ASSUMPTION UNIVERSITY
Vincent Mary School of Science and Technology
Department of Computer Science


# 1136 Parliament

## Term Project Report

CS3201 Algorithm Design

**ZHENG WANG 5826905**

**Semester 1/2018**

Timus problem analysis and solution. [http://acm.timus.ru/problem.aspx?space=1&num=1136]

# Content

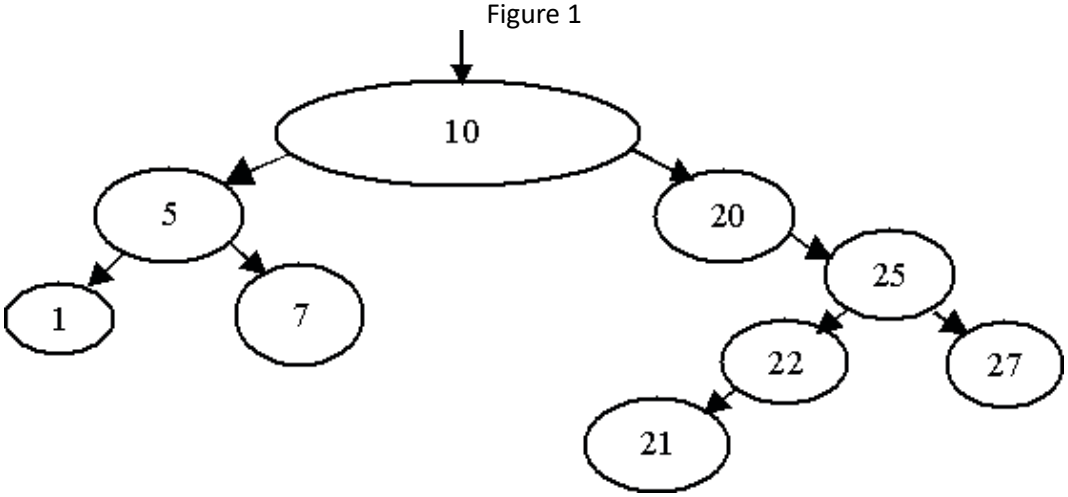
# Problem Description

Time limit: 1.0 second2
Memory limit: 64 MB

A new parliament is elected in the state of MMMM. Each member of the parliament gets his unique positive integer identification number during the parliament registration. The numbers were given in a random order; gaps in the sequence of numbers were also possible. The chairs in the parliament were arranged resembling a tree-like structure. When members of the parliament entered the auditorium they took seats in the following order. The first of them took the chairman's seat. Each of the following delegates headed left if his number was less than the chairman's, or right, otherwise. After that he took the empty seat and declared himself as a wing chairman. If the seat of the wing chairman has been already taken then the seating algorithm continued in the same way: the delegate headed left or right depending on the wing chairman's identification number.

The Figure 1 below demonstrates an example of the seating of the members of parliament if they entered the auditorium in the following order: 10, 5, 1, 7, 20, 25, 22, 21, 27.

Figure 1




During its first session the parliament decided not to change the seats in the future. The speech order was also adopted. If the number of the session was odd then the members of parliament spoke in the following order: the left wing, the

right wing and the chairman. If a wing had more than one parliamentarian then their speech order was the same: the left wing, the right wing, and the wing chairman. If the number of the session was even, the speech order was different: the right wing, the left wing, and the chairman. For a given example the speech order for odd sessions will be 1, 7, 5, 21, 22, 27, 25, 20, 10; while for even sessions — 27, 21, 22, 25, 20, 7, 1, 5, 10.

Determine the speech order for an even session if the speech order for an odd session is given.

## Input

The first line of the input contains N, the total number of parliamentarians. The following lines contain N integer numbers, the identification numbers of the members of parliament according to the speech order for an odd session.

The total number of the members of parliament does not exceed 3000. Identification numbers do not exceed 65535.

## Output

The output should contain the identification numbers of the members of parliament in accordance with the speech order for an even session.

## Sample

| input | output |
|---|---|
| 9<br>1<br>7<br>5<br>21<br>22<br>27<br>25<br>20<br>10 | 27<br>21<br>22<br>25<br>20<br>7<br>1<br>5<br>10 |

**Problem Source:** Quarterfinal, Central region of Russia, Rybinsk, October 17-18 2001
**Difficulty:** 101

# Problem Analysis

## 1. Data Construction

First, the way of parliament members choosing the seats is similar to Binary Search Tree (BST).

**BST Properties**
Let x be a node in a BST.
If y is a node in the left subtree of x, then key[y]≤ key[x].
If y is a node in the right subtree of x, the key[x]≤key[y].

In other words, a subtree with root R, left child l, and right child r has key[l]≤key[R]≤key[r].

The first of them who took the chairman's seat is the root of the BST, and then the other members took the seats following the logic of INSERT operation of BST. The way of constructing data becomes clear now.

## 2. Input Reading

The way of tester giving the numbers is like giving the nodes in Post-order in BST.

**Post-order (LRN)**

1. Check if the current node is empty or null.
2. Traverse the left subtree by recursively calling the post-order function.
3. Traverse the right subtree by recursively calling the post-order function.
4. Display the data part of the root (or current node).

Therefore, the last number of input is the root of BST and previous numbers are the children of it, and then the children of its children, and so on. Thus we can insert the numbers of input in reverse, to the BST.

