**Indicies**

**Bahadur Khan**

Concordia University

January 14, 2022

CSC- 543

**Self-Assessment**

**1**. It took me just more than a week to complete the assignment.

**2**. I indulged my entire effort into coding the programs. In this assignment, I will expect an A Grade.

**3**. Every coding solution is accurate. So, I would expect an A grade.

**4**. Learning the fundamentals of C++ produced coding simple. The main issue I encountered was in running the code. As I complete all of the weekly assignments, I am becoming more precise in detecting the errors and executing the program. The overall experience was excellent.

**Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target**

class Solution {

public:

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

vector<int> ret;

int size = nums.size();

int i,j;

\* for(i=0;i<size;i++) {

for(j=i+1;j<size;j++) {

if(nums[i]+nums[j] == target) {

ret.push\_back(i);

ret.push\_back(j);

return ret;

}

}

}

return ret;

}

};

Time Complexity: O(n^2)

Space Complexity: O(1)

**Improvements:**

I used brute force to overcome the challenge. I did an initial iteration of the array's loop, then another. I finally opted to save the indexes in a vector and made the decision to restore the vector if I discovered two potential values that increased the target. Additionally, after sorting the data using a sorting algorithm, we could simply utilize two hints and tips to locate the identical pair of values which add up to the desired result. The selection sort would cause this to have an O time complexity (nlogn). We wouldn't have needed any extra space, thus this same case complexity could have been O (1).

Time Complexity: O(n)

Space Complexity: O(n)

\*/

class Solution {

public:

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

vector<int> ret;

int size = nums.size();

int i,j;

for(i=0;i<size;i++) {

for(j=i+1;j<size;j++) {

if(nums[i]+nums[j] == target) {

ret.push\_back(i);

ret.push\_back(j);

return ret;

}

}

}

return ret;

}

};

The proposed approach is a brute force algorithm that repeatedly loops through the array to identify two value systems that add to the objective together. It has an O time complexity (n2). We can make it better by storing the value systems inside a hashmap and then checking the hashmap to see if the conjunction appears. This would have an O(n) time complexity and an O(n) space complexity (n).

We iterate through the array in the initial loop, storing its value systems in a hashmap. The second loop checks to see if the complement is present in the hashmap. The indices are reintroduced if this is the situation. If not, the cycle will keep going. Due to the fact that we only loop and through the array once, this same running time is O. (n). Because we store the identical mindsets in a hashmap, the space complexity is O. (n).

The array might also be sorted using a sorting algorithm, and the identical pair of entries that add up to the goal could then be found by using two pointers. The sorting algorithm would cause this to have an O time complexity (nlogn). The space complexity could have been O because we wouldn't have needed any extra room. (1).

A binary search strategy may be used to discover the same complements of each value in the array. Due to the binary search algorithm, this would have an O time complexity (nlogn). Because we didn't require additional space, the complexity would've been O. (1).

**Github repository link:**

https://github.com/Bahadurk/12lang.git