# Contents

# Part I

# 1. Two Sum

# Chapter 1

# Problem Statement

Link

# Chapter 2

# Analysis

# Chapter 3

# Implementation

## 3.1 C++

### 3.1.1 $O(N^2)$ (faster than 35.43%)

Idea: traverse the vector. For each encountered value, calculate the corresponding value it needs to add up to the target value. And then traverse the vector to look for this value.

The time complexity is $O(N^2)$, because for each value in the vector, you'll go through the vector and search its corresponding part so they add up to the target. This is linear searching, which has $O(N)$ complexity.

```
class Solution {
public:
  vector<int> twoSum(vector<int>& nums, int target) {
    for (auto i = nums.begin(); i != nums.end(); ++i) {
      int other_part = target - (*i);
      auto itr = find(nums.begin(), nums.end(), other_part);

      if (itr != nums.end() && itr != i)
        return {static_cast<int>(i - nums.begin()), static_cast<int>(itr -
        ↪  nums.begin())};
    }

    return {0, 1};
  }
};
```

### 3.1.2 $O(N \log N)$ (faster than 99.24%)

Idea: the searching part is optimized. First we sort the vector. In order to keep the original relative order of each element, we sort a vector of iterators that referring each element in the

original vector **nums**. Then, we can use this sorted vector to perform binary search, whose time complexity is $\log N$. The total time complexity is reduced to $O(N \log N)$.

I made some bugs when writting this code, because I didn't realize the following assumption:

- duplicates allowed
- each input would have ***exactly*** one solution

Code:

```
class Solution {
public:
  /*Notes:
    The compare object used to sort vector of iterators
  */
  struct Compare {
    bool operator()(vector<int>::iterator a, vector<int>::iterator b) {
      return (*a < *b);
    }

  };

  /*Notes:
    A binary search to find target value in a vector of iterators;
    if found: return the index value of that iterator
    if not found: return -1
  */
  int findTarget(int target, const vector<vector<int>::iterator>&
  ↪  itr_vector, const vector<int>::iterator& current_itr) {
    int start_index = 0;
    int end_index = itr_vector.size() - 1;
    int middle;
    int result = -1;

    while (start_index <= end_index) {
      // update middle
      middle = (start_index + end_index) / 2;
      // check value
      if (*itr_vector[middle] == target) {
        if (itr_vector[middle] == current_itr) {
          start_index += 1;
          end_index += 1;
          continue;
        }

        result = middle;
```

```
36          break;
37        }
38
39        else if (*itr_vector[middle] > target) {
40          end_index = middle - 1;
41          continue;
42        }
43
44        else if (*itr_vector[middle] < target) {
45          start_index = middle + 1;
46          continue;
47        }
48
49      }
50
51      return result;
52    }
53
54
55    vector<int> twoSum(vector<int>& nums, int target) {
56      // create a vector of iterators
57      vector<vector<int>::iterator> itr_vector;
58      for (auto i = nums.begin(); i != nums.end(); ++i)
59        itr_vector.push_back(i);
60
61      // sort the vector of iterators, so the values these iterators referred
        ↪  to
62      // are in ascending order
63      sort(itr_vector.begin(), itr_vector.end(), Compare());
64
65      // go over nums, and find the pair
66      for (auto i = nums.begin(); i != nums.end() - 1; ++i) {
67        int other_part = target - (*i);
68        int other_part_index = findTarget(other_part, itr_vector, i);
69
70        if (other_part_index != -1) // found
71          return {static_cast<int>(i - nums.begin()),
            ↪  static_cast<int>(itr_vector[other_part_index] - nums.begin())};
72      }
73
74      // for syntax
75      return {0, 1};
76
77    }
78  };
```

# Part II

# 771. Jewels and Stones

# Chapter 4

# Problem Statement

Link

# Chapter 5

# Analysis

# Chapter 6

# Implementation

## 6.1 C++

### 6.1.1 $N^2$ solution

```cpp
class Solution {
public:
  int numJewelsInStones(string J, string S) {
    int numJewl = 0;
    for (auto s : S)
      if (isJewels(s, J))
        numJewl++;
    return numJewl;
  }

  bool isJewels(char s, string J) {
    for (auto j : J)
      if (s == j)
        return true;

    return false;
  }
};
```