## 3. Output Solution

The required output is to print out the numbers in Right-Left-Parent order. So we can implement it similar to another style of TREE-WALK of BST, following the order

of Right-Left-Parent.

# Running time Analysis

In general cases, the running time of the INSERT part is O($\log n$). In worst case, the running time will be O($n^2$), when the numbers are contiguous ascending or descending.

The running time of TREE-WALK is O(n) in all cases.

# Program Implementation

## 1. Node

First, the class Node, has the field `key` to store the identification number of parliamentarian, the field `left` point to its left parliamentarian whose identification number is smaller than its, the field `right` point to its right parliamentarian whose identification number is greater than its.

```
class Node{
    int key;
    Node left;
    Node right;

    public Node(int key){
        this.key = key;
        this.left = null;
        this.right = null;
    }
}
```

## 2. BinarySearchTree

Second, the class `BinarySearchTree` has the field `root`, which is the first Node come into the tree, and the other operation method.

The method `insert` and `_insert` operate insertion of the BST. If the BST is empty, set the current coming in Node, which is also the first Node as the root of BST, otherwise, compare the coming in key with the current Node's key. If the coming in key is smaller than or equal to the current Node's key then check the current Node's left is empty or not, and check the current Node's right otherwise. Then if the left or right is not empty, repeat check the coming in key with the Node's key, until it is empty and allocate on there.

The method `printRePost` and `_printRePost` operate the tree walk technic that deal with the output. First it will check if the BST is empty or not in case it has runtime error with null type, then it start with the root and repeat check the current Node's right is null or not, if not repeat, same as the current Node's left, and then print it self's key.

```
class BinarySearchTree{
    private Node root;

    public BinarySearchTree(){
        this.root = null;
    }

    public void insert(int key){
        if(this.root == null){
            this.root = new Node(key);
        }else{
            _insert(this.root, key);
        }
    }

    public void _insert(Node currentN, int key){
        if(key <= currentN.key){
            if(currentN.left != null){
                _insert(currentN.left, key);
            }else{
                currentN.left = new Node(key);
            }
        }else{
            if(currentN.right != null){
                _insert(currentN.right, key);
            }else{
                currentN.right = new Node(key);
            }
```

```
        }
    }

    public void printRePost(){
        if(this.root != null){
            _printRePost(this.root);
        }
    }

    public void _printRePost(Node currentN){
        if(currentN != null){
            _printRePost(currentN.right);
            _printRePost(currentN.left);
            System.out.println(currentN.key);
        }
    }
}
```

## 3. Input and Output

Last, we read the input as the first line input N is the total number of Members of Parliament, which is the length of array `parlis` that we store the numbers, then repeat store the numbers to the array. Next we insert the numbers stored in `parlis` to the `tree` which we defined as `BinarySearchTree`, so that it will automatically construct the tree. Then print the tree in right, left , root order as it required.

```
import java.util.Scanner;
public class Parliament {
    static Scanner s = new Scanner(System.in);
    static BinarySearchTree tree = new BinarySearchTree();
    public static void main(String[] args) {

        int N = s.nextInt();
        int[] parlis = new int[N];
       for(int i = 0; i<N; i++){
           parlis[i] = s.nextInt();
       }

       for(int n = N-1; n>= 0; n--){
           tree.insert(parlis[n]);
       }

       tree.printRePost();
       s.close();
```

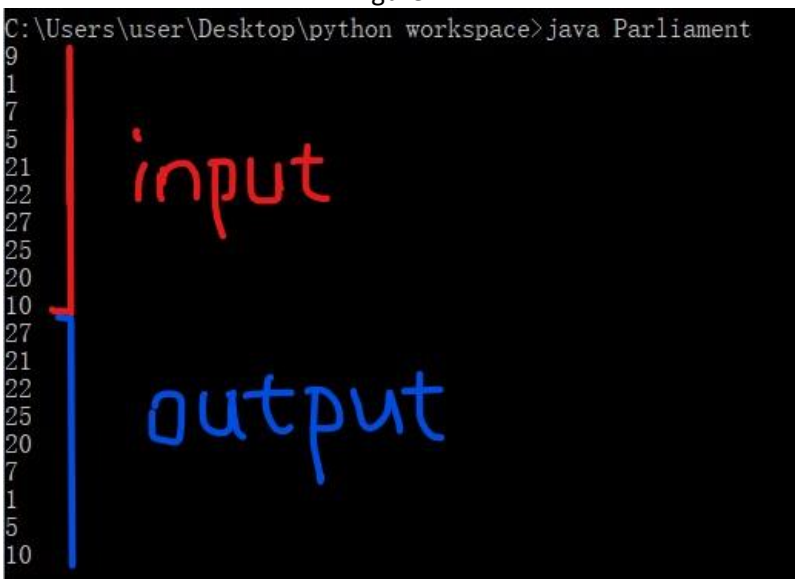
```
    }
}
```

# Execution Examples

**Test Case:** 9 1 7 5 21 22 27 25 20 10
(The same as the sample in project description)

**Test Result:**
Show as Figure 2.

Figure 2




# Judgement Result

Show as Figure 3.

Figure 3

| ID | Date | Author | Problem | Language | Judgement result | Test # | Execution time | Memory used |
|---|---|---|---|---|---|---|---|---|
| 8148706 | 13:35:22 25 Nov 2018 | ZHENG WANG | 1136. Parliament | Java 1.8 | Accepted | | 0.218 | 5 144 KB |