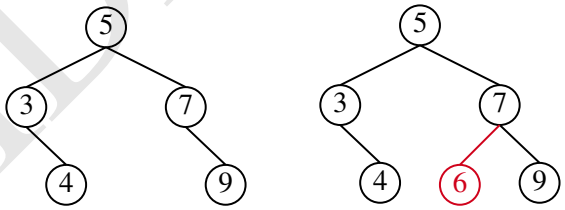
Tree Tweaking

Time limit: 1 second

In computer science, a binary search tree is a binary tree where each node is associated with a key; the key of a node is greater than all keys in its left subtree but smaller than all in its right subtree. Binary search trees are commonly used to implement dynamic sets, which support inserting or deleting data items, as well as searching for them.

When searching for a specific key from a set of keys, a binary search tree usually performs better than the naive search algorithm, since when comparing with the value in a node and walking down into a subtree, we don't need to compare the key with all values in the other subtree. The average number of comparisons made is logarithmic to the size of the tree if the binary search tree is balanced. However, it is not so effective if the tree is unbalanced; in extreme cases, the time complexity degenerates to linear. Malicious attackers may leverage this vulnerability by crafting a bad key insertion sequence. If the binary search tree is not self-balanced, the resulting tree may be biased, which significantly undermines the performance of the application.

Consider a scenario where you run a server that maintains a dynamic set implemented by a binary search tree. The binary search tree adopts a simple insertion algorithm, which inserts the key as a new leaf in the appropriate position without moving existing nodes. Figure 1 shows how a new key is inserted into a binary search tree using this simple insertion algorithm.



(a) Before insertion (b) After insertion

Figure 1: A running example of the simple insertion algorithm.

.

Now a client is ready to send a stream of keys to the server, and the server inserts the keys one by one. You are allowed to reorder a specific range of insertions, such that the resulting tree is as balanced as possible. Please report the minimum possible sum of depths of the nodes in the resulting tree after reordering part of the insertions. The depth of a node is defined as the number of nodes in the path from that node to the root.

Input

The first line of the input contains a single integer n $(1 \leq n \leq 10^5)$, denoting the number of keys to insert. The second line of the input contains n integers $a_1, a_2, \cdots, a_n$ $(1 \leq a_i \leq n)$, denoting the keys to insert in order. All keys inserted are distinct. The third line contains two integers l, r, $(1 \leq l \leq r \leq n, r - l + 1 \leq 200)$, denoting the range of insertions you can reorder, i.e., you can permute the insertions of keys $a_l, a_{l+1}, \cdots, a_r$.

Output

Output a single integer in one line, denoting the minimum possible sum of depths of the nodes in the resulting binary search tree.

Examples

standard input	standard output
8 2 4 5 7 1 3 8 6 3 6	24
5 5 1 2 3 4 3 5	14
7 3 2 4 6 7 5 1 1 7	17

Note

For the first sample data, one optimal insertion sequence after rearrangement is `[2,4,1,7,3,5,8,6]`. The resulting binary search tree is shown in Figure 2.

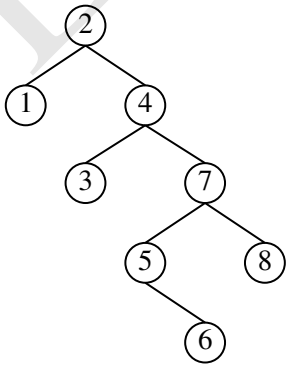


Figure 2: The optimal binary search tree in the first sample data.