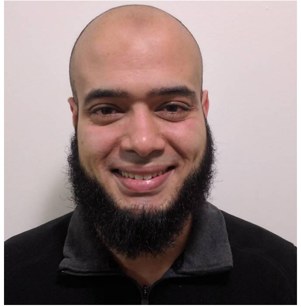
Cracking Coding Interviews

4 sums

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

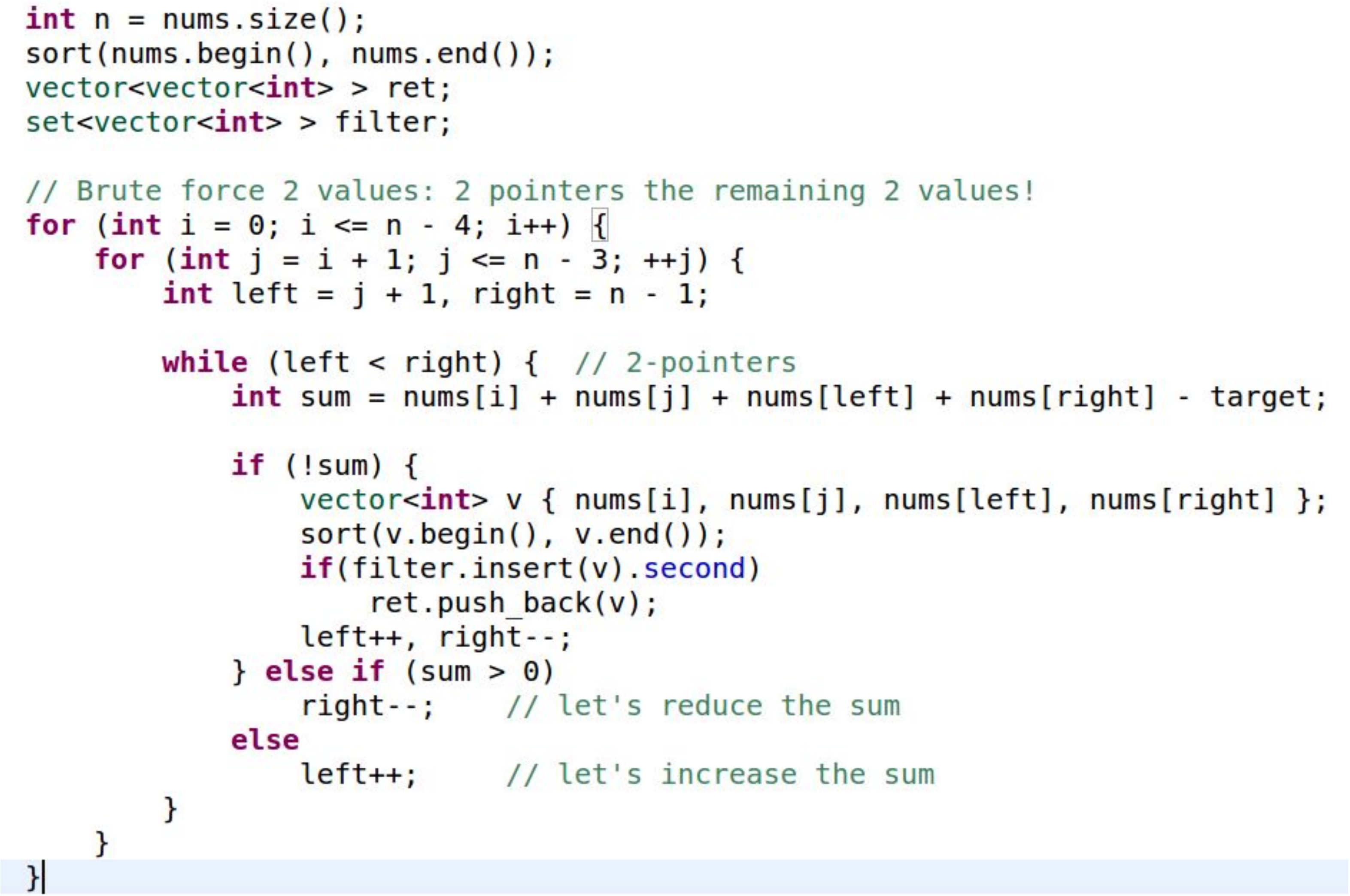


Leetcode 18 - 4Sum

- **Please solve 3Sum first**
- Let’s extend 3Sum to the following
  - Find 4 numbers instead of 3
  - Their sum = target (previously zero in 3sum) - trivial change
- Input ⇒ Output
  - [1,0,-1,0,-2,2], target = 0 ⇒ [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]
  - [2,2,2,2,2], target = 8 ⇒ [[2,2,2,2]]
- Signature
  - C++: vector<vector<int>> fourSum(vector<int>& nums, int target)
- How can we solve this problem again using 2-pointers style?

- Hint: Brute-force 2 values and 2-pointers on the remaining 2 values
  - Then order is O(n^3) instead of O(n^4)

4Sum



```
int n = nums.size();
sort(nums.begin(), nums.end());
vector<vector<int> > ret;
set<vector<int> > filter;

// Brute force 2 values: 2 pointers the remaining 2 values!
for (int i = 0; i <= n - 4; i++) {
    for (int j = i + 1; j <= n - 3; ++j) {
        int left = j + 1, right = n - 1;

        while (left < right) {  // 2-pointers
            int sum = nums[i] + nums[j] + nums[left] + nums[right] - target;

            if (!sum) {
                vector<int> v { nums[i], nums[j], nums[left], nums[right] };
                sort(v.begin(), v.end());
                if(filter.insert(v).second)
                    ret.push_back(v);
                left++, right--;
            } else if (sum > 0)
                right--;    // let's reduce the sum
            else
                left++;     // let's increase the sum
        }
    }
}
```

Handling kSum

- The previous idea should be direct. See code sample here
- The point to see: there is a pattern to use 2-pointers for the general problem
- For k-sum, use nested k-2 loops for k-2 variables
- For the remaining 2 variables, we use 2-pointers
- In other words, instead of O(n^k) brute-force, we use 2-pointers to remove a single loop and have $O(n^{k-1})$ order!
- But how to code k-2 nested loops if the k is parameter?
    - Using recursion: k-2 times recursive calls each is generating a loop and finally 2-pointers
    - For code sample: see leetcode editorial

