simple example

nums: 3, 2, 4 , target = 6

6 → (2, 4) → (1, 2)

2 + 4 = 6

indexes

x + y = 6

nums[i]

looping over nums

i = 0, nums[0] = 3, y = 6 − 3 = 3 does it exists in nums (another index?) NO

i = 1, nums[1] = 2, y = 6 − 2 = 4 does it ..... YES

indexes 1, 2 (2 + 4 = 6)

we do the search with map (O(1))

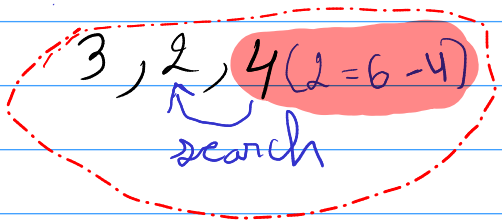
```
class Solution {
public:
    std::vector<int> twoSum(std::vector<int> &nums, int target) {
        map<int, int> numsIndexes;
        for (int i = 0; i < nums.size(); i++) {
            numsIndexes[nums[i]] = i;
        }
        for (int i = 0; i < nums.size(); i++) {
            int calculatedTarget = target - nums[i];
            if (numsIndexes.contains(calculatedTarget)) {
                int j = numsIndexes[calculatedTarget];
                if (i == j) {
                    continue;
                }
                return {i, j};
            }
        }
        return {};
    }
};
```

How to avoid the loop for storing in map?

3, 2, 4 — 2 + 4 = 6 (break condition in last example)

can also be 4 + 2 = 6

means that we can search for 2 when we attempt the 4 instead of searching for 4 at 2 (i+1) future



index 0: 3 → 6 - 3 = 3 not in map empty
store 3 in map

index 1: 2 → 6 - 2 = 4 not in map
store 2 in map, we have (3, 2)

index 2: 4 → 6 - 4 = 2 → exists in map
so 4 + 2 = 6 (1, 2)

```
class Solution {
public:
    std::vector<int> twoSum(std::vector<int> &nums, int target) {
        map<int, int> numsIndexes;
        for (int i = 0; i < nums.size(); i++) {
            int calculatedTarget = target - nums[i];
            if (numsIndexes.contains(calculatedTarget)) {
                int j = numsIndexes[calculatedTarget];
                if (i == j) {
                    continue;
                }
                return {j, i};
            } else {
                numsIndexes[nums[i]] = i;
            }
        }
        return {};
    }
};
```