Find a triplet with the given sum in an array

Given an unsorted integer array, find a triplet with a given sum in it.

For example,

```
Input:

nums = [ 2, 7, 4, 0, 9, 5, 1, 3 ]
target = 6

Output: Triplet exists.

The triplets with the given sum 6 are {0, 1, 5}, {0, 2, 4}, {1, 2, 3}
```

This website uses cookies. By using this site you agree to the use of cookies, our policies, copyright terms and other conditions. Read our Privacy Policy. Accept and Close

Practice this problem

The problem is a standard variation of the 3SUM problem, where instead of looking for numbers whose sum is 0, we look for numbers whose sum is any constant `C`.

1. Naive Recursive Approach

The idea is similar to the 0–1 Knapsack problem and uses recursion. We either consider the current item or exclude it and recur for the remaining elements for each item. Return true if we get the desired sum by including or excluding the current item. Following is the implementation in C++, Java, and Python based on the idea:

C++ Java Python

```
#include <iostream>
using namespace std;

// Naive recursive function to check if triplet exists in an array
// with the given sum
bool isTripletExist(int nums[], int n, int target, int count)
{
    // if triplet has the desired sum, return true
    if (count == 3 && target == 0) {
        return true;
    }

    // return false if the sum is not possible with the current configuration
    if (count == 3 || n == 0 || target < 0) {
```

This website uses cookies. By using this site you agree to the use of cookies, our policies, copyright terms and other conditions. Read our Privacy Policy.

Accept and Close

```

    // recur with including and excluding the current element
    return isTripletExist(nums, n - 1, target - nums[n - 1], count + 1) ||
            isTripletExist(nums, n - 1, target, count);
}

int main()
{
    int nums[] = { 2, 7, 4, 0, 9, 5, 1, 3 };
    int target = 6;

    int n = sizeof(nums) / sizeof(nums[0]);

    isTripletExist(nums, n, target, 0) ? cout << "Triplet exists":
                                         cout << "Triplet doesn't exist";

    return 0;
}
```

Download Run Code

Output:

```
Triplet exists
```

We can also use three nested loops and consider every triplet in the given array to check if the desired sum is found.

2. Using Hashing

This website uses cookies. By using this site you agree to the use of cookies, our policies, copyright terms and other conditions. Read our Privacy Policy.

Accept and Close

The idea is to insert each array element into a hash table. Then consider all pairs present in the array and check if the remaining sum exists in the map or not. If the remaining sum is seen before and triplet doesn't overlap with each other, i.e., `(i, j, i)` or `(i, j, j)`, print the triplet and return. The algorithm can be implemented as follows in C++, Java, and Python:

C++ Java Python

```cpp
#include <iostream>
#include <algorithm>
#include <unordered_map>
using namespace std;

// Function to check if triplet exists in an array with the given sum
bool isTripletExist(int nums[], int n, int target)
{
    // create an empty map
    unordered_map<int, int> map;

    // insert (element, index) pair into the map for each array element
    for (int i = 0; i < n; i++) {
        map[nums[i]] = i;
    }

    // consider each element except the last element
    for (int i = 0; i < n - 1; i++)
    {
        // start from the i'th element until the last element
        for (int j = i + 1; j < n; j++)
        {
            // remaining sum
            int val = target - (nums[i] + nums[j]);

            // if the remaining sum is found, we have found a triplet
```

This website uses cookies. By using this site you agree to the use of cookies, our policies, copyright terms and other conditions. Read our Privacy Policy.

Accept and Close

```
                if (map[val] != i && map[val] != j) {
                    return true;
                }
            }
        }
    }

    // return false if triplet doesn't exist
    return false;
}

int main()
{
    int nums[] = { 2, 7, 4, 0, 9, 5, 1, 3 };
    int target = 6;

    int n = sizeof(nums) / sizeof(nums[0]);

    isTripletExist(nums, n, target) ? cout << "Triplet exists":
                                    cout << "Triplet Don't Exist";

    return 0;
}
```

Download Run Code

Output:

```
Triplet exists
```

The time complexity of the above solution is `O(n²)` and requires `O(n)` extra space, where `n` is the size of the input.

This website uses cookies. By using this site you agree to the use of cookies, our policies, copyright terms and other conditions. Read our Privacy Policy.
Accept and Close

3. Printing distinct triplets

The idea is to sort the given array in ascending order, and for each element `nums[i]` in the array, check if the triplet is formed by nums[i] and a pair from subarray `nums[i+1…n)`. This is demonstrated below in C++, Java, and Python:

C++ Java Python

```
#include <iostream>
#include <algorithm>
using namespace std;

// Function to print all distinct triplet in an array with the given sum
void printAllTriplets(int nums[], int n, int target)
{
    // sort the array in ascending order
    sort(nums, nums + n);

    // check if triplet is formed by nums[i] and a pair from
    // subarray nums[i+1…n)
    for (int i = 0; i <= n - 3; i++)
    {
        // remaining sum
        int k = target - nums[i];

        // maintain two indices pointing to endpoints of the
        // subarray nums[i+1…n)
        int low = i + 1, high = n - 1;

        // loop if `low` is less than `high`
        while (low < high)
        {
            // increment `low` index if the total is less than the remaining sum
```

This website uses cookies. By using this site you agree to the use of cookies, our policies, copyright terms and other conditions. Read our Privacy Policy. Accept and Close

```
            }

            // decrement `high` index if the total is more than the remaining sum
            else if (nums[low] + nums[high] > k) {
                high--;
            }

            // triplet with the given sum is found
            else {
                // print the triplet
                cout << "(" << nums[i] << " " << nums[low] << " " <<
                        nums[high] << ")\n";

                // increment `low` index and decrement `high` index
                low++, high--;
            }
        }
    }
}

int main()
{
    int nums[] = { 2, 7, 4, 0, 9, 5, 1, 3 };
    int target = 6;

    int n = sizeof(nums) / sizeof(nums[0]);

    printAllTriplets(nums, n, target);

    return 0;
}
```

Download Run Code

Output:

```
(0 1 5)
```

This website uses cookies. By using this site you agree to the use of cookies, our policies, copyright terms and other conditions. Read our Privacy Policy.

Accept and Close

```
(0 2 4)
(1 2 3)
```

The time complexity of the above solution is `O(n²)` and doesn't require any extra space.

Array, Sorting

Hashing, Medium, Recursive

Techie Delight © 2022 All Rights Reserved. | Privacy Policy | Send feedback

This website uses cookies. By using this site you agree to the use of cookies, our policies, copyright terms and other conditions. Read our Privacy Policy. Accept and Close