**Complexity analysis:**

# Naïve approach

The time complexity of the brute force search into lists is O(n). Since we are searching elements inside two list, we have a total wort case complexity of T(n) = O(n²).

# Using a suitable data structure

## Red-Black Tree

The time complexity of the search into a red-black tree is O(log(n)). The worst case is when we must search into the tree n times. So we have a total wort case complexity of T(n) = O(log(n)²).

## Hash Table

Although searching for an element in a hash table can take as long as searching for an element in a linked list (O(n) time worst case), in practice, hashing performs extremely well. Under reasonable assumptions, the average time to search for an element in a hash table is O(1).

In our case we are performing n times an insertion with a worst case time complexity of O(n) and n times a search with the same time complexity. We thus obtain an overall complexity of O(n).

# Complexity comparison

[Hash Table](http://geeksquiz.com/hashing-set-1-introduction/) supports following operations in Θ(1) time.  
1) Search  
2) Insert  
3) Delete

The time complexity of above operations in a Red-Black Tree is O(Logn). So Hash Table seems to beating Red-Black Tree in all common operations.

Using Inorder Traversal of Red-Black Tree, we can easily get all keys in sorted order where this is not a natural operation in Hash Tables. It is also easier to find the closest lower or greater elements and to do range queries with Red-Black Tree. Again, these operations are not natural with Hash Tables. On thing we can also see is that Red-Black Trees are easier to implement than Hash Tables. Finally, with a Red-Black tree we are guaranteed to work in O(Log(n)) time. But with Hashing, O(1) is the average time and some particular operations may be costly, especially table resizing happens.

# Combine any number of command