Two Sum

Problem:

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.**

Solution:

0	1	2	3	4
2	1	3	5	8

Target = 9

brute - force (using two for loops finding all possible pair):

2 + 1 = 3 **1 + 3 = 4** **3 + 5 = 8** **5 + 8 = 13**

2 + 3 = 5 **1 + 5 = 6** **3 + 8 = 11**

2 + 5 = 7 **1 + 8 = 9 !!** **3 + 5 = 8**

2 + 8 = 10

```
for(int i = 0 ; i < nums.length; i ++ ){                    —----------> O(n)
        for (int j = i + 1 ; j < nums.length; j++){         —----------> O(n)
                If (nums[i] + nums[j] === target){          —----------> O(1)
                        return new int [] {i,j};            —----------> O(1)
                }
        return  new int[] {};                               —----------> O(1)
        }                                                   ____________________
}                                                           Time Complexity : O(n*n+1+1+1)
```

: O(n^2)

Optimized Solution (Using HashMap):

0	1	2	3	4
2	1	3	5	8

Target = 9

Diff = 2 - 9 = 7

= 1 - 9 = 8

= 3 - 9 = 6

= 5 - 9 = 4

= 8 - 9 = 1 —------- Exist in Hashmap return index

2	0
1	1
3	2
5	3
8	4

HashMap

```
Int ans[2] = {};

Hashmap <Integer,Integer> map = new Hashmap<>;          —----------> O(n) Space

for(int i = 0; i<nums.length; i++){                     —---------> O(n) Time

        Int diff = nums[i] - target;                    —---------->O(1) Time

        if(map.containsKey(diff)){                      —---------->O(1) Time

            ans[0] = i;                                 —---------->O(1) Time

            ans[1] = map.get(diff);                     —---------->O(1) Time

            break;                                      —---------->O(1) Time

        }else{

            map.put(nums[i],i);                         —---------->O(1) Time

          }

        }

        return ans;                                     —---------->O(1) Time

}                                                       ————————————————————

                                                        Time Complexity = O(n+7)

                                                                        = O(n)

                                                        Space Complexity = O(n)
```