4Sum using hash table

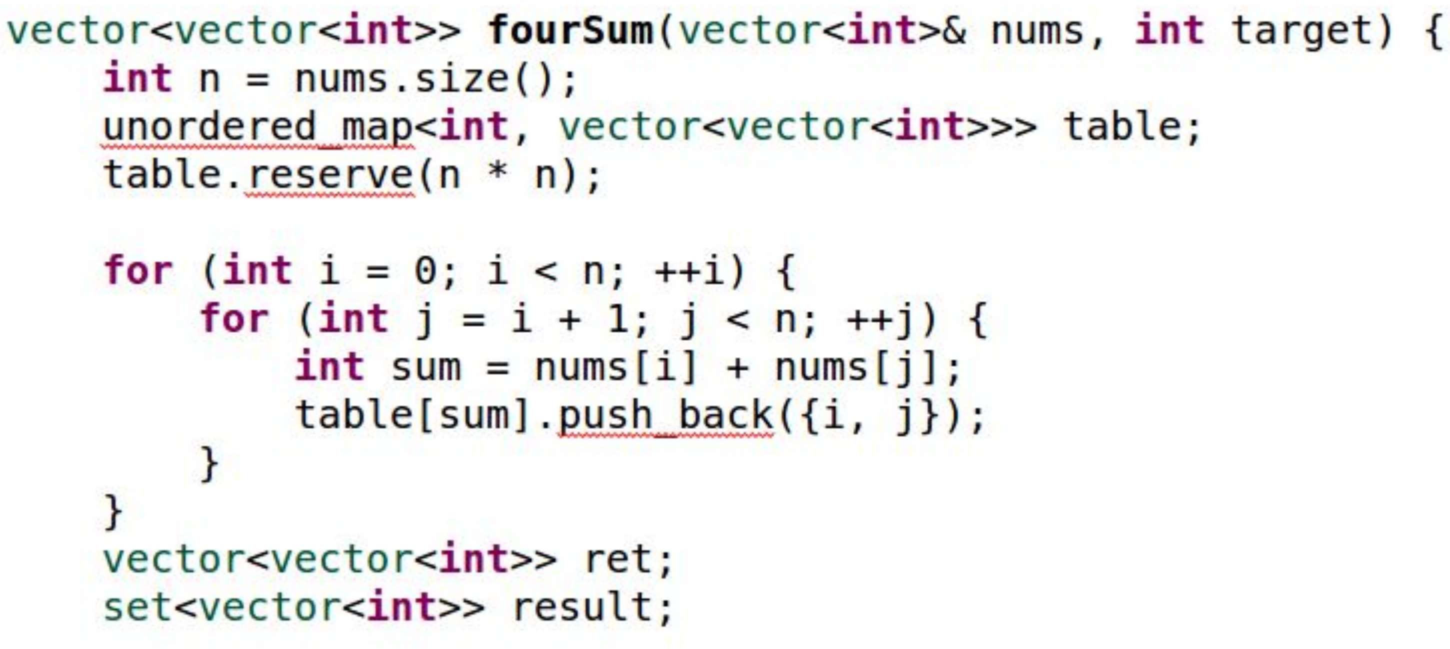
- We managed 2-pointers for 3sum, 4sum and even general ksum
- We knew how to use hashing for 2sum
- We can extend this for also kSum
- Think about hashing for 4sum
- Hint: create a hash table for the sum of every 2 values

4Sum using hash table

- The idea is to split the values to 2 parts: e.g. 2 values and 2 values
- We store the sum of any pair of values in the hash table
- Then loop with 2 nested loops to look up for the remaining sum
- E.g. Assume arr is: {1, 2, -3, 5}
    - We create a hash table for the sum of every pair sum
        - (1, 2), (1, -3), (1, 5), (2, -3), (2, 5), (3, -5)
        - For every pair: we get the sum, then we use table[sum]
        - Table[sum] will be an array of the possible pairs for the indices
    - With the other 2 nested loops, we search for the remaining pair
    - Let’s the target = 5
    - Let the current pair is (1, 2) which is 3. We need to search the table for 5-3 = 2
    - Table[2] will have indices of the values (-3, 5).

4Sum using hash table

- Our table maps from int (the pair sum) to vector of the pairs (indices)

