COMP2119A Introduction to Data Structures and Algorithms
Programming Assignment 2 **Due Date:** 7pm, Dec 10, 2019

Rules: discussion of the problems is permitted, but writing the assignment together is not (i.e. you are not allowed to see the actual solution of another student).

Course Outcomes

- **[O4]. Implementation**

[O4] In this assignment, you are requested write programs to solve two tasks, **A** and **B**.

Your program will be judged by a computer automatically, please follow the exact format. You should read from standard input and write to standard output. e.g, scanf/print, cin/cout.

Languages.

We only accept C/C++ programming languages. Undefined Behavior in C and C++ is not allowed.

Judging.

Please note that your solution is automatically judged by a computer.

Solutions that fail to compile or execute get 0 points.

We have designed 20 test cases, 10 for each task. Each test case is 5 points.

The time limit for each test case is 1 second.

Self Testing.

You should test your program by yourself using the provided sample input/output file.

The sample input/output is **different** from the ones used to judge your program, and it is designed for you to test your program by your own.

Note that your program should always use standard input/output.

To test your program in Windows:
1. Compile your program, and get the executable file, "main.exe"
2. Put sample input file, "input.txt", in the same directory as "main.exe"
3. Open command line and go to the directory that contains "main.exe" and "input.txt"
4. Run "main.exe < input.txt > myoutput.txt"
5. Compare myoutput.txt with sample output file. You may find "fc.exe" tool useful.
6. Your output needs to be **exactly** the same as the sample output file.

To test your program in Linux or Mac OS X:
1. Put your source code "main.cpp" and sample input file "input.txt" in the same directory.
2. Open a terminal and go to that directory.
3. Compile your program by "g++ main.cpp -o main"
4. Run your program using the given input file, "./main < input.txt > myoutput.txt"
5. Compare myoutput.txt with sample output file.

6. Your output needs to be **exactly** the same as the sample output file.
7. You may find the **diff** command useful to help you compare two files. Like "diff -w myOutput.txt sampleOutput.txt". The $-w$ option ignores all blanks (SPACE and TAB characters)

Note that myoutput.txt file should be **exactly** the same as sample output. Otherwise it will be judged as wrong.

Submission.

Please submit your source files through moodle.

You should submit two source files(**without** compression), one for each task.

Please name your files in format UniversityNumber-X.cpp for X in {A, B}, e.g. 1234554321-A.cpp.

Task A
Given a binary tree, determine if it is a valid binary search tree (BST).

Assume a BST is defined as follows:

- The left subtree of a node contains only nodes with keys strictly **less** than the node's key.
- The right subtree of a node contains only nodes with keys strictly **greater** than the node's key.
- Both the left and right subtrees must also be binary search trees.

Note: All node values of the tree are positive integers. We use '0' to stand for 'NULL'.

You can define a binary tree according to the following:

```
// Definition for a binary tree node.
 struct TreeNode {
     int val;
     TreeNode *left;
     TreeNode *right;
     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 };
```

Input:
The input contains two lines:
The first line is an integer n, which is the number of integers in the second line. $n = 2^r - 1, 1 \leq r \leq 20$.
The second line are n non-negative integers, standing for nodes of the tree. (You are required to input the node by layer, at each layer, from left to right.)

Note: All nodes are positive integers, and less than 2^{31}. You may assume the input always forms a complete binary tree. (0's mean the tree nodes do not exist.)

Output:

If the input is a valid BST, output 'true'; Otherwise, output 'false'.

Examples:

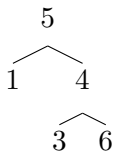
1.

Input:
3
2 1 3

Output:

true

2.

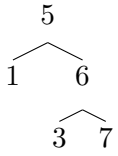


Input:
7
5 1 4 0 0 3 6

Output:

false

3.



Input:
7
5 1 6 0 0 3 7

Output:

false

Task B

Implement an AVL tree. You should implement the operations using exactly the same algorithms as we taught in class. Starting from an empty tree, you should apply the operations to the tree one by one. The operations are described below.

1. Insertion. A key x is given, and you need to insert it to the AVL tree. You may assume x is not currently in the tree. Here x is an integer, $0 \leq x \leq 2^{31} - 1$.
2. Deletion. A key x is given, and you need to delete it from the current tree. You may assume x is currently in the tree. Here x is an integer.
3. KthMin. Given an integer k, output the k-th smallest element in current tree. You may assume k always satisfies $\mathbf{1 \leq k \leq}$ ***number of nodes in AVL tree***.

Note: When deleting a node with two children, you should replace it with its immediate successor. Then you can delete that node.

Notice for Rotation: Please read the text-book carefully for the conditions of rotation. Below is one of the conditions that you need to be careful. Suppose node r is not balanced, say $height(r.left) == height(r.right) - 2$. If we have $height(r.right.left) == height(r.right.right)$, then you should only do a single rotation. A double rotation would break the tree in some cases.

Notice for Storing Height Information Balance factors can be kept up-to-date by knowing the previous balance factors and the change in height – it is not necessary to know the absolute height, **two bits per node are sufficient**.

Input:

The input contains multiple lines. The last line contains the word "END", which means end of input. You program should exit after reading this line. Except the last line, each line corresponds to an operation.
The format of each line is described as below.

1. **Insertion.** Character 'A' followed by a space then an integer x. e.g. "A 10". You should insert the key x into the AVL tree.
2. **Deletion.** Character 'D' followed by a space then an integer x. e.g. "D 10". You should delete the key x from the AVL tree.
3. **KthMin.** A single character 'M' followed by a space then an integer k. e.g. "M 1". You should output the k-th smallest element in the current tree. Output a newline after **each** element.

The number of operations in the input is less than 170000.

Output:

Output one integer followed by a newline for each KthMin operation.
A sample input/output is given below.

Example:

Input:

```
A 10
A 5
A 1
M 1
A 20
A 19
M 4
D 1
M 1
M 2
M 3
M 4
END
```

Output:

```
1
19
5
10
19
20
```