

Ahmed M. Al-Shaikh Khalil
120220263
Lab06: Hashing and Maps
Exercise01

The screenshot shows the LeetCode interface for the 'Two Sum' problem. The submission is 'Accepted' with 63/63 testcases passed. Performance metrics are 2 ms runtime, 98.95% beats, and 45.38 MB memory. A line graph shows the runtime distribution. The code editor on the right contains a Java solution using a HashMap.

```
1 class Solution {
2     public int[] twoSum(int[] nums, int target) {
3
4         Map<Integer, Integer> numMap = new HashMap<>();
5
6         for (int i = 0; i < nums.length; i++) {
7             int complement = target - nums[i];
8             if (numMap.containsKey(complement)) {
9                 return new int[] { numMap.get(complement), i };
10            }
11            numMap.put(nums[i], i);
12        }
13        throw new IllegalArgumentException("No two sum solution");
14    }
15 }
```

```
class Solution {
    public int[] twoSum(int[] nums, int target) {

        Map<Integer, Integer> numMap = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (numMap.containsKey(complement)) {
                return new int[] { numMap.get(complement), i };
            }
            numMap.put(nums[i], i);
        }

        throw new IllegalArgumentException("No two sum solution");
    }
}
```

Ahmed M. Al-Shaikh Khalil
120220263
Lab06: Hashing and Maps
Exercise02

The screenshot displays the LeetCode interface for the 'Contains Duplicate' problem. On the left, the 'Runtime' section shows a bar chart indicating the solution's performance, with a runtime of 14 ms and 62.33% of submissions beaten. The 'Code' section on the right shows the Java solution. The solution uses a HashMap to store the elements of the array and checks for duplicates by seeing if an element already exists in the map.

```
class Solution {  
    public boolean containsDuplicate(int[] nums) {  
        Map<Integer, Boolean> map = new HashMap<>();  
        for (int num : nums) {  
            if (map.containsKey(num)) {  
                return true;  
            }  
            map.put(num, true);  
        }  
        return false;  
    }  
}
```

```
class Solution {  
    public boolean containsDuplicate(int[] nums) {  
        Map<Integer, Boolean> map = new HashMap<>();  
        for (int num : nums) {  
            if (map.containsKey(num)) {  
                return true;  
            }  
            map.put(num, true);  
        }  
        return false;  
    }  
}
```

Ahmed M. Al-Shaikh Khalil
120220263
Lab06: Hashing and Maps
Exercise03

The screenshot shows a LeetCode submission for the problem "Intersection of Two Arrays". The submission is accepted, with a runtime of 2 ms and a memory usage of 43.50 MB. The code is written in Java and uses HashSet to find the intersection of two arrays.

```
class Solution {  
    public int[] intersection(int[] nums1, int[] nums2) {  
        Set<Integer> set1 = new HashSet<>();  
        Set<Integer> resultSet = new HashSet<>();  
  
        for (int i = 0; i < nums1.length; i++) {  
            set1.add(nums1[i]);  
        }  
  
        for (int i = 0; i < nums2.length; i++) {  
            if (set1.contains(nums2[i])) {  
                resultSet.add(nums2[i]);  
            }  
        }  
  
        int[] result = new int[resultSet.size()];  
        int index = 0;  
        for (Integer num : resultSet) {  
            result[index++] = num;  
        }  
  
        return result;  
    }  
}
```

```
class Solution {  
    public int[] intersection(int[] nums1, int[] nums2) {  
        Set<Integer> set1 = new HashSet<>();  
        Set<Integer> resultSet = new HashSet<>();  
  
        for (int i = 0; i < nums1.length; i++) {  
            set1.add(nums1[i]);  
        }  
  
        for (int i = 0; i < nums2.length; i++) {  
            if (set1.contains(nums2[i])) {  
                resultSet.add(nums2[i]);  
            }  
        }  
  
        int[] result = new int[resultSet.size()];  
        int index = 0;  
        for (Integer num : resultSet) {  
            result[index++] = num;  
        }  
  
        return result;  
    }  
}
```