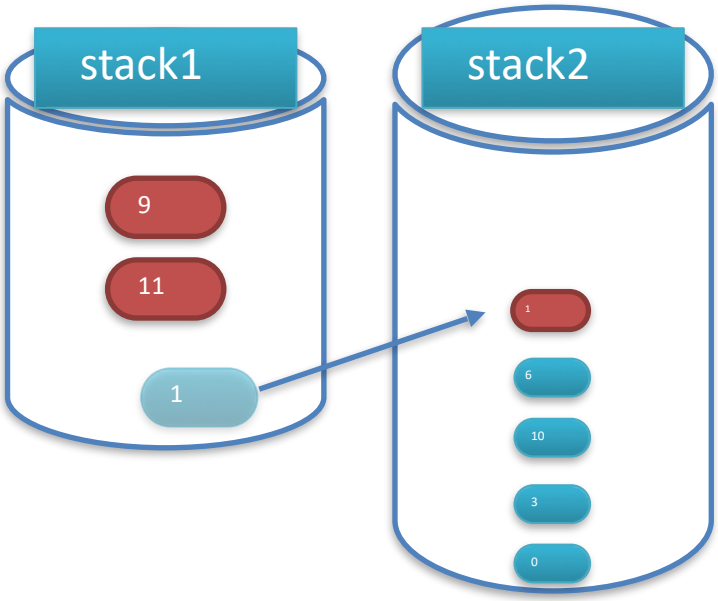
Left of the treeCell is not null then;

Push the left of the treeCell into stack1.

Right of the treeCell is not null then;

Push the right of the treeCell into stack1.

}



stack1 is not empty then{

treeCell =>(9)

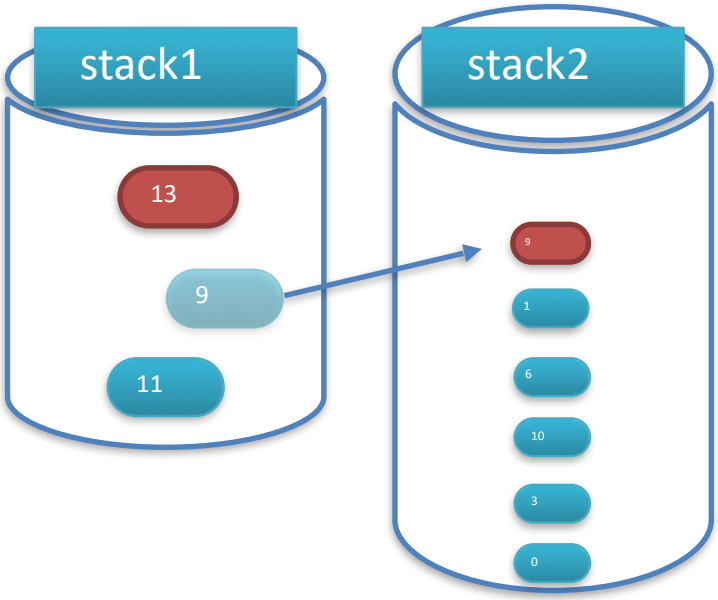
Push the treeCell into stack2.

Left of the treeCell is null then;

Right of the treeCell is not null then;

Push the right of the treeCell into stack1.

}



stack1 is not empty then{

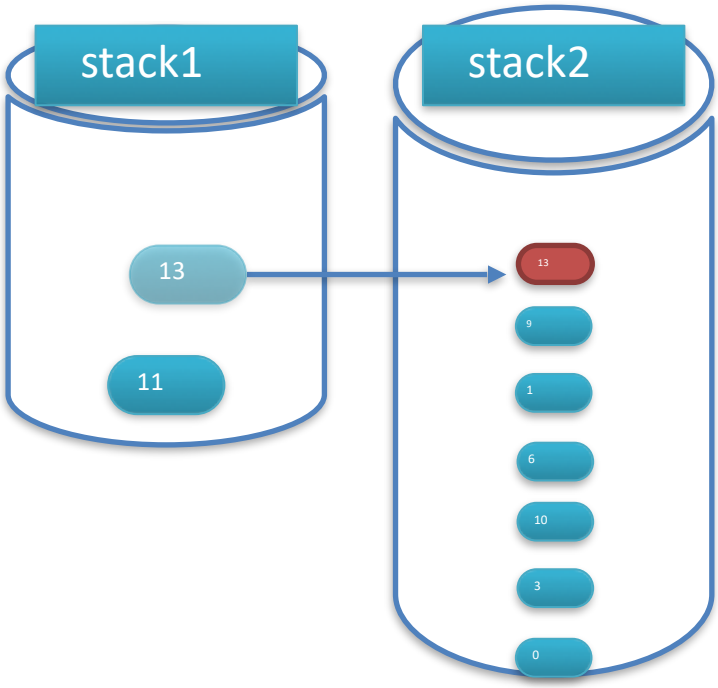
treeCell =>(13)

Push the treeCell into stack2.

