Array Subset of another array

```
string isSubset(int a1[], int a2[], int n, int m) {
    sort(a1, a1 + n);
    sort(a2, a2 + m);
    int i = 0, j = 0;
    while (i < n && j < m ){
        if( a1[i] < a2[j] )
            i++;
        else if( a1[i] == a2[j] ){
            j++;
            i++;
        }
        else if( a1[i] > a2[j] )
            return "No";
    }
 if (j<m)
   return "No";
 else
   return "Yes";
}
```

Triplet Sum in Array

```
bool find3Numbers(int A[], int n, int X)
   {
       sort(A,A+n);
       for (int i = 0; i <n; i++){
           int j = i + 1;
           int k = n - 1;
           while (j < k) {
               int sum = A[i] + A[j] + A[k];
               if (sum == X) {
                   return true;
                   j++;
                   k--;
               } else if (sum < X) {
                   j++;
               } else {
                   k--;
               }
           }
       }
       return false;
```

```
}
```

Count pairs with given sum

```
int getPairsCount(int arr[], int n, int k){
    unordered_map<int, int> m;
    int count = 0;
    for (int i = 0; i < n; i++) {
        if (m.find(k - arr[i]) != m.end()) {
            count += m[k - arr[i]];
        }
        m[arr[i]]++;
    }
    return count;
    }
```

Next Permutation

```
void nextPermutation(vector<int>& nums) {
        int n = nums.size();
        int i = n - 1;
        while (i > 0 && nums[i-1] >= nums[i])
        i--;

        if (i == 0) {
        reverse(nums.begin(), nums.end());
        return;
        }
        int j = n - 1;
        while (nums[j] <= nums[i-1])
        j--;
        swap(nums[i-1], nums[j]);
        reverse(nums.begin() + i, nums.end());
    }
```