1) Two Sum :

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int n = nums.size();
        vector<int> op;
        int sum =0;

        for(int i =0; i<n-1; i++){
            for(int j = i+1;j<n; j++){
                sum = nums[i]+ nums[j];
                if(sum == target){
                    op.push_back(i);
                    op.push_back(j);
                    break;
                }
            }
            if(op.size()!=0){
                break;
            }
        }
        return op; } };
```

2) Palindrome Number

```
class Solution {
public:
    bool isPalindrome(int x) {
        if(x<0 || (x%10 == 0 && x!=0 )){
            return false;
        }
        int num = x;
```

```
        cout<<num;
        int copy = num;
        int rev = 0;
        int digit;

        while(copy>0){
            digit = copy%10;
            if (rev > INT_MAX / 10 || (rev == INT_MAX / 10 && digit > INT_MAX
%10)) {
                return false;
            }
            rev = (rev*10) + digit;
            copy/=10;
        }
        if(num==rev){
            return true;
        }
        else{
            return false;
        } } };
```

3) Remove element:

```
class Solution {
public:
    int removeElement(vector<int>& nums, int val) {
        int n = nums.size();
        int fp = 0;
        for(int i = 0; i<n; i++){
            if(nums[i]!=val){
                nums[fp] = nums[i];
                fp++;
            }
        }
```

```
        return fp; } };
```

4) Remove duplicates from sorted array :

```
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        int n = nums.size();
        int fp = 0;

        for(int i = 1; i<n; i++){
            if(nums[fp]!=nums[i]){
                nums[fp+1] = nums[i];
                fp++;
            }
        }
        return fp+1; } };
```

5) Merge sorted array :

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i = m-1;
        int j = n-1;
        int k = m + n -1;
        while(i>=0 && j>=0){
            if(nums1[i]>nums2[j]){
                nums1[k] = nums1[i];
                i--;
            }
            else{
                nums1[k] = nums2[j];
                j--;
```

```
        }
        k--;
    }
    while(j>=0){
        nums1[k] = nums2[j];
        j--;
        k--;
    } } };
```

6) Search insert position :

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int op;
        for(int i = 0; i<nums.size(); i++){
            if(nums[i] == target){
                op = i;
                break;
            }
            else if(nums[i]>target){
                op = i;
                break;
            }
        }
        return op; } };
```

7) Plus one :

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        int n = digits.size();
        int lastDigit = digits[n-1];
```

```
        for(int i = n-1; i>=0; i--){
        if(digits[i]<9){
            digits[i]++;
            return digits;
        }

        else{
            digits[i] = 0;
        }
        }
        digits.insert(digits.begin(), 1);
        return digits; } };
```

8) Buy and sell stock

```
   class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        if(n == 0){
            return 0;
        }
        int buy = prices[0];
        int profit = 0;
        for (int i = 1; i < n; i++) {
            buy = min(prices[i], buy);

            profit = max(profit, prices[i] - buy);
        }
        Return profit; } };
```

9) Check if array is sorted(rotated) :

```
class Solution {
```

```
public:
    bool check(vector<int>& nums) {
        int n = nums.size();
        int count = 0;
        for(int i = 0; i<n; i++){
            if(nums[i]>nums[(i+1) % n]){
                count++;
            }
        }
        if(count>1){
            return false;
        }
        return true; } };
```

10) Contains duplicate

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        int n = nums.size();
        unordered_set<int> seen;


        for(int i = 0; i<n; i++){
            if(seen.find(nums[i]) != seen.end()){
                return true;
            }
            seen.insert(nums[i]);
        }
        return false; } };
```

11) Median of two sorted array (hard)

```
class Solution {
```

```
public:
   double findMedianSortedArrays(vector<int>& nums1, vector<int>&
nums2) {

        int n=nums1.size();
        int m=nums2.size();
        vector<int> result;
        int i=0;
        int j=0;
        float floor=0;
        while(i<n && j<m){
            if(nums1[i]<nums2[j]){
                result.push_back(nums1[i]);
                i++;}
            else{
                result.push_back(nums2[j]);
                J++; }
            }
            while(i<n){
                result.push_back(nums1[i]);
                i++;
           }
           while(j<m){
                result.push_back(nums2[j]);
                j++;
           }
           if((n+m)%2==0){
             int a=(n+m)/2;
             int b=a-1;
              floor=(result[a]+result[b])/2.0;
           }
           else {
            int a=(n+m)/2;
            floor=result[a];
           }
```

```
        return floor; } };
```

12) Left rotation :

Normal method (me)

```
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n = nums.size();
        int temp;
        k = k % n;
        while(k > 0){
            temp = nums[n-1];
            for(int i = n-1; i>0; i--){
                nums[i] = nums[i-1];
            }
            nums[0] = temp;
            k--;
        } } };
```

Efficient method :

```
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n = nums.size();
        k = k % n;

        reverse(nums.begin(), nums.end());
        reverse(nums.begin(), nums.begin() + k);
        reverse(nums.begin() + k, nums.end());
    }
};
```

13) Move zeroes to end:

Normal method (me)

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int n = nums.size();

        int sp = n-1;

        for(int i = 0; i<n-1; i++){
            if(nums[i] == 0){
                nums[i] = nums[i+1];
                nums.push_back(0);
                sp--;
                nums.erase(nums.begin() + i+1);
            }
        } } };
```

Efficient method :

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int n = nums.size();
        int nonZero = 0;

        for(int i = 0; i<n; i++){
            if(nums[i] != 0){
                nums[nonZero] = nums[i];
                nonZero++;
            }
        }
        for(int i = nonZero; i<n; i++){
```

```
        nums[i] = 0;
    } } };
```

Another optimal :

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int n = nums.size();
        int j = 0;

        for(int i = j; i<n; i++){
            if(nums[i] != 0){
                swap(nums[i], nums[j]);
                j++;
            } } } };
```

14) Missing Number :

Brute force : Linear search, check one by one
Better : hashing

```
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int n = nums.size();
        vector<bool> op(n+1, false);

        for(int i = 0; i<n; i++){
            op[nums[i]] = true;
        }

        for(int i = 0; i<=n; i++){
            if(!op[i]) return i;
        }
```

```
return -1; } };
```

Another efficient : sum

Another efficient : Xor

```
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int n = nums.size();
        int xorAll = 0;

        for(int i = 0; i<n; i++){
            xorAll ^= i;
            xorAll ^= nums[i];
        }
        xorAll ^= n;

        return xorAll;
    }
};
```

15) Max consecutive of ones :

```
class Solution {
public:
    int findMaxConsecutiveOnes(vector<int>& nums) {
        int n = nums.size();
        int maxCon = 0;
        int con = 0;

        for(int i = 0; i<n; i++){
            if(nums[i] == 1){
                con++;
                maxCon = max(con, maxCon);
            }
```

```
        Else con = 0;
    }
    return maxCon; } };
```

