My idea:

1. Create unordered\_set for nums1, nums2.
2. Traverse unordered\_set 1, check if item exist unordered\_set 2 => if not existed => push to vector result 1.
3. Do the same thing with unoreder\_set 2.

World idea:

**Solution**

**Overview**

We are given two integer arrays, nums1 and nums2, and need to return a list of two lists. The first list has the elements that are present only in nums1, while the second list has the elements that are present only in nums2.

**Approach 1: Brute Force**

**Intuition**

To find the elements in a list that are not present in another list, we can loop over every element in the first list and for each element we loop over the elements in the second list to check if it's present or not. If we find the element, we will not store it in the answer list; otherwise, we can store it.

This way, we will have to apply the above method twice once for the elements that are only in nums1 and then again for the elements that are only present in nums2.

**Algorithm**

1. Define method getElementsOnlyInFirstList, this method accepts two lists of integers and returns the elements that are present only in the first argument:
   * This method uses the same algorithm that we discussed: loop over the first list and for every element, check the second list. If the element is found in the second list we break and mark a boolean variable existInNums2 as true.
   * After checking all the elements in the second list, if the variable existInNums2 is false, only then we store the element in the list onlyInNums1.
   * Return onlyInNums1.
2. Call the method getElementsOnlyInFirstList once for the param (nums1, nums2) and then again for (nums2, nums1).

**Implementation**



**Complexity Analysis**

Here, NN*N* is the length of list nums1, and MM*M* is the length of nums2.

* Time complexity: O(N∗M)O(N \* M)*O*(*N*∗*M*).

In the first call to getElementsOnlyInFirstList, we iterate over the first list and, for each element, iterate over the second, which costs us N∗MN \* M*N*∗*M* operations. Then again, doing it for the other pair (nums2, nums1), the total operations will be M∗NM \* N*M*∗*N*. Hence the total time complexity would be O(N∗M)O(N \* M)*O*(*N*∗*M*).

* Space complexity: O(1)O(1)*O*(1).

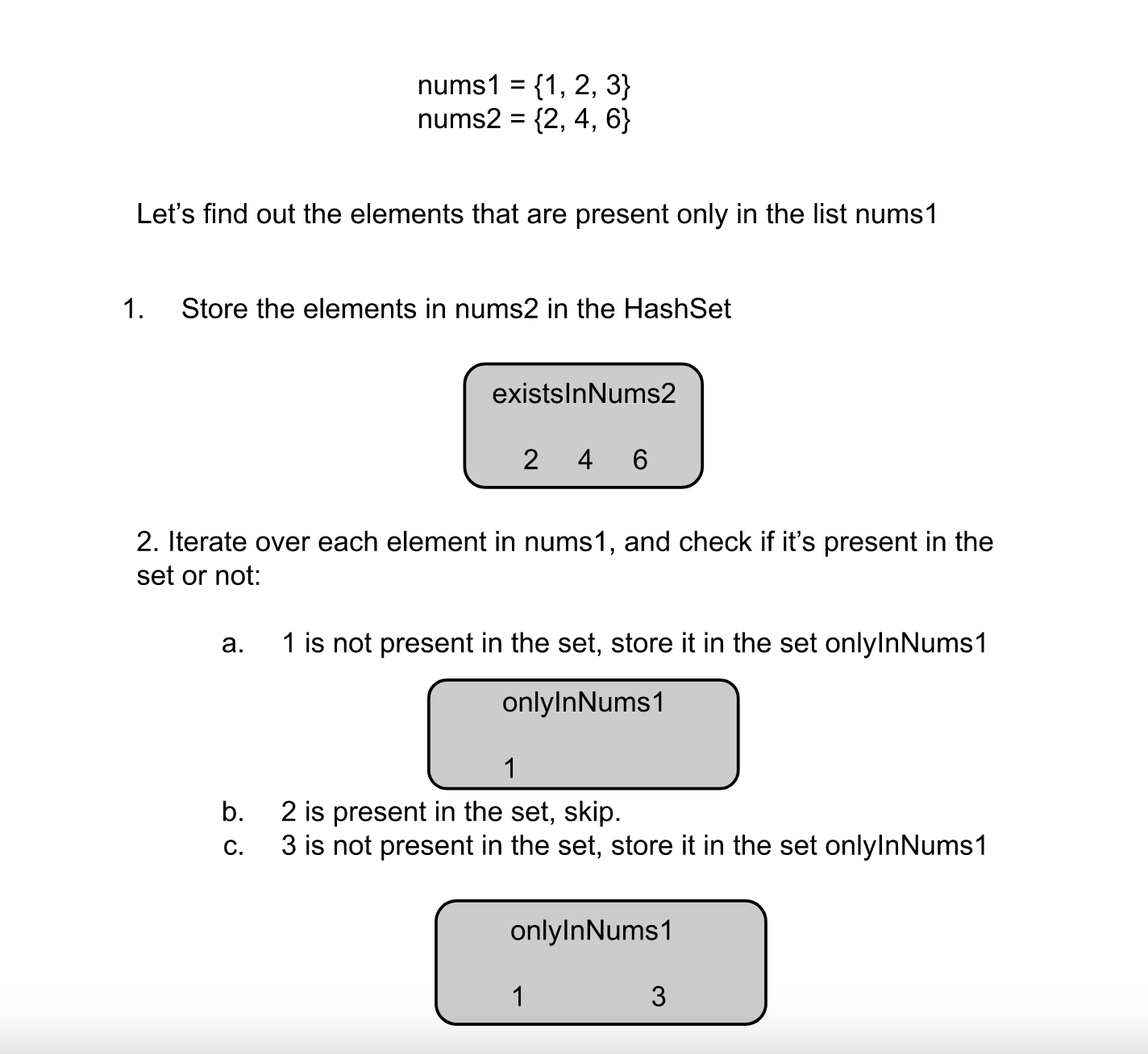
The only space required (two lists of size NN*N* and MM*M*) is to store the output list that is not considered part of the space complexity. Hence, the total space complexity would be constant.

