**Indicies**

**Hameeduddin Imad Mohammed**

Concordia University

November 6, 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)

**Modifications:**

I solved the problem by using brute force. I first looped thru the array, then looped it again. I ended up saving the indices into the a vector as well as decided to return the vector if I found two possible values that added to the target.

We could also use a sorting algorithm to sort the myriad, then just use two hints and tips to find the same pair of values that add up to the goal. This would have a time complexity of O due to the selection sort (nlogn). Because we would not have required any additional space, this same space complexity would 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 solution presented above is a brute force algorithm that loops through the array multiple times to recognize a pair of value systems which add up to the objective. The time complexity is O(n2). We can improve it by putting away the value systems in a hashmap and afterwards examining the hashmap to see whether the conjunction occurs. This would have a time complexity of O(n) and a space complexity of O(n) (n).

In the first loop, we loop through the array and store the value systems inside a hashmap. The second loop determines whether the complement exists in the hashmap. If this is the case, the indices are brought back. Otherwise, we'll continue to loop. This same time complexity is O because we only loop through the array once (n). The space complexity is O because we store the same value systems in a hashmap (n).

We could also use a sorting algorithm to sort the array, then just use two pointers to discover the same pair of values that add up to the target. This would have a time complexity of O due to the sorting algorithm (nlogn). Because we would not have required any additional space, the space complexity might have been O. (1).

To find the same complement of each value in the array, a binary search algorithm might be used. This would have a time complexity of O due to binary search algorithm (nlogn). The space complexity would've been O because we didn't need any extra space (1).