Consecutive of any numbers or letters in string :

```
class Solution {
public:
   int findMaxConsecutive(vector<int>& nums) {
      if (nums.empty()) return 0;
      int maxCon = 1;
       int con = 1;
      for (int i = 1; i < nums.size(); i++) {
         if (nums[i] == nums[i - 1]) {
            con++;
            maxCon = max(maxCon, con);
         } else con = 1;
      }
      return maxCon; } };
```

16) Longer consecutive of 1 than 0 :

```
class Solution {
public:
   bool checkZeroOnes(string s) {
      int n = s.size();
      int max0 = 0;
      int maxCon0 = 0;
      int max1 = 0;
      int maxCon1 = 0;

      for(int i = 0; i<n; i++){
         if(s[i] == '0'){
            max1 = 0;
```

```
            max0++;
            maxCon0 = max(maxCon0, max0);
        }
        else if(s[i] == '1'){
            max0 = 0;
            max1++;
            maxCon1 = max(max1, maxCon1);
        }
    }
    if(maxCon0 == maxCon1) return false;
    else if(maxCon0 > maxCon1) return false;
    return true; } };
```

17) Single Number :

Brute : Linear search

Using map : Better

```
class Solution {
public:
    bool checkZeroOnes(string s) {
        class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int n = nums.size();

        map<int, int> mpp;

        for(int i = 0; i<n; i++){
            mpp[nums[i]]++;
        }

        for(auto it : mpp){
            if(it.second == 1) return it.first;
        }
```

```
 return 0; } };
```

Optimal : Xor

```
int xor1 = 0;
    int n = nums.size();

    for(int i = 0; i<n; i++){
     xor1 ^= nums[i];
    }
    return xor1;
```

18) Sort colors(0s 1s 2s):

Brute : Normal sorting, 2 loops

Efficient method : Hash

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
      vector<int> counter(3,0);
       int n = nums.size();

      for(int i = 0; i<n; i++){
          counter[nums[i]]++;
      }
      for(int i = 0; i<n; i++){
       if(counter[0] > 0){
          nums[i] = 0;
          counter[0]--;
       }
       else if(counter[1] > 0){
          nums[i] = 1;
          counter[1]--;
       }
```

```
            else if(counter[2] > 0){
                nums[i] = 2;
                Counter[2]--; } } }
        };
```

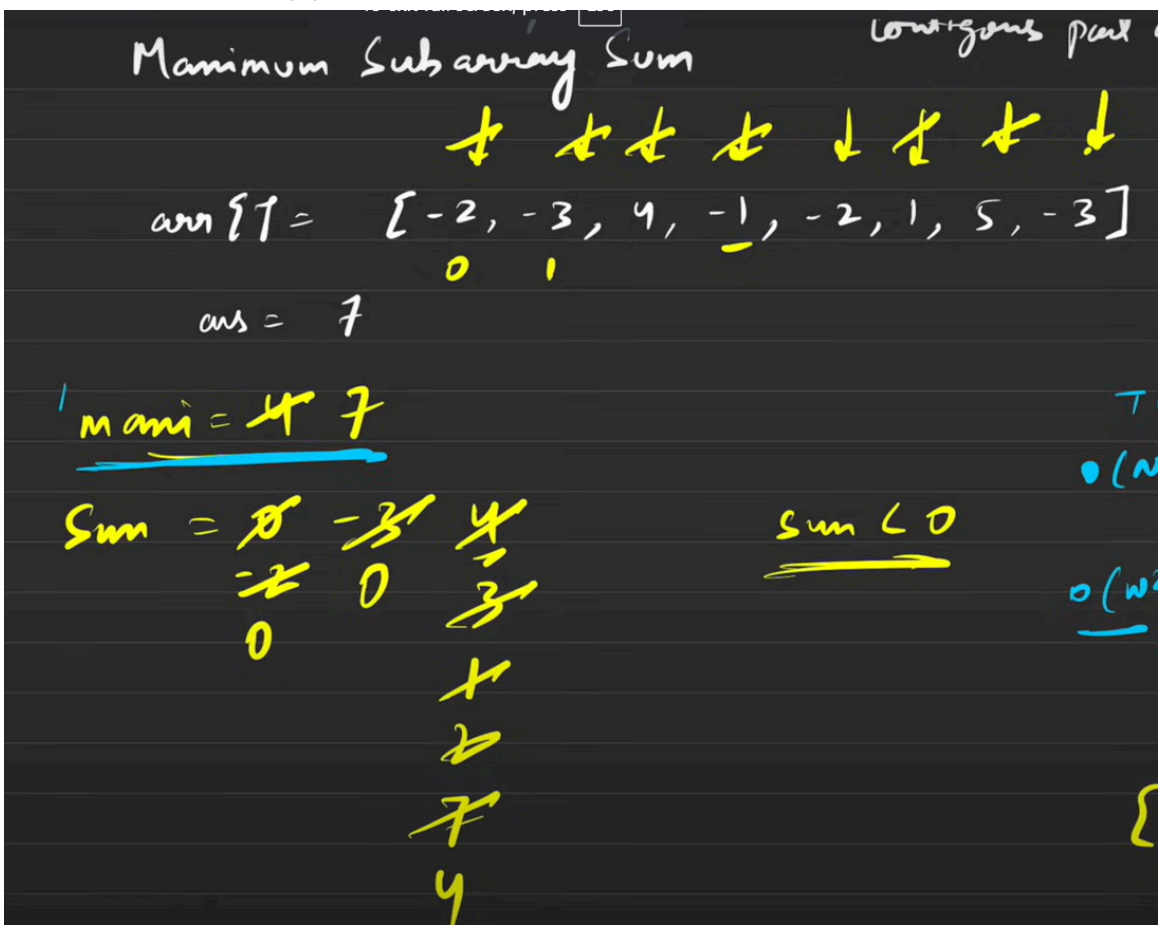
19) Next permutation :

```
class Solution {
public:
    void nextPermutation(vector<int>& nums) {
        int index = -1;
        int n = nums.size();

        for(int i = n-2; i>=0; i--){
            if(nums[i] < nums[i+1]){
                index = i;
                break;
            }
        }
        if(index == -1){
            reverse(nums.begin(), nums.end());
        }
        else{
            for(int i = n-1; i>index; i--){
            if(nums[i] > nums[index]){
                swap(nums[i], nums[index]);
                break;
            }
        }
        reverse(nums.begin() + index+1, nums.end()); } } };
```

20) Maxi subarray :

```
class Solution {
public:
```

```
int maxSubArray(vector<int>& nums) {
    int n = nums.size();
    int maxi = INT_MIN;
    int sum = 0;

    for(int i = 0; i<n; i++){
        sum+=nums[i];
        maxi = max(sum, maxi);
        if(sum<0){
            sum = 0;
        }
    }
    return maxi; } };
```



21) Find index of first occurrence of string :

