Two pointer and Binary Search
Professor Wladimir

Two pointer

Solução Trivial: $\mathcal{O}(n^2)$
Solução two pointer com vetor ordenado: $\mathcal{O}(n)$
Solução two pointer com vetor não ordenado: $\mathcal{O}(n \log n)$

Two Integer Sum II

Given an array of integers numbers that is sorted in non-decreasing order.

Return the indices (1-indexed) of two numbers, [index1, index2], such that they add up to a given target number target and index1 ≠ index2. Note that index1 and index2 cannot be equal, therefore you may not use the same element twice.

There will always be exactly one valid solution.

Your solution must use $\mathcal{O}(1)$ additional space.

```
vector<int> twoSum(vector<int>& numbers, int target) {
        vector <int> res;
        int i = 0;
        int j = numbers.size() - 1;

        while(i < j){
            int A = numbers[i];
            int B = numbers[j];

            if(A+B == target){
                res.push_back(i+1);
                res.push_back(j+1);
                return res;
            }else if(A+B < target){
                i++;
            }else {
                j--;
            }
        }
    }
```

3SUM

Solução Trivial: $\mathcal{O}(n^3)$

Solução two pointer : $\mathcal{O}(n^2)$

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] where nums[i] + nums[j] + nums[k] == 0, and the indices i, j and k are all distinct.

The output should not contain any duplicate triplets. You may return the output and the triplets in any order.

Example 1:

Input: nums = [-1,0,1,2,-1,-4]

Output: [[-1,-1,2],[-1,0,1]]

Explanation:

nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.

nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.

nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.

The distinct triplets are [-1,0,1] and [-1,-1,2].

```cpp
vector<vector<int>> threeSum(vector<int>& nums) {

    vector< vector<int> > triplet;

    set < vector<int> > s;

    sort(nums.begin(), nums.end() );


    for(int i = 0; i < nums.size()-2; i++){
        if(i > 0 && nums[i] == nums[i-1]) continue;
        if( nums[i] > 0) break;
        int j = i+1;
        int k = nums.size() - 1;
        int target = -nums[i];
        cout << "target " << target << endl;
        while(j < k){
            int soma = nums[j] + nums[k];
            if(soma == target){
                //triplet.push_back({nums[i], nums[j], nums[k]});
                s.insert({nums[i], nums[j], nums[k]});
                j++;
                k--;
            }else if(soma > target){
                k--;
            }else{
                j++;
            }


        }
    }
    for(auto triple : s){
        triplet.push_back(triple);
    }
    return triplet;
}
```

```cpp
vector<vector<int>> threeSum(vector<int>& nums) {
    vector<vector<int>> res;
    sort(nums.begin(), nums.end());

    for(int i = 0; i < nums.size() - 2; i++){
        //pula os elementos repetidos já processados
        if(i > 0 && nums[i] == nums[i-1]) continue;

        int target = -nums[i];
        int start = i + 1;
        int end = nums.size() - 1;

        while(start < end){
            int sum = nums[start] + nums[end];
            if(sum == target){
                res.push_back({nums[i], nums[start], nums[end]});
                //pula elementos repetidos
                while(start < end && nums[start] == nums[start + 1]) start++;
                //pula elementos repetidos
                while(start < end && nums[end] == nums[end - 1]) end--;
                start++;
                end--;
            } else if(sum < target){
                start++;
            } else {
                end--;
            }
        }
    }

    return res;
}
```

Binary Search

Search a 2D Matrix

You are given an m x n 2-D integer array matrix and an integer target.

- Each row in matrix is sorted in non-decreasing order.
- The first integer of every row is greater than the last integer of the previous row.

Return true if target exists within matrix or false otherwise.
Can you write a solution that runs in O(log(m * n)) time?

```cpp
bool searchMatrix(vector<vector<int>>& matrix, int target) {


        int start = 0;
        int end = matrix.size() - 1;
        int sol = -1;
        //Encontrando o maior indice da linha em que o primeiro elemento é menor ou
        ↪  igual ao target
        while( start <= end ){
            int mid = (start +end)/2;

            if( matrix[mid][0] <= target ){
                sol = mid;
                start = mid+1;
            }else{
                end = mid-1;
            }

        }
        if(sol == -1) return false;

        //cout << "linha " << sol << endl;

        start = 0;
        end   = matrix[sol].size() - 1;

        while( start <= end ){
            int mid = (start + end)/2;

            if( matrix[sol][mid] == target ){
                return true;
            }else if( matrix[sol][mid] > target ){
                end = mid-1;
            }else {
                start = mid+1;
            }
        }

        return false;

    }
```

```cpp
bool searchMatrix(vector<vector<int>>& matrix, int target) {

    int n = matrix.size(); // rows
    int m = matrix[0].size(); //cols

    int start = 0;
    int end = n*m - 1;

    while( start <= end){
        int r, c,  mid;
        mid = (start + end)/2;

        r = mid/m;
        c = mid%m;

        if( matrix[r][c] == target )
            return true;
        else if( matrix[r][c] > target){
            end = mid - 1;
        }else{
            start = mid + 1;
        }

    }
    return false;
}
```