Two Sum

BRUTE FORCE :

The naive idea is to check out all combination whetehr thery sum upto the target using two nested loops

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int n=nums.size();
        vector<int> ans;
        for(int i=0;i<n;i++)
        {
            int one=nums[i];
            int more_needed=target-nums[i];
            for(int j=i+1;j<n;j++)
            {
                if(nums[j]==more_needed)
                {
                    ans.push_back(i);
                    ans.push_back(j);
                    return ans;
                }
            }
        }
    return ans;
    }
};
```

- Time Complexity : O(N^2)
- Space Complexity : O(1)

Coding ninjas solution :

```
#include <bits/stdc++.h>

vector<vector<int>> pairSum(vector<int> &nums, int target){
   // Write your code here.
   int n=nums.size();
        vector<vector<int>> ans;
        for(int i=0;i<n;i++)
        {
            int one=nums[i];
```

```
            int more_needed=target-nums[i];
            for(int j=i+1;j<n;j++)
            {
                if(nums[j]==more_needed)
                {
                    vector<int> temp;
                    temp.push_back(one);
                    temp.push_back(more_needed);
                    sort(temp.begin(),temp.end());
                    ans.push_back(temp);
                }
            }
        }
    sort(ans.begin(),ans.end());
    return ans;
}
```

Optimal Approach 1 :

Store the elements in a map and search for target-nums[i] if it is found push into answer.

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int n=nums.size();
        vector<int> ans;
        unordered_map<int,int> m;
        for(int i=0;i<n;i++)
        {
            int needed=target-nums[i];
            if(m.find(needed)!=m.end())
            {
                ans.push_back(i);
                ans.push_back(m[target-nums[i]]);
                return ans;
            }
            m[nums[i]]=i;
        }
    return ans;
    }
};
```

- Time Complexity : O(NlogN)
- Space Complexity : O(N)

Optimal Approach 2 :

Sort the array and then use two pointer if sum is equal return ans else if sum is less then move left pointer otherwise move right pointer.

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int n=nums.size();
        vector<int> ans;
        sort(nums.begin(),nums.end());
        int i=0,j=n-1;
        while(i<j)
        {
            int sum=nums[i]+nums[j];
            if(sum==target)
            {
                ans.push_back(i);
                ans.push_back(j);
                return ans;
            }
            else if(sum<target)
            {
                i++;
            }
            else
            {
                j--;
            }
        }
    return ans;
    }
};
```

- Time Complexity : O(NlogN)
- Space Complexity : O(1)