Interval Deletion in Binary Search Tree

Distinguished professor Fabinaris Suchbaum from Max Planck Institute for Software Systems in Saarbrücken is a well recognized expert on binary search trees (BST) and their improvements. Recently he has been contacted by Data4Ever company. To improve the performance of their latest database, the company needs an efficient data structure which supports a deletion of records whose key is within a given range of values. The solution delivered by professor Suchbaum is a binary search tree where the requested operation is designed as follows. Given a BST, which contains distinct keys, and interval I, the interval deletion works in two phases. During the first phase it removes all nodes whose key is in I and all edges adjacent to the removed nodes. Let the resulting graph contain k connected components $T_1,...,T_k$. Each of the components is a BST where the root is the node with the smallest depth among all nodes of this component in the original BST. We assume that the sequence of trees T_i is sorted so that for each $i < j$ all keys in T_i are smaller than keys in T_j. During the second phase, trees T_i are merged together to form one BST. We denote this operation by $Merge(T_1,...,T_k)$. Its output is defined recurrently as follows:

1. For an empty sequence of trees, $Merge()$ gives an empty BST.
2. For a one-element sequence containing a tree T, $Merge(T) = T$.
3. For a sequence of trees $T_1,...,T_k$ where $k > 1$, let $a_1 < a_2 < ... < a_n$ be the sequence of keys stored in the union of all trees $T_1,...,T_k$, sorted in ascending order. Moreover, let $m = \lfloor (1+k)/2 \rfloor$ and let T_s be the tree which contains a_m. Then, $Merge(T_1,...,T_k)$ gives a tree T created by merging three trees T_s, $T_L = Merge(T_1,...,T_{s-1})$ and $T_R = Merge(T_{s+1},...,T_k)$. These trees are merged by establishing the following two links: T_L is appended as the left subtree of the node storing the minimal key of T_s and T_R is appended as the right subtree of the node storing the maximal key of T_s.

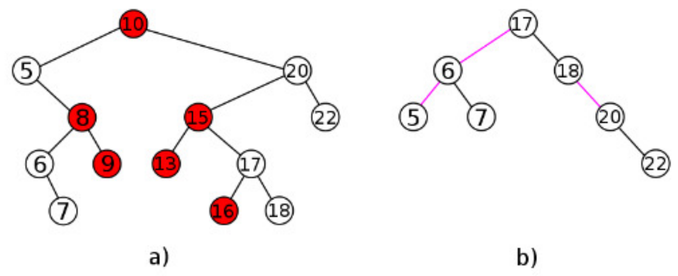

Image 1. a) A binary search tree containing 13 distinct keys. Interval deletion is called for the range [8, 16]. The nodes that are removed by the first phase are colored in red. Four connected components emerge: T_1 storing key 5, T_2 storing keys 6 and 7, T_3 storing keys 17 and 18, and T_4 storing keys 20 and 22. b) $Merge(T_1, T_2, T_3, T_4)$ is applied during the second phase. The magenta edges are added to form the resulting BST T. Since (1+7)/2=4 and the fourth key in the sequence 5,6,7,17,18,20,22 is key 17, which belongs to T_3, the root of T_3 becomes the root of T.

The task

Implement the operation of interval deletion. Test the operation by building a BST using N inserts, followed by performing one interval deletion.

Input

The first input line contains integer N determining the number of keys to be inserted into an empty BST. The second input line contains N distinct integers separated by spaces. The integers represent keys to be inserted into the BST (in the order they are listed in). The third input line contains integers $R_{\min}$ and $R_{\max}$, separated by a space. The interval deletion removes each inserted key K such that $R_{\min} \leq K \leq R_{\max}$. The keys to be inserted are values from 1 to 10^7. It is ensured that the resulting BST has at least 2 nodes. It holds $R_{\min} \leq R_{\max}$ and $N \leq 10^6$.

Output

The output consists of one line containing integers D and H separated by space. Integer D is the depth of the final BST, obtained after performing N inserts and one interval deletion. Integer H is the total number of nodes of depth $D-1$ in the final BST.

Example 1

Input

```
6
3 1 4 2 6 7
3 5
```

Output

```
3 1
```

Example 2

Input

```
13
10 5 8 6 9 7 20 15 22 13 17 16 18
8 16
```

Output

```
3 3
```

Example 2 is visualized in Image 1.

Example 3

Input

```
20
187 146 95 128 200 11 186 179 189 135 94 16 51 45 105 194 145 68 57 77
150 190
```

Output

```
7 2
```

Public data

The public data set is intended for easier debugging and approximate program correctness checking. The