```
class Solution {
public:
    int strStr(string haystack, string needle) {
        int n = haystack.size();
        int m = needle.size();
        int j = 0;
        for(int i = 0; i<n; i++){
            if(haystack[i] == needle[j]){
                j++;
                if(j==m){
                    return i-m+1;
                    break;
                }
            }
            else{
                i = i-j;
                 j = 0;
            }
        }
        return -1; } };
```

22) Length of last word :

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int n = s.size();
        int count = 0;
        int letterFound = 0;

        for(int i = n-1; i>=0; i--){
            if(s[i] != ' '){
```

```
            letterFound = i;
            break;
        }
    }
    for(int i = letterFound; i>=0; i--){
            if(s[i] != ' '){
                count++;
            }else{
                break;
            }
    }
    return count; } };
```

Another method, only one loop :

```
class Solution {
public:
    int lengthOfLastWord(string s) {
        int n = s.size();
        int count = 0;

        for (int i = n - 1; i >= 0; i--) {
            if (s[i] != ' ') {
                count++;
            } else if (count > 0) {
                break;
            }
        }
        return count; } };
```

Optimized :

```
class Solution {
public:
    bool isPalindrome(string s) {
```

```
        int n = s.size();
        int left = 0;
        int right = n-1;

        while(left<right){
            while(left<right && !isalnum(s[left])){
                left++;
            }
            while(left<right && !isalnum(s[right])){
                right--;
            }
            if(tolower(s[left]) != tolower(s[right])){
                return false;
            }
            left++;
            right--;
        }
        return true; } };
```

23) Majority element (>n/2) :

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int n = nums.size();
        unordered_map<int, int> mpp;

        for(int i = 0; i<n; i++){
            mpp[nums[i]]++;

        if(mpp[nums[i]]>(n/2)) return nums[i];
        }
        return 0; } };
```

24) Happy Number(sqr and sum until 1)

```
class Solution {
```

```
public:
    bool isHappy(int n) {
        unordered_set<int> seen;
        int digits = 0;
        int squaredNumber = 0;

        while(n != 1 && seen.find(n) == seen.end()){
            seen.insert(n);
                squaredNumber = 0;

            while(n>0){
                digits = n % 10;
                squaredNumber = squaredNumber + (digits*digits);
                n = n /10;
            }
            n = squaredNumber;
        }
        if(n == 1){
            return true;
        }
        return false; } };
```

Optimized :

```
class Solution {
public:
    int getNext(int num) {
        int sum = 0;
        while (num > 0) {
            int digit = num % 10;
            sum += digit * digit;
            num /= 10;
        }
        return sum;
```

```
    }

    bool isHappy(int n) {
        int slow = n;
        int fast = getNext(n);

        while (fast != 1 && slow != fast) {
            slow = getNext(slow);
            fast = getNext(getNext(fast));
        }

        if(fast == 1){
            return true;
        }
        return false;
    }
};
```

25) Intersection of 2 unsorted arrays: (repetition allowed)

```
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
        int n1 = nums1.size();
        int n2 = nums2.size();
        unordered_map<int, int> mpp;
        vector<int> result;

        for (int i = 0; i < n1; i++) {
            mpp[nums1[i]]++;
        }

        for (int i = 0; i < n2; i++) {
            if (mpp[nums2[i]] > 0) {
```

```
            result.push_back(nums2[i]);
            mpp[nums2[i]]--;
        }
    }
    return result; } };
```

Same intersection, Repetition Not allowed :

```
class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
        int n1 = nums1.size();
        int n2 = nums2.size();
        vector<int> inter;
        unordered_map<int, int> mpp;

        for (int i = 0; i < n1; i++) {
            mpp[nums1[i]]++;
        }

        for (int i = 0; i < n2; i++) {
            if (mpp[nums2[i]] > 0) {
                inter.push_back(nums2[i]);
                mpp[nums2[i]] = 0;
            }
        }
        return inter; } };
```

26) Third Largest :
Three pointer

```
class Solution {
public:
```

```
    int thirdMax(vector<int>& nums) {
        int n = nums.size();

        long one = LONG_MIN;
        long two = LONG_MIN;
        long three = LONG_MIN;

        for (int i = 0; i < n; i++) {
            if (nums[i] > one) {
                three = two;
                two = one;
                one = nums[i];
            } else if ((nums[i] < one) && (nums[i] > two)) {
                three = two;
                two = nums[i];
            } else if ((nums[i] < two) && (nums[i] > three)) {
                three = nums[i];
            }
        }
        if (three == LONG_MIN) return one;
        return three; } };
```

27) Neither max nor min :

```
class Solution {
public:
    int findNonMinOrMax(vector<int>& nums) {
        int n = nums.size();
        if (n < 3) return -1;

        int mini = INT_MAX;
        int maxi = INT_MIN;

        for (int i = 0; i < n; i++) {
            if (nums[i] < mini) mini = nums[i];
```

```
        if (nums[i] > maxi) maxi = nums[i];
    }

    for (int i = 0; i < n; i++) {
        if ((nums[i] != mini) && (nums[i] != maxi)) {
            return nums[i];
        }
    }
    return -1; } };
```

28) Contains duplicate(their index abs == k)

```
class Solution {
public:
    bool containsNearbyDuplicate(vector<int>& nums, int k) {
        int n = nums.size();
        unordered_map<int, int> mpp;

        for (int i = 0; i < n; i++) {
            if (mpp.find(nums[i]) != mpp.end()) {
                if (i - mpp[nums[i]] <= k) {
                    return true;
                }
            }
            mpp[nums[i]] = i;
        }
        return false; } };
```

29) Summary Ranges([0,1,2,4,5,7] —-->
["0->2","4->5","7"])

```
class Solution {
public:
    vector<string> summaryRanges(vector<int>& nums) {
        int n = nums.size();
        if (n == 0) {
```

```
        return {};
    }

    int ptr = 0;
    vector<string> result;

    for (int i = 1; i < n; i++) {
        if (static_cast<long long>(nums[i]) - static_cast<long long>(nums[i -
1]) != 1) {
            if (ptr == i - 1) {
                result.push_back(to_string(nums[ptr]));
            } else {
                result.push_back(to_string(nums[ptr]) + "->" + to_string(nums[i
- 1]));
            }
            ptr = i;
        }
    }
    if (ptr == n - 1) {
        result.push_back(to_string(nums[ptr]));
    } else {
        result.push_back(to_string(nums[ptr]) + "->" + to_string(nums[n -
1]));
}

    return result; } };
```

30) Difference of two arrays ;

```
class Solution {
public:
   vector<vector<int>> findDifference(vector<int>& nums1, vector<int>&
nums2) {
    unordered_set<int> set1(nums1.begin(), nums1.end());
    unordered_set<int> set2(nums2.begin(), nums2.end());
```