Left of the treeCell is null then;

Right of the treeCell is null then;

}



stack1 is not empty then{

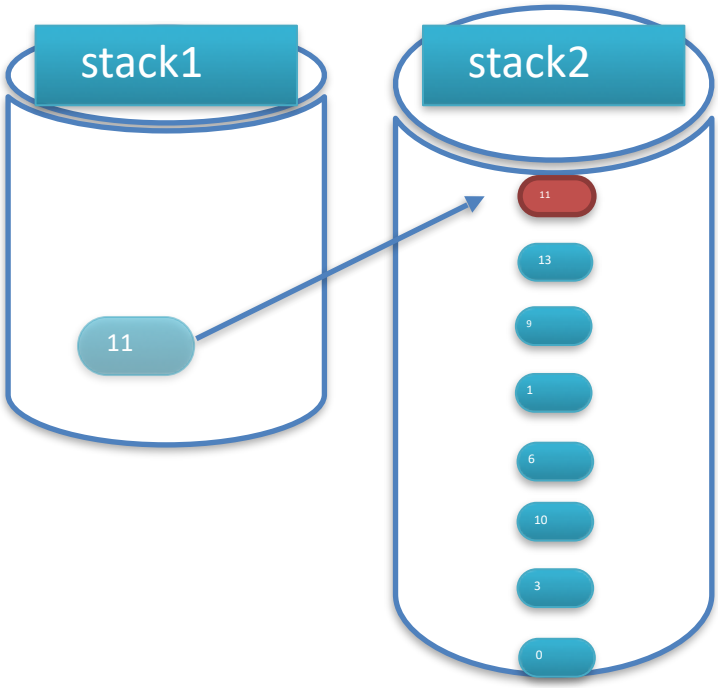
treeCell => (11)

Push the treeCell into stack2.

Left of the treeCell is null then;

Right of the treeCell is null then;

}



Add elements to the postOrder one by one until there are no more elements in stack2.Then return the postOrder;

11	13	9	1	6	10	3	0

In order to solve this question I looked as how to create a tree from preorder and inorder and I understood the logic. Then try to write this code.

Challenge:Insert a node at the head of a linked list

Time Complexity: O(1)

Example Input:

5
383
484
392
975
321

```
    static SinglyLinkedListNode insertNodeAtHead(SinglyLinkedListNode llist, int data) {
        SinglyLinkedListNode tempNode = new SinglyLinkedListNode(data);
        tempNode.next = llist;
        llist = tempNode;
        return llist;

    }
```

Challenge:Insert a Node at the Tail of a Linked List

Time Complexity: O(n)

Example Input:

5
141
302
164
530
474

```
55      static SinglyLinkedListNode insertNodeAtTail(SinglyLinkedListNode head, int data) {
56              SinglyLinkedListNode tempNode = new SinglyLinkedListNode(data);
57
58              if(head == null){
59                  head = tempNode;
60                  return head;
61              }
62              SinglyLinkedListNode current = head;
63              while(current.next!= null){
64                  current = current.next;
65              }
66              current.next = tempNode;
67          return head;
68      }
```

Challenge:Insert a Node at a Specific Position in a Linked List

Time Complexity: O(n)

Example Input:

```
3
16
13
7
1
2
```

```
static SinglyLinkedListNode insertNodeAtPosition(SinglyLinkedListNode head, int data, int position) {
     int count = 0;
    SinglyLinkedListNode tempNode = new SinglyLinkedListNode(data);
    SinglyLinkedListNode current = head;
    while(count<position-1){
        current = current.next;
        count++;
        }
    SinglyLinkedListNode right = current.next;
    current.next = tempNode;
    tempNode.next = right;
    return head;


}
```

Challenge:Delete a Node

Time Complexity: O(n)

Example Input:

```
8
20
6
2
19
7
4
15
9
3
```

```
static SinglyLinkedListNode deleteNode(SinglyLinkedListNode head, int position) {
        int count = 0;
        if(position == 0){
            return head.next;
        }
        SinglyLinkedListNode current = head;
        while(count<position-1){
            current = current.next;
            count++;
        }
        SinglyLinkedListNode right = current.next.next;
        current.next = right;
        return head;
}
```

Challenge:Merge two sorted linked lists

Time Complexity: O(n)

Example

Input:

1
3
1
2
3
2
3
4

```
67      static SinglyLinkedListNode mergeLists(SinglyLinkedListNode head1, SinglyLinkedListNode head2) {
68              SinglyLinkedList linkedList = new SinglyLinkedList();
69
70              while(head1!= null || head2!= null){
71                  if(head1 != null && head2 == null){
72                      linkedList.insertNode(head1.data);
73                      head1 = head1.next;
74                  }
75                  else if(head2 != null && head1 == null){
76                      linkedList.insertNode(head2.data);
77                      head2 = head2.next;
78                  }
79                  else{
80                      if(head1.data<head2.data){
81                          linkedList.insertNode(head1.data);
82                          head1 = head1.next;
83                      }
84                      else if(head1.data == head2.data){
85                          linkedList.insertNode(head1.data);
86                          head1 = head1.next;
87                          linkedList.insertNode(head2.data);
88                          head2 = head2.next;
89                      }
90                      else{
91                          linkedList.insertNode(head2.data);
92                          head2 = head2.next;
93                      }
94                  }
95              }
96          return linkedList.head;
97
98      }
```

Challenge:Tree: Preorder Traversal

Time Complexity: O(n)

Example Input:

```
28      public static void preOrder(Node root) {
29          if(root == null){
30              return;
31          }
32          System.out.print(root.data + " ");
33          preOrder(root.left);
34          preOrder(root.right);
35
36      }
```

Challenge:Tree: Inorder Traversal

Time Complexity: O(n)

Example Input:

```
28      public static void inOrder(Node root) {
29          if(root == null){
30              return;
31          }
32          inOrder(root.left);
33          System.out.print(root.data + " ");
34          inOrder(root.right);
35      }
```

Challenge:Tree: Postorder Traversal

Time Complexity: O(n)

Example Input:

```
28      public static void postOrder(Node root) {
29          if(root == null){
30              return;
31          }
32          postOrder(root.left);
33          postOrder(root.right);
34          System.out.print(root.data + " ");
35
36      }
```