Brute force, Hash Table, optimized hash table.

NotAllWhoWanderAreLost ★ 3 September 23, 2020 2:42 PM 188 VIEWS

3

Richard Feynman : *If you can't explain something in simple terms, you don't understand it*

The Brute Force method:

We go through the array, and for each element, we traverse the remaining array - for each pair, we check if the sum of these elements is equal to the target. If yes, we return it. If in the end, if there's no such pair, we return an empty vector.

Since there are two *for* loops embedded, the time complexity is O(n^2). **Runtime**: 344ms.

Other than our basic variables, we don't use any storage, and the space complexity is O(1). **Memory**: 9.3 MB.

```
vector<int> twoSum(vector<int>& nums, int target) {

    int n = nums.size();
    for(int start = 0; start < n-1; start++)
    {
        for(int end = start+1; end < n; end++)
        {
            if(nums[start] + nums[end] == target)
                return {start, end};
        }
    }
    return {};
}
```

Now, how can we do better than the brute force approach?

Notice that we're traversing most of the array for each element. If we could get rid of the second *for* loop in our code, the time complexity would come down to O(n). That's a big improvement, especially for large inputs.

What we're doing in the second loop is looking for an element which can sum with our current element to reach the target. Essentially, we're performing a look-up. That's where a hash-table is useful. It can perform lookups, generally, in O(1) time.

Hash-table approach:

```
vector<int> twoSum(vector<int>& nums, int target) {

    unordered_map<int, int> stored;

    for (int i = 0; i < nums.size() ; i++) {
        auto it = stored.find(target - nums[i]);

        if (it != stored.end()) //means we've found our pair.
            return vector<int> {i, it->second};

        stored[nums[i]] = i; //Keep inserting the elements we've already seen into the table.
}
    //If we find no pair which adds up to the target,
    return {};
}
```

Runtime: 16 ms, a big improvement over the 344 ms of the brute force method.

Memory: 10.3 MB - An increase from the brute force as we use the unordered_map here.

Now, how can we do better than this?

Same approach, but instead of using an iterator, we directly search for the element in the hash-table. This reduces both the memory and time needed to execute.

Runtime : 8ms, down from 16ms.
Memory : 10.2 MB.

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {

        unordered_map<int, int> hash;
        for (int i = 0; i < nums.size(); i++) {
            if (hash.find(target - nums[i]) != hash.end()) {
                return {hash[target - nums[i]], i};
            }
            hash[nums[i]] = i;
        }
        return {};
    }
};
```

If anyone solved it faster than 8ms, lemme know the approach! I've already tried sorting+binary search, but that runs in O(nlogn) time.