```
        vector<vector<int>> result(2);

        for (int num : set1) {
            if (set2.find(num) == set2.end()) {
                result[0].push_back(num);
            }
        }

        for (int num : set2) {
            if (set1.find(num) == set1.end()) {
                result[1].push_back(num);
            }
        }

        return result; } };
```

31) Minimum common value :

```
class Solution {
public:
    int getCommon(vector<int>& nums1, vector<int>& nums2) {
        int i = 0, j = 0;

        while (i < nums1.size() && j < nums2.size()) {
            if (nums1[i] == nums2[j]) {
                return nums1[i];
            } else if (nums1[i] < nums2[j]) {
                i++;
            } else {
                j++;
            }
        }

        return -1; } };
```

32) Intersection of multiple arrays :

```
class Solution {
public:
    vector<int> intersection(vector<vector<int>>& nums) {
        unordered_set<int> commonSet(nums[0].begin(), nums[0].end());

        for (int i = 1; i < nums.size(); ++i) {
            unordered_set<int> currentSet(nums[i].begin(), nums[i].end());
            unordered_set<int> tempSet;

            for (auto& num : commonSet) {
                if (currentSet.find(num) != currentSet.end()) {
                    tempSet.insert(num);
                }
            }
            commonSet = tempSet;
        }

        vector<int> result(commonSet.begin(), commonSet.end());
        sort(result.begin(), result.end());
        return result; } };
```

33) Roman to integer :

```
class Solution {
public:
    int romanToInt(string s) {
        int n = s.size();
        map<char, int> mpp;
        mpp['I'] = 1;
        mpp['V'] = 5;
        mpp['X'] = 10;
        mpp['L'] = 50;
        mpp['C'] = 100;
        mpp['D'] = 500;
```

```
    mpp['M'] = 1000;

    int result = 0;

    for (int i = 0; i < n; i++) {
        if ((i < n - 1) && (mpp[s[i]] < mpp[s[i + 1]]))
            result -= mpp[s[i]];
        else
            result += mpp[s[i]];
    }
    return result; } };
```

34) Integer to Roman :

```
class Solution {
public:
    string intToRoman(int num) {
        vector<string> stValues = {"M",  "CM", "D",  "CD", "C",  "XC", "L",
                                   "XL", "X",  "IX", "V",  "IV", "I"};
        vector<int> inValues = {1000, 900, 500, 400, 100, 90, 50,
                                40,   10,  9,   5,   4,   1};

        string result;
        int i = 0;
        while (num > 0) {
            while (num >= inValues[i]) {    //while used instead of If
                result += stValues[i];      bcz, it should run until it fails
                num -= inValues[i];         or i will be incremented
            }
            i++;
        }
        return result; } };
```

More optimized : (No separate memory allocation)

```
class Solution {
public:
```

```
    string intToRoman(int num) {
        const pair<int, string> values[] = {            //Const avoids dynamic
            {1000, "M"}, {900, "CM"}, {500, "D"}, {400, "CD"},
            {100, "C"},  {90, "XC"},  {50, "L"},  {40, "XL"},
            {10, "X"},   {9, "IX"},   {5, "V"},   {4, "IV"}, {1, "I"}};

        string result;
        int i = 0;
        while (num > 0) {
            while (num >= values[i].first) {
                result += values[i].second;
                num -= values[i].first;
            }
            i++;
        }
        return result; } };
```

35) Longest common prefix :

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        string result = "";
        int n = strs.size();
        int ptr = 0;

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

        string first = strs[0];
        string last = strs[n - 1];
        int len = min(first.length(), last.length());

        for (int i = 0; i < len; i++) {
            if (first[i] != last[i])
                break;
            result += first[i];
```

```
        }
        return result; } };
```

36) Set Matrix Zeroes :

```
class Solution {
public:
    void setZeroes(vector<vector<int>>& matrix) {
        int n = matrix.size();
        int m = matrix[0].size();
        vector<int> col(m,0);
        vector<int> row(n, 0);

        for(int i = 0; i<n; i++){
            for(int j = 0; j<m; j++){
                if(matrix[i][j] == 0){
                    row[i] = 1;
                    col[j] = 1;
                }
            }
        }

        for(int i = 0; i<n; i++){
            for(int j = 0; j<m; j++){
                if((col[j] == 1) || (row[i] ==1)){
                    matrix[i][j] = 0;
                }
            }
        }
    }
};
```

37) Rotate matrix by 90 deg:

```
class Solution {
public:
```

```
void rotate(vector<vector<int>>& matrix) {
    int n = matrix.size();
    int m = matrix[0].size();

    // transpose
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (i != j) {
                swap(matrix[i][j], matrix[j][i]);
            }
        }
    }
    // reverse rows
    for (int i = 0; i < n; i++) {
        reverse(matrix[i].begin(), matrix[i].end());
    } } };
```

38) Longest subarray with sum k:

Better:

```
#include <bits/stdc++.h>
int getLongestSubarray(vector<int> a, long long k)  {
    map<long long, int> mpp;
    long long sum = 0;
    int len = 0;
    int maxLen = 0;

    for(int i = 0; i < a.size(); i++){
        sum += a[i];
        if(sum == k){
            maxLen = max(maxLen, i+1);
        }

        long long rem = sum - k;
```

```
        if(mpp.find(rem) != mpp.end()){
            len = i - mpp[rem];
            maxLen = max(maxLen, len);
        }
        if(mpp.find(sum) == mpp.end()){
            mpp[sum] = i;
        }
    }
    return maxLen; };
```

Optimal :

```
#include <map>
int longestSubarrayWithSumK(vector<int> a, long long k) {
    int n = a.size();
    long long sum = a[0];
    int len = 0;
    int maxLen = 0;
    int fp = 0, sp = 0;

    while(sp < n){
        while(fp <= sp && sum > k){
            sum = sum - a[fp];
            fp++;
        }
        if(sum == k) len = sp - fp+1;
        sp++;
         if(sp < n){
             sum += a[sp];
         }
        maxLen = max(maxLen, len);
    }
    return maxLen; };
```

39) Max 2 consecutive sum :

```
class Solution {
```

```
  public:
    int pairWithMaxSum(vector<int> &arr) {
        int n = arr.size();
        int sum = 0;
        int maxi = INT_MIN;
        for(int i = 0; i < n-1; i++){
            sum = arr[i] + arr[i+1];
            maxi = max(maxi, sum);
    }
    return maxi; } };
```

40) Reverse Prefix of a str:

```
class Solution {
public:
    string reversePrefix(string word, char ch) {
        int n = word.size();
        int first = -1;
        for (int i = 0; i < n; i++) {
            if (word[i] == ch){
                first = i;
                break;
            }
        }
        if(first == -1) return word;
        reverse(word.begin(), word.begin() + first + 1);
        return word;} };
```

41) Number of substrs in a str from an array of strings:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;
```

