**Problem Name:** Two sum

**Topics:**

**Companies:**

**Level:** Easy

**Language:** C++

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

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

**Input Format:**

First line of the input contain integer n (size of list)

Second line contain n space separated integer list values.

Last line contain integer value pos representing value of node to delete.

Ex:

5

1 2 3 4 5

1

**Output Format:** Print linked list after removing node having value pos

**Constraints:**

* The number of the nodes in the given list is in the range [2, 1000].
* -1000 <= Node.val <= 1000
* The value of each node in the list is **unique**.
* The node to be deleted is **in the list** and is **not a tail** node

**Examples:**

**Input:** head = [4,5,1,9], node = 5

**Output:** [4,1,9]

**Explanation:** You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.

**Brute force Solution:**

**Explanation:**

1. Brute Force : Using Two Loops

Use two loops and check A[i] + A[j] == K for each pair (i, j) in A[].  
If there exists a pair with sum equals to K then return true. By end of both loops,  
If you didn’t find such a pair then return false.

**Code:**

**Time Complexity**: O(N2) The total no. of comparison in worst case = Total no. of possible pairs = nC2 = n(n-1)/2 = O(n²)

**Space Complexity:** O(1)

**Optimized Solution:**

**Explanation:**

The idea here, to make the code more efficient than a nested loop (which takes O(n^2) time), is to use a hash table - to map every value with its index as a <key, value> pair (i.e. <number, index> pair) until we reach a number which is the difference between the target and the number being currently processed (theoretically it is until we reach either that or the end of the array, but we are told we can assume that the array contains one solution). To that end, we check at every iteration whether the value equalling the target minus the current number has already been saved. This code uses an unordered\_map, since it is stored as a hash table and would be most efficient for this process.

**Code:**

**Time Complexity**: O(n) In the best-case scenario, the 2 numbers would be the first 2 values of the array, and we would get the results very fast, in O(1) time. However, in the average and worst-case scenarios, we would loop through a large part of the array, or even all of it, before we find the pair of values, resulting in a running time of **O(n)**.

**Space Complexity:** O(n) This solution saves key and value pairs for each item in the array until we find the required indices, if they exist. The hash map uses O(n) space.