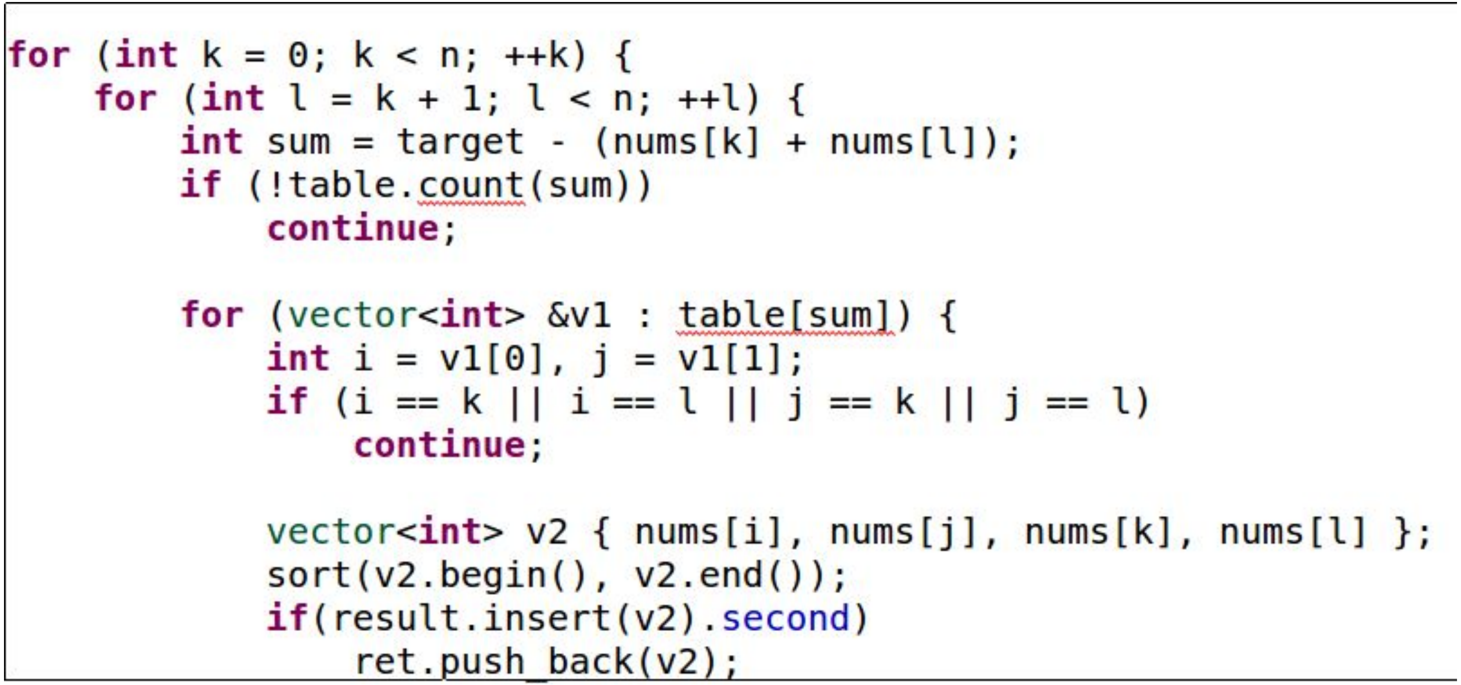


```
vector<vector<int>> fourSum(vector<int>& nums, int target) {
    int n = nums.size();
    unordered_map<int, vector<vector<int>>> table;
    table.reserve(n * n);

    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            int sum = nums[i] + nums[j];
            table[sum].push_back({i, j});
        }
    }
    vector<vector<int>> ret;
    set<vector<int>> result;
```

4Sum using hash table

- The table has a pair of values. Let’s generate the other pair of values. Use their sum to decide the remaining sum and find it from the table



```
for (int k = 0; k < n; ++k) {
    for (int l = k + 1; l < n; ++l) {
        int sum = target - (nums[k] + nums[l]);
        if (!table.count(sum))
            continue;

        for (vector<int> &v1 : table[sum]) {
            int i = v1[0], j = v1[1];
            if (i == k || i == l || j == k || j == l)
                continue;

            vector<int> v2 { nums[i], nums[j], nums[k], nums[l] };
            sort(v2.begin(), v2.end());
            if(result.insert(v2).second)
                ret.push_back(v2);
```

4Sum using hashtable

- Our code is doing O(n^2), then another O(n^2), but there is also an internal list of items with the requested sum (let’s call them S)
  - For large n, this code still could be slow, as we are are creating too many vectors
- As you see, with hashing, we actually get $O(n^{k/2}+S)$. With a large k, this approach will be way faster than 2-pointers, which only remove a single loop!
  - For k =8: 4 loops to add all 4 items in the table. Another 4 loops to find the remaining ones
- “Meet in middle” technique
  - What we did has this name. It is actually a repetitive trick
  - Divide your processing to 2 parts
  - Store half of them (in our case every pair results)
  - Search for the other half. This helps us reduce the processing a lot to $O(n^{k/2})$

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."