```
class Solution {
public:
    int numOfStrings(vector<string>& patterns, string word) {
        int n = patterns.size();
        int count = 0;
        string temp;

        for (int i = 0; i < n; i++) {
            temp = patterns[i];
            size_t pos = word.find(temp);  // 'pos' stores the index of the first
occurrence of 'temp' in 'word', or string::npos if not found

            if (pos != string::npos) {  // 'npos' is a constant representing "not
found" in the string, so this checks if the pattern exists
                count++;
            }
        }
        return count;  } };
```

42) Check if string is a prefix of an array :

```
class Solution {
public:
    bool isPrefixString(string s, vector<string>& words) {
        string prefix = "";

        for(int i = 0; i < words.size(); i++){
            prefix += words[i];
            if(s == prefix){
                return true;
            }
        }
        return false; } };
```

43) Sum of digits of string after convert :

```
class Solution {
```

```
public:
    int getLucky(string s, int k) {
        int n = s.size();

        string Final;
        char temp;
        string afterConversion;
        int num, numeric, finalNum;

        for(int i = 0; i < n; i++){
            temp = s[i];
            numeric = static_cast<int>(temp);
            finalNum = numeric - 96;
            afterConversion = to_string(finalNum);
            Final += afterConversion;
        }

        int sum = 0;

        while(k > 0){
            sum = 0;
            for (char c : Final) {
                sum += (c - '0');
            }
            Final = to_string(sum);
            k--;
        }
        return sum; } };
```

44) Min time to type words(circular typewriter) :

```
class Solution {
public:
    int minTimeToType(string word) {
        int n = word.size();
        int front, back, miniStep;
```

```
        int op = 0;
        char currentChar;
        char prevChar = 'a';

        for(int i = 0; i < n; i++){
            currentChar = word[i];
            front = abs(currentChar - prevChar);
            back = 26 - front;
            miniStep = min(front, back);
            op += miniStep;
            prevChar = currentChar;
        }
        op += n;
        return op; } };
```

45) Check if numbers are ascending in a string :

```
class Solution {
public:
    bool areNumbersAscending(string s) {
        int n = s.size();
        int prevNum = 0;
        int num = 0;
        string numericalString = "";

        for(char ch : s){
            if(isdigit(ch)){
                numericalString += ch;
            }
            else if(!numericalString.empty()){
                num = stoi(numericalString);
                numericalString = "";
                if(num <= prevNum){
                    return false;
                }
                prevNum = num;
```

```
        }
    }
    if (!numericalString.empty()) {     //Edge case(last word is num in str)
        int num = stoi(numericalString);
        if (num <= prevNum) {
            return false;
        }
    }
    return true; } };
```

46) Kth distinct string in array :

```
class Solution {
public:
    string kthDistinct(vector<string>& arr, int k) {
        unordered_map<string, int> mpp;
        int n = arr.size();
        string result;

        for(int i = 0; i < n; i++){
            mpp[arr[i]]++;
        }
        for(int i = 0; i < n; i++){
            if(mpp[arr[i]] == 1){
                k--;
                if(k == 0) return arr[i];
            }
        }
        return ""; } }
```

47) Check if two strings are almost equivalent:

```
class Solution {
public:
    bool checkAlmostEquivalent(string word1, string word2) {
        int n1 = word1.size();
        int n2 = word2.size();
        int freq[26] = {0};
```

```
        for(int i = 0; i < max(n2, n1); i++){
            if(i < n1) freq[word1[i] - 'a']++;
            if(i < n2) freq[word2[i] - 'a']--;
        }
        for(int i = 0; i < 26; i++){
            if(abs(freq[i]) > 3) return false;
        }
        return true; } };
```

48) First palindrome string in a array:

Better(2ms)

```
class Solution {
public:
    string firstPalindrome(vector<string>& words) {
        int n = words.size();
        string word;
        string revWord;
        string result;

        for(int i = 0; i < n; i++){
            word = words[i];
            revWord = words[i];
            reverse(revWord.begin(), revWord.end());
            if(word == revWord){
                result = word;
                break;
            }
        }
        return result; } };
```

Optimal (0ms) :

```
class Solution {
public:
    string firstPalindrome(vector<string>& words) {
        int n = words.size();
```

```
        string word;
        int left, right;

        for(int i = 0; i < n; i++){
            word = words[i];
            left = 0;
            right = word.size() -1;
            while(left <= right && word[left] == word[right]){
                left++;
                right--;
            }
            if(left >= right) return word;
        }
        return ""; } };
```

49) Rings and rods :

```
class Solution {
public:
    int countPoints(string rings) {
        int n = rings.size();
        int count = 0;
        vector<int> num(10);
        char character;

        for (int i = 0; i < n; i++) {
            character = rings[i];
            int rod = rings[i + 1] - '0';

            if (character == 'R')
                num[rod] |= 1;
            if (character == 'G')
                num[rod] |= 2;
            if (character == 'B')
                num[rod] |= 4;
        }
```

```
        for (int i = 0; i < 10; i++) {
            if (num[i] == 7)
                count++;
        }

        return count; } };
```

B0B6G0R6R0R6G9 (Example)

50) Vowel substring :

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

class Solution {
public:
    int countVowelSubstrings(string word) {
        int n = word.size();
        int count = 0;

        for (int i = 0; i < n; i++) {
            unordered_map<char, int> freq;

            for (int j = i; j < n; j++) {
                char letter = word[j];

                if (letter == 'a' || letter == 'e' || letter == 'i' || letter == 'o' || letter ==
'u') {
                    freq[letter]++;
                    if (freq.size() == 5) count++;   // whether aeiou presents or
                }
                else break;
            }
        }

        return count; } };
```

Start Index (i)	Expanding j	Vowel Set	Count
0 → "a"	"a"	{a}	0
0 → "ae"	"ae"	{a, e}	0
0 → "aei"	"aei"	{a, e, i}	0
0 → "aeio"	"aeio"	{a, e, i, o}	0
0 → "aeiou"	"aeiou"	{a, e, i, o, u}	✅ +1
0 → "aeiouu"	"aeiouu"	{a, e, i, o, u}	✅ +2
1 → "e"	"e"	{e}	0
1 → "ei"	"ei"	{e, i}	0
1 → "eio"	"eio"	{e, i, o}	0
1 → "eiou"	"eiou"	{e, i, o, u}	0
1 → "eiouu"	"eiouu"	{e, i, o, u}	0
...	...	...	...

✔ Total Count = 2 for "aeiouu" ✅

51) No A should appear after B :

```
class Solution {
public:
    bool checkString(string s) {
        int n = s.size();
        bool bFound = false;
        bool aFound = false;
        char letter;

        for(int i = 0; i < n; i++){
            letter = s[i];
            if(letter == 'b') bFound = true;
            if(letter == 'a' && bFound == true){
                return false;
            }
        }
```

```
        return true; } };
```

52) Count prefixes of a string :

```
class Solution {
public:
    int countPrefixes(vector<string>& words, string s) {
        int n = words.size();
        string word;
        int count = 0;

        for(int i = 0; i < n; i++){
            word = words[i];
            if(isPrefix(s, word)) count++;

        }
        return count; }

    bool isPrefix(const string& s, const string& word){
        return s.compare(0, word.size(), word) == 0; } }; //compare() returns
```

0 for an exact match so ==0 is used.

