18. 4Sum

Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that:

- 0 <= a, b, c, d < n
- a, b, c, and d are distinct.
- nums[a] + nums[b] + nums[c] + nums[d] == target

You may return the answer in any order.

```
//Time Complexity: O(n^3)

class Solution {

public:

    vector<vector<int>> fourSum(vector<int>& nums, long long target) {

        sort(nums.begin(),nums.end());

        int n=nums.size();

        vector<vector<int>>ans;



        for(int i=0;i<n-3;i++)

        {

            if(i>0 && nums[i]==nums[i-1])

            {

                continue;

            }

            for(int j=i+1;j<n-2;j++)

            {

            if(j>i+1 && nums[j]==nums[j-1])

            {
```

```
                continue;
            }

                int low=j+1;
                int  high=n-1;
                long sum=target-(nums[i]+nums[j]);
                while(low<high)
                {
                    if(nums[low]+nums[high]==sum)
                    {
vector<int>ansOne={nums[i],nums[j],nums[low],nums[high]};
                        ans.push_back(ansOne);
                         while(low < high && nums[low] == nums[low + 1]){
                            low++;
                        }
                        while(low < high && nums[high] == nums[high - 1]){
                            high--;
                        }
                        low++;
                        high--;
                    }

                    else if(nums[low]+nums[high]>sum)
```

```
                    {
                        high--;
                    }
                    else
                    {
                       low++;
                    }
                }
            }
        }
        return ans;
    }
};
```

19. Single Number

Given a **non-empty** array of integers nums, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

```
//Time Complexity: O(N)

//Space Complexity:O(1)

class Solution {

public:

    int singleNumber(vector<int>& nums) {

    int ans=0;

    for(int i=0;i<nums.size();i++)

    {



     ans=ans^nums[i];

     cout<<ans;

    }



    return ans;

    }

};
```

20: Plus One

You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

```
//Time Complexity: O(N)

//Space Complexity:O(1)

class Solution {

public:

    vector<int> plusOne(vector<int>& digits) {

        for(int i=digits.size()-1; i>=0; i--)

        {

            if (digits[i]<9)

            {

                digits[i]++;

                return digits;

            }

            else

            {

                digits[i]=0;

            }

        }

        digits.insert(begin(digits), 1);

        return digits;

    }

};
```

21. Summary Ranges

You are given a sorted unique integer array nums.

A range [a,b] is the set of all integers from a to b (inclusive).

Return the smallest sorted list of ranges that cover all the numbers in the array exactly. That is, each element of nums is covered by exactly one of the ranges, and there is no integer x such that x is in one of the ranges but not in nums.

Each range [a,b] in the list should be output as:

- "a->b" if a != b
- "a" if a == b

```
//Time Complexity: O(n)

//Space Complexity: O(n)

class Solution {

public:

    vector<string> summaryRanges(vector<int>& nums) {

        int n=nums.size();

        vector<string>ans;

        if(n==0)

        {

            return ans;

        }

        for(int i=0;i<nums.size();)

        {

            int start=i,end=i;

            while(end+1<n && nums[end]+1==nums[end+1])
```

```
            {
                end++;
            }
            if(end>start)
            {

ans.push_back(to_string(nums[start])+"->"+(to_string(nums[end])));
            }
            else
            {
                ans.push_back(to_string(nums[start]));
            }
            i=end+1;
        }
        return ans;

    }
};
```

22. Next Permutation

A permutation of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].

```
//Time Complexity: O(n)

//Space Complexity: O(1)

class Solution {

public:

    void nextPermutation(vector<int>& nums) {

        int n=nums.size();

        if(nums.size()==1)

        {

            return;

        }

        int idx1;

        for(int i=n-2;i>=0;i--)

        {

            if(nums[i]<nums[i+1])

            {

                idx1=i;

                break;

            }

        }
```

```
        if(idx1<0)
        {
            reverse(nums.begin(),nums.end());
        }

        else
        {
            int idx2;
             for(int i=n-1;i>=0;i--)
                {
                    if(nums[idx1]<nums[i])
                    {
                        idx2=i;
                        break;
                    }
                }

        swap(nums[idx1],nums[idx2]);
        sort(nums.begin()+idx1+1,nums.end());
        }


    }
};
```