**Approach 2: HashSet**

**Intuition**

Instead of iterating over each element in the second array to check if it exists in the list or not, we can store the elements in a HashSet. Then we can find if an element exists in the list or not in O(1)O(1)*O*(1) time compared to O(N)O(N)*O*(*N*) time in the previous approach.

In this approach, we follow the above intuition. To find the elements that only exist in nums1, we first store the elements in nums2 in the HashSet. Then we iterate over each element in the list nums1, and for each element, we check if it's there in the HashSet; if yes, we skip the element; otherwise, we store it in the list onlyInNums1.



**Algorithm**

1. Define method getElementsOnlyInFirstList. This method accepts two lists of integers and returns the elements that are present only in the first argument:
   * Iterate over the elements in the second argument nums2 and store each in the Hashset existsInNums2.
   * Iterate over the elements in the first argument nums1 and for each element check if it's present in the set existsInNums2. If yes, skip to the next element; otherwise, store it in the set onlyInNums1.
2. Call the method getElementsOnlyInFirstList once for the param (nums1, nums2) and then again for (nums2, nums1).

**Implementation**

class Solution {

public:

    // Returns the elements in the first arg nums1 that don't exist in the second arg nums2.

    vector<int> getElementsOnlyInFirstList(vector<int>& nums1, vector<int>& nums2) {

        unordered\_set<int> onlyInNums1;

        // Store nums2 elements in an unordered set.

        unordered\_set<int> existsInNums2;

        for (int num : nums2) {

            existsInNums2.insert(num);

        }

        // Iterate over each element in the list nums1.

        for (int num : nums1) {

            if (existsInNums2.find(num) == existsInNums2.end()) {

                onlyInNums1.insert(num);

            }

        }

        // Convert to vector.

        return vector<int> (onlyInNums1.begin(), onlyInNums1.end());

    }

    vector<vector<int>> findDifference(vector<int>& nums1, vector<int>& nums2) {

        return {getElementsOnlyInFirstList(nums1, nums2), getElementsOnlyInFirstList(nums2, nums1)};

    }

};

**Complexity Analysis**

Here, NN*N* is the length of list nums1, and MM*M* is the length of nums2.

* Time complexity: O(N+M)O(N + M)*O*(*N*+*M*).

For each of the two calls to getElementsOnlyInFirstList we create a hash set from the elements in the second list, which takes linear time. Then we iterate over the elements in the first list and check in the set if it's present. The find operation in the set takes O(1)O(1)*O*(1) time. Hence, the total time complexity would be O(N+M)O(N + M)*O*(*N*+*M*).

* Space complexity: O(max(N,M))O(max(N, M))*O*(*max*(*N*,*M*)).

The method getElementsOnlyInFirstList need to store elements in the set. In the first call, it would be nums1 elements and in the second call, it would be nums2 elements. The space required to store the output list is not considered part of space complexity, and hence the total space complexity would be equal to O(max(N,M))O(max(N, M))*O*(*max*(*N*,*M*)).