53) Check if a word occurs as a prefix as any word of the sentence :

```
class Solution {
public:
    int isPrefixOfWord(string sentence, string searchWord) {
        int n = sentence.size();
        string word = "";
        vector<string> words;

        for(int i = 0; i < n; i++){
            if(sentence[i] != ' '){
                word+=sentence[i];
            }
```

```
        else if(!word.empty()){
            words.push_back(word);
            word = "";
        }
    }

    if(!word.empty()){
        words.push_back(word);
        word = "";
    }        //Last word in the sentence would not be added in for loop
as there is no space.

    for(int i = 0; i < words.size(); i++){
        if(isPrefix(words[i], searchWord)) return i+1;
    }
    return -1; }

  bool isPrefix(const string& word, const string& searchWord){
    return word.compare(0, searchWord.size(), searchWord) == 0; } };
```

54) Counting words with given prefix :

```
class Solution {
public:
  int prefixCount(vector<string>& words, string pref) {
    int  n = words.size();
    int count = 0;

    for(int i = 0; i < n; i++){
        if(isPrefix(words[i], pref)) count++;
    }

    return count; }

  bool isPrefix(const string& word, const string& pref){
```

```
        return word.compare(0, pref.size(), pref) == 0; } };
```

55) Valid anagram :

7ms Time with map:

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        int n1 = s.size();
        int n2 = t.size();

        if(n1 != n2) return false;

        map<char, int> mpp1;
        map<char, int> mpp2;
        for(char ch = 'a'; ch <= 'z'; ch++) {
            mpp1[ch] = 0;
            mpp2[ch] = 0;
        }

        for(int i = 0; i < n1; i++){
            mpp1[s[i]]++;
            mpp2[t[i]]++;
        }
        for(char ch = 'a'; ch <= 'z'; ch++) {
            if(mpp1[ch] != mpp2[ch]) return false;
        }

        return true; } };
```

0ms with array only:

```
class Solution {
public:
    bool isAnagram(string s, string t) {
        int n1 = s.size();
        int n2 = t.size();
```

```
        if(n1 != n2) return false;

        int arr[26] = {0};

        for(int i = 0; i < n1; i++){
            arr[s[i] - 'a']++;
            arr[t[i] - 'a']--;
        }

        for(int i = 0; i < 26; i++){
            if(arr[i] != 0) return false;
        }

        return true; } };
```

56) Check if a number has equal digit value and digit count:

```
class Solution {
public:
    bool digitCount(string num) {
        int n = num.size();
        vector<int> freq(10,0);

        for(int i = 0; i < n; i++){
            freq[num[i] - '0']++;
        }
        for(int i = 0; i < n; i++){
            if(num[i] - '0' != freq[i]) return false;
        }
        return true; } };
```

57) Percentage of letter in string :

```
class Solution {
public:
    int percentageLetter(string s, char letter) {
        int n = s.size();
        char letterFromS;
```

```
        int count = 0;

        for(int i = 0; i < n; i++){
            letterFromS = s[i];
            if(letterFromS == letter) count++;
        }
        return count*100/n; } };
```

58) Reverse string without extra space:

```
class Solution {
public:
    void reverseString(vector<char>& s) {

        int lp = 0;
        int rp = s.size()-1;

        while(lp <= rp){
            swap(s[lp], s[rp]);
            lp++;
            Rp--;
} } };
```

59) Min number of operations to convert time(60, 15, 5, 1):

```
class Solution {
public:
    int convertTime(string current, string correct) {
        int currentMins = 0;
        int correctMins = 0;
        int hour = 0;
        int min = 0;

        hour = ((current[0] - '0') * 10 + current[1] - '0') ;
        min = ((current[3] - '0') * 10 + current[4] - '0') ;
        currentMins = (hour*60) + min;
```

```
        hour = 0;
        min = 0;

        hour = ((correct[0] - '0') * 10 + correct[1] - '0') ;
        min = ((correct[3] - '0') * 10 + correct[4] - '0') ;
        correctMins = (hour*60) + min;

        if(currentMins == 0 && correctMins == 0) return 0;

        int diff = correctMins - currentMins;
        int count = 0;

        count += diff / 60;        //Greedy Algo
        diff = diff % 60;
        count += diff / 15;
        diff = diff % 15;
        count += diff / 5;
        diff = diff % 5;
        count += diff / 1;
        diff = diff % 1;

        return count; } };
```

60) Count common words with one occurance:

```
class Solution {
public:
    int countWords(vector<string>& words1, vector<string>& words2) {
        int count = 0;
        unordered_map<string, int> wordsMap1, wordsMap2;

        for(const string& word : words1){
            wordsMap1[word]++;
        }
        for(const string& word : words2){
            wordsMap2[word]++;
```

```
        }

        for(const auto& [word, freq] : wordsMap1){
            if(freq == 1 && wordsMap2[word] == 1){
                count++;
            }
        }
        return count; } };
```

61) Count star symbols(asterisks):

```
class Solution {
public:
    int countAsterisks(string s) {
        int n = s.size();
        int barCount = 0;
        int astCount = 0;

        for(int i = 0; i < n; i++){
            if(s[i] == '|') barCount++;

            if(barCount%2 == 0){
                if(s[i] == '*') astCount++;
            }
        }
        return astCount; } };
```

62) Decode the message:

```
class Solution {
public:
    string decodeMessage(string key, string message) {
        int n1 = key.size();
        int n2 = message.size();
        int ptr = 0;
        char mpp[128] {};
```

```
        string result = message;

        for(int i = 0; i < n1; i++){
            if(key[i] != ' ' && mpp[key[i]] == '\0'){
                mpp[key[i]] = ptr + 'a';
                ptr++;
            }
        }

        for(int i = 0; i < n2; i++){
            if(message[i] != ' '){
                result[i] = mpp[message[i]];
            }
        }
        return result; } };
```

63) Sort people based on height:

13ms time : pair

```
class Solution {
public:
    vector<string> sortPeople(vector<string>& names, vector<int>& heights)
{
        int n = names.size();
        vector<pair<int, string>> namesWithHeight;
        vector<string> result;

        for(int i = 0; i < n; i++){
            namesWithHeight.push_back({heights[i], names[i]});
        }

        sort(namesWithHeight.begin(), namesWithHeight.end(),
        [](const pair<int, string>& a, const pair<int, string>& b){
            return a.first > b.first;
        });
```

```
        for(auto& it : namesWithHeight){
            result.push_back(it.second);
        }
        return result; } };
```

0ms (only index vector) :

```
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

class Solution {
public:
    vector<string> sortPeople(vector<string>& names, vector<int>& heights)
{
        int n = names.size();
        vector<int> indices(n);
        for (int i = 0; i < n; i++) indices[i] = i;

        sort(indices.begin(), indices.end(), [&](int a, int b) {
            return heights[a] > heights[b];
        });

        vector<string> result(n);
        for (int i = 0; i < n; i++) {
            result[i] = names[indices[i]];
        }
        return result; } }
```

64) Find the difference :

3ms time(map)

```
class Solution {
public:
    char findTheDifference(string s, string t) {
        char result;
```

```
        int n1 = s.size();
        int n2 = t.size();
        unordered_map<char, int> freq;

        for(int i = 0; i < n1; i++){
            freq[s[i]]++;
        }

        for(int i = 0; i < n2; i++){
            freq[t[i]]--;
        }

        for(const auto& pair : freq){
            if(pair.second != 0) return pair.first;
        }
        return '0'; } };
```

0ms time(XOR):

```
class Solution {
public:
    char findTheDifference(string s, string t) {
        char result = 0;

        for(int i = 0; i < s.size(); i++){
            result ^= s[i];
        }
        for(int i = 0; i < t.size(); i++){
            result ^= t[i];
        }
        return result; } };
```

65) First letter to appear twice :

```
class Solution {
public:
    char repeatedCharacter(string s) {
```

```
        char result;
        unordered_map<char, int> freq;

        for(int i = 0; i < s.size(); i++){
            freq[s[i]]++;
            if(freq[s[i]] == 2){
                result = s[i];
                break;
            }
        }
        return result; } };
```

66) Largest 3 same dig sum:

```
class Solution {
public:
    string largestGoodInteger(string num) {
        string result = "";
        int n = num.size();
        int current = -1;
        int prev = -1;
        int count = 1;

        for(int i = 1; i < n; i++){
            if(num[i] == num[i-1]){
                count++;
            }
            else count = 1;
            if(count == 3){
                current = num[i] -'0';
                if(current > prev){
                    result = string(3, num[i]);
                    prev = current;
                }
            }
        }
```

```
        return result; }  };
```

Another easy method :

```
class Solution {
public:
    string largestGoodInteger(string num) {
        string result = "";
        string triplet;
        int n = num.size();
        for(int i = 0; i < n-2; i++){
            if(num[i] == num[i+1] && num[i+1] == num[i+2]){
                triplet = string(3, num[i]);
            }

            if(result.empty() || triplet > result){
                result = triplet;
            }
        }
        return result; } };
```

67) Reverse vowels of a string:

Vector char method:

```
class Solution {
public:
    string reverseVowels(string s) {
        int n = s.size();
        vector<char> vowels;
        char letter;
        int tracker;

        for(int i = 0; i < n; i++){
            letter = s[i];

            if(letter == 'a' || letter == 'e' || letter == 'i' || letter == 'o' || letter == 'u' ||
letter == 'A' || letter == 'E' || letter == 'I' || letter == 'O' || letter == 'U' ){
                vowels.push_back(letter);
```

```
        }
    }

    tracker = vowels.size() -1;

    for(int i = 0; i < n; i++){

        if(letter == 'a' || letter == 'e' || letter == 'i' || letter == 'o' || letter == 'u' ||
letter == 'A' || letter == 'E' || letter == 'I' || letter == 'O' || letter == 'U'){
            swap(s[i], vowels[tracker]);
            tracker--;
        }

    }
    return s; } };
```

Using two pointers:

```
class Solution {
public:
   string reverseVowels(string s) {
       string vowels = "aeiouAEIOU";
       int left = 0;
       int right = s.size() - 1;

       while(left < right){
           while(left < right && vowels.find(s[left]) == string::npos){
               left++;
           }
           while(left < right && vowels.find(s[right]) == string::npos){
               right--;
           }

           swap(s[left], s[right]);
           left++;
           right--;
```

```
        }
        return s; } };
```

68) Add Binary :

```
class Solution {
public:
    string addBinary(string a, string b) {
        int n1 = a.size();
        int n2 = b.size();
        int carry = 0;
        int sum = 0;
        string result;

        while (n1 > 0 || n2 > 0 || carry) {
            sum = carry;

            if (n1 > 0) {
                sum += a[n1 - 1] - '0';
                n1--;
            }
            if (n2 > 0) {
                sum += b[n2 - 1] - '0';
                n2--;
            }
            result.push_back((sum % 2) + '0');
            carry = sum / 2;
        }
        reverse(result.begin(), result.end());
        return result; } };
```

69) Check distance between same letters:

```
class Solution {
public:
    bool checkDistances(string s, vector<int>& distance) {
        int n = s.size();
        char letter;
```

```
        int index;
        vector<int> first(26, -1);

        for(int i = 0; i < n; i++){
            index = s[i] - 'a';

            if (first[index] != -1){
                if (i - first[index] - 1 != distance[index]){
                    return false;
                }
            }
            else{
                first[index] = i;
            }
        }
        return true; } };
```

70) FizzBuzz :

```
class Solution {
public:
    vector<string> fizzBuzz(int n) {
        vector<string> result;

        for(int i = 1; i <= n; i++){
            if(i % 3 == 0 && i % 5 == 0){
                result.push_back("FizzBuzz");
            }
            else if(i % 3 == 0){
                result.push_back("Fizz");
            }
            else if(i % 5 == 0){
                result.push_back("Buzz");
            }

            else{
```

```
            result.push_back(to_string(i));
        }
    }
    return result; } };
```

71) Keyboard rows :

```
class Solution {
public:
    vector<string> findWords(vector<string>& words) {
        vector<string> rows;
        rows = {"qwertyuiop", "asdfghjkl", "zxcvbnm"};
        vector<string> result;
        string word;
        char letter;

        for(int i = 0; i < words.size(); i++){
            vector<int> rowsFlag(3, 0);
            int sum = 0;
            word = words[i];
            for(int j = 0; j < word.size(); j++){
                letter = tolower(word[j]);
                if(rows[0].find(letter) != string::npos){
                    rowsFlag[0] = 1;
                }
                if(rows[1].find(letter) != string::npos){
                    rowsFlag[1] = 1;
                }
                if(rows[2].find(letter) != string::npos){
                    rowsFlag[2] = 1;
                }

            }

            for(auto it : rowsFlag){
                sum += it;
```

```
        }
        if(sum == 1){
            result.push_back(words[i]);
        }
    }
    return result; } };
```

72) Max values of a string in an array :

```
class Solution {
public:
    int maximumValue(vector<string>& strs) {
        int n = strs.size();
        string word;
        int count = 0;
        int maxValue = 0;

        for(int i = 0; i < n; i++){
            word = strs[i];
            if(isInt(word)){
                count = stoi(word);
            }
            else count = word.size();

            maxValue = max(count, maxValue);
        }
        return maxValue; }


    bool isInt(string s){
        if(s.empty()) return false;

        int i = 0;

        for(i = 0; i < s.size(); i++){
            if(!isdigit(s[i])) return false;
```

```
        }

        return true; } };
```

73) Count pairs of similar strings :

```
class Solution {
public:
    int similarPairs(vector<string>& words) {
        int n = words.size();
        int count = 0;
        vector<unordered_set<char>> letters(n);
        string word;

        for(int i = 0; i < n; i++){
            word = words[i];

            for(char it : word){
                letters[i].insert(it);
            }
        }

        for(int i = 0; i < n; i++){
            for(int j = i + 1; j < n; j++){
                if(letters[i] == letters[j]) count++;
            }
        }

        return count; } };
```

Bitmask method(efficient) :

```
class Solution {
public:
    int similarPairs(vector<string>& words) {
        unordered_map<int, int> freq;
        int count = 0;
```

```
for (string& word : words) {
    int mask = 0;
    for (char ch : word) {
        mask |= (1 << (ch - 'a'));
    }
    count += freq[mask];
    freq[mask]++;
}

return count; } };
```

📌 **How does `mask |= (1 << (ch - 'a'))` work?**

- `'a' - 'a' = 0`, so `1 << 0` → `0000001`
- `'b' - 'a' = 1`, so `1 << 1` → `0000010`
- `'c' - 'a' = 2`, so `1 << 2` → `0000100`

If `word = "bac"`, the **bitmask becomes:**

rust Copy Edit

```rust
'b' -> 0000010
'a' -> 0000011
'c' -> 0000111  (final mask)
```

cpp Copy Edit

```cpp
words = ["aba", "aabb", "abcd", "bac", "aabc"]
```

Word	Bitmask (Binary)	Value (Decimal)	`freq` Map Update	`count` Increase
"aba"	`00000011`	3	`{3: 1}`	`0`
"aabb"	`00000011`	3	`{3: 2}`	`1` (Found match with "aba")
"abcd"	`00001111`	15	`{3: 2, 15: 1}`	`1`
"bac"	`00000011`	3	`{3: 3, 15: 1}`	`2` (Found matches with "aba", "aabb")
"aabc"	`00001111`	15	`{3: 3, 15: 2}`	`2` (Found match with "abcd")

✅ **Final `count = 2`, which is correct!**

74) Number of consistent words :

```
class Solution {
public:
    int countConsistentStrings(string allowed, vector<string>& words) {
        int n = words.size();
        string word;
        int count = 0;
        vector<unordered_set<char>> freq(n);
        unordered_set<char> allowedSet(allowed.begin(), allowed.end());

        for(int i = 0; i < n; i++){
            word = words[i];
            bool isInconsistent = true;

            for(char it : word){
                if(allowedSet.find(it) == allowedSet.end()){
                    isInconsistent = false;
                    break;
                }
            }
            if(inConsistent == true) count++;
        }
        return count; } };
```

75) Split a string in balanced string(equal number of R and L)

```
class Solution {
public:
    int balancedStringSplit(string s) {
        int n = s.size();
        int count = 0;
        int balance = 0;

        for(int i = 0; i < n; i++){
            if(s[i] == 'R'){
                balance++;
```

```
        }
        else{
            balance--;
        }
        if(balance == 0) count++;
    }
    return count; } };
```

76) To lowercase :

```
class Solution {
public:
    string toLowerCase(string s) {
        for(int i = 0; i < s.size(); i++){
            if(s[i] >= 'A' && s[i] <= 'Z'){
                s[i] = s[i] + 32 ;
            }
        }
        return s; } };
```

77) Check balanced string(oddSum == evenSum) :

```
class Solution {
public:
    bool isBalanced(string num) {
        int oddSum = 0;
        int evenSum = 0;

        for(int i = 0; i < num.size(); i++){
            if((i+1) % 2 == 0){
                evenSum += num[i] - '0';
            }
            else{
                oddSum += num[i] - '0';
            }
        }
        return (oddSum == evenSum); } };
```

78) Sentence is pangram :

```
class Solution {
public:
    bool checkIfPangram(string sentence) {
        unordered_set<char> freq;

        for(int i = 0; i < sentence.size(); i++){
            freq.insert(sentence[i]);
        }
        if(freq.size() == 26) return true;    //set has unique elements
        return false; } };
```

79) Reverse words in a string :

```
class Solution {
public:
    string reverseWords(string s) {
        string word;
        string result;
        int end = 0;
        int start = 0;

        while(start < s.size()){
            while(s[end] != ' ' && end < s.size()){
                end++;
            }
            reverse(s.begin() + start, s.begin() + end);
            start = end + 1;
            end++;
        }
        return s; } };
```

Iteration 1 (Reversing `"Let's"`):

- `while (end < n && s[end] != ' ')` → Moves `end` until it hits a space.
- `end` stops at `5` (`s[5] = ' '`) → The word `"Let's"` is from `start = 0` to `end - 1 = 4`.
- Reverse `"Let's"` → `"s'teL"`
- Move `start` to `6` (after the space).
- Set `end = start` (so `end = 6` now).

80) Reverse string every 2k times :

```
class Solution {
public:
   string reverseStr(string s, int k) {
      int start;
      int end = start + k;
      int n = s.size();

      for(start = 0; start < n; start+= (2*k)){
         reverse(s.begin() + start, s.begin() + k);

      }

      return s; } };
```

`s = "abcdef", k = 4`

Initial State:

makefile Copy Edit

```
Index:   0  1  2  3  4  5
String:  a  b  c  d  e  f
```

First Iteration (`start = 0`)

cpp Copy Edit

```cpp
reverse(s.begin() + 0, s.begin() + min(0 + 4, 6));
```

Second Iteration (`start = 4`)

cpp Copy Edit

```cpp
reverse(s.begin() + 4, s.begin() + min(4 + 4, 6));
```

81) Sorting the sentence with number at each word's end :

```
class Solution {
public:
    string sortSentence(string s) {
        int n = s.size();
        vector<string> words(10);
        string word;
        string result;

        for(int i = 0; i < n; i++){
            if(s[i] == ' '){
                int pos = word.back() - '0';
                word.pop_back();
                words[pos] = word;
                word = "";
            }
            else{
                word += s[i];
            }
        }

        int pos = word.back() - '0';
```

```
word.pop_back();
words[pos] = word;

for(int i = 1; i < 10; i++){
    if(words[i] != ""){
        if(!result.empty()) result += " ";
        result += words[i];
    }
}

return result; } };
```

Input:

cpp Copy Edit

```cpp
s = "is2 sentence4 This1 a3"
```

If `word = "is2"`:

- `word.back()` is `'2'`, which we convert to an **integer** (`2`).
- We **remove the last digit** so that `word` becomes `"is"`.
- We store `"is"` at `words[2]`.

For every new word, the process repeats.

Processing Steps:

Step	`word`	Extