이진검색트리(Recursion)

다음과 같은 노드 구조를 가진 이진검색트리의 관련함수를 모두 재귀함수로 구현하는 프로그램을 작성하시오.

```
struct node
{
    int data;              // 노드에 저장되는 정수 데이터
    struct node* left;     // 왼쪽 서브트리
    struct node* right;    // 오른쪽 서브트리
};
```

(1) `void insert(struct node** root, int data);`
루트노드가 `root` 인 이진검색트리에 데이터 `data` 를 입력하는 함수.
각 노드는 동적으로 메모리를 할당하여 구현한다.

(2) `void preOrder(struct node* root);`
루트노드가 `root` 인 이진검색트리를 전위(preorder) 탐색하면서 모든 노드에 저장된 데이터를 한 줄에 출력.

(3) `void inOrder(struct node* root);`
루트노드가 `root` 인 이진검색트리를 중위(inorder) 탐색하면서 모든 노드에 저장된 데이터를 한 줄에 출력.

(4) `void postOrder(struct node* root);`
루트노드가 `root` 인 이진검색트리를 후위(postorder) 탐색하면서 모든 노드에 저장된 데이터를 한 줄에 출력.

(5) `int size(struct node* root);`
루트노드가 `root` 인 이진검색트리의 모든 노드의 개수를 계산함.

(6) `int height(struct node* root);`
루트노드가 `root` 인 이진검색트리의 높이를 계산함.

(7) `int sumOfWeight(struct node* root);`
루트노드가 `root` 인 이진검색트리의 모든 노드에 저장된 데이터의 합을 계산함.

(8) `int maxPathWeight(struct node* root);`
루트노드가 `root` 인 이진검색트리의 루트노드부터 단말노드까지의 경로 상의 모든 노드에 저장된 데이터의 합이 가장 큰 합을 계산함.

(9) `void mirror(struct node** root);`
루트노드가 `root` 인 이진검색트리를 미러 이지미가 되도록 노드의 순서를 변환함.

(9) `void destruct(struct node** root);`
루트노드가 `root` 인 이진검색트리의 동적으로 메모리 할당된 노드를 해제하여 이진검색트리를 소멸시킴.

입력

입력은 표준입력(standard input)을 사용한다. 입력은 t개의 테스트 케이스로 주어진다. 입력의 첫 번째 줄에 테스트 케이스의 개수를 나타내는 정수 t가 주어진다. 두 번째 줄부터 t 개의 줄에는 한 줄에 한 개의 테스트 케이스에 해당하는 데이터가 입력된다. 각 줄에서 첫 번째로 입력되는 정수 $n\,(1 \le n \le 100)$은 이진검색트리에 입력되는 정수 데이터의 개수를 나타낸다. 그 다음으로는 n 개의 정수가 입력된다. 이 정수들은 최소 1 이며 최대 10,000 이며, 또한 이 정수들은 모두 다른 수이다. 각 정수들 사이에는 한 개의 공백이 있으며, 잘못된 데이터가 입력되는 경우는 없다.

출력

출력은 표준출력(standard output)을 사용한다. 입력되는 테스트 케이스의 순서대로 다음 줄에 이어서 각 테스트 케이스의 결과를 출력한다. 각 테스트 케이스의 출력되는 첫 줄에 입력으로 주어진 데이터에 대하여 이진검색트리의 함수들이 실행된 결과를 출력한다.

입력과 출력의 예

입력	출력
3 5 3 4 5 1 2 1 1 5 1 2 3 4 5	5 2 15 12 3 4 5 1 2 5 4 3 2 1 5 4 2 1 3 0 1 0 1 1 1 1 1 0 5 4 15 15 1 2 3 4 5 5 4 3 2 1 5 4 3 2 1 0

BinarySearchTree.c

```
#include <stdio.h>

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

void insert(struct node** root, int data);
void preOrder(struct node* root);
void inOrder(struct node* root);
void postOrder(struct node* root);
int size(struct node* root);
int height(struct node* root);
int sumOfWeight(struct node* root);
int maxPathWeight(struct node* root);
void mirror(struct node** root);
```

```
void destruct(struct node** root);

int main()
{
    int numTestCases;

    scanf("%d", &numTestCases);
    while (numTestCases--)
    {
        int num, i;
        struct node* root = NULL;

        scanf("%d", &num);
        for (i = 0; i < num; i++)
        {
            int data;

            scanf("%d", &data);
            insert(&root, data);
        }
        printf("%d\n", size(root));
        printf("%d\n", height(root));
        printf("%d\n", sumOfWeight(root));
        printf("%d\n", maxPathWeight(root));
        mirror(&root);
        preOrder(root); printf("\n");
        inOrder(root); printf("\n");
        postOrder(root); printf("\n");
        destruct(&root);   // BST의 모든 노드의 동적 메모리 해제
        if (root == NULL)
            printf("0\n");
        else
            printf("1\n");
    }

    return 0;
}

// 데이터 삽입(recursion)
void insert(struct node** root, int data)
{
}

// 전위(preorder)탐색(recursion)
void preOrder(struct node* root)
{
...
    printf("%d ", root->data);
...
}

// 중위(inorder)탐색(recursion)
void inOrder(struct node* root)
{
...
    printf("%d ", root->data);
```

```
...
}

// 후위(postorder)탐색(recursion)
void postOrder(struct node* root)
{
...
    printf("%d ", root->data);
...
}

// 노드의 개수(recursion)
int size(struct node* root)
{
}

// 높이(recursion)
int height(struct node* root)
{
}

// 미러 이미지로 변환하기(recursion)
void mirror(struct node** root)
{
}

// 노드에 저장된 데이터의 값의 합 구하기(recursion)
int sumOfWeight(struct node* root)
{
}

// 루트노드부터 단말노드까지의 경로 상의 데이터의 최대합(recusrion)
int maxPathWeight(struct node* root)
{
}

// 트리노드의 동적 메모리 해제하기(recursion)
void destruct(struct node** root)
{
}
```