java.util.Set

- Does not allow duplicates.
- add(e) ->
  Time Complexity : O(1)
  Worst Case : O(logn)
- remove(e) ->
  Time Complexity : O(1)
  Worst Case : O(logn)
- search(e) ->
  Time Complexity : O(1)
  Worst Case : O(logn)

java.util.Map

- Map has key & value pairs.
- In Hashing is purely based
  On Key.
- Does not allow duplicate keys,
  But allows duplicate values.
- If the key presents
  Value will be replaced.
- add(e) ->
  Time Complexity : O(1)
  Worst Case : O(logn)
- remove(e) ->
  Time Complexity : O(1)
  Worst Case : O(logn)
- search(e) ->
  Time Complexity : O(1)
  Worst Case : O(logn)

java.util.Set [Interface]

- HashSet
  Does not guarantee insertion order.
- Guarantees the insertion order.
  LinkedHashSet
- Sorts elements ascending order
  TreeSet

java.util.Map [Interface]

- HashMap
  Does not guarantee insertion order.
- LinkedHashMap
  Guarantees the insertion order.
- Sorts elements ascending order
  TreeMap

217. Contains Duplicate

Easy 4819 958 Add to List Share

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

Example 1:

```
Input: nums = [1,2,3,1]
Output: true
```

Example 2:

```
Input: nums = [1,2,3,4]
Output: false
```

Example 3:

```
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true
```

Constraints:

- $1 <= \text{nums.length} <= 10^5$
- $-10^9 <= \text{nums[i]} <= 10^9$

Has Duplicates should return true.

int[] arr = {1,2,3,1}

Set<Integer> set = new HashSet<>();

set[]

↓

On index:0 we have e:1 which is not presented in Set so add to the Set.

i:0 → set[1]

↓

On index:1 we have e:2 which is not presented in Set so add to the Set.

i:1 → set[1,2]

↓

On index:2 we have e:3 which is not presented in Set so add to the Set.

i:2 → set[1,2,3]

↓

On index:3 we have e:1 which is presented in Set so add to the Set.

i:3 → set[1,2,3]

Return True

Time Complexity : O(n)
Space Complexity : O(n)

Does not have Duplicates should return false.

int[] arr = {1,2,3}

Set<Integer> set = new HashSet<>();

set[]

↓

On index:0 we have e:1 which is not presented in Set so add to the Set.

i:0 → set[1]

↓

On index:1 we have e:2 which is not presented in Set so add to the Set.

i:1 → set[1,2]

↓

On index:2 we have e:3 which is not presented in Set so add to the Set.

i:2 → set[1,2,3]

↓

As we reached Out of the array just return : False

442. Find All Duplicates in an Array

Medium 6269 252 Add to List Share

Given an integer array `nums` of length `n` where all the integers of `nums` are in the range `[1, n]` and each integer appears **once** or **twice**, return *an array of all the integers that appears **twice***.

You must write an algorithm that runs in `O(n)` time and uses only constant extra space.

Example 1:

```
Input: nums = [4,3,2,7,8,2,3,1]
Output: [2,3]
```

Example 2:

```
Input: nums = [1,1,2]
Output: [1]
```

Example 3:

```
Input: nums = [1]
Output: []
```

Constraints:

- `n == nums.length`
- $1 <= n <= 10^5$
- `1 <= nums[i] <= n`
- Each element in `nums` appears **once** or **twice**.

```
int[] arr = {1,2,3,1,7,3,11}
```

Return all the duplicates : { 1, 3 }

Make use of List and Set Data Structures,
If the element is repeated add to List otherwise add to Set

```
List<Integer> list = new ArrayList<>();

Set<Integer> set = new HashSet<>();
```

Time Complexity : O(n)
Space Complexity : O(n)

1. Two Sum

Easy 32645 1034 Add to List Share

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

Constraints:

- $2 <= \texttt{nums.length} <= 10^4$
- $-10^9 <= \texttt{nums[i]} <= 10^9$
- $-10^9 <= \texttt{target} <= 10^9$
- **Only one valid answer exists.**

[2, 7, 11, 5]

0	1	2	3
2	7	11	5

Target Sum = 9

Return the two index's summation is equals to targetSum => nums[0] + nums[1] = 9 —> {0,1}

```
Map<Integer, Integer> map = new HashMap<>();
```

map[]

↓

On index:0 we have e:2, target-2 = 9 - 2 = 7 , check if 7 is presented in the map , No —> so add key : e:2, value index:0

i:0 → Map[2->0]

↓

On index:1 we have e:7, target-7 = 9 - 7 = 2 , check if 2 is presented in the map

i:1 →

↓

Yes

return map.get(2) & map.get(7) I.e = {0,1}

Time Complexity : O(n)
Space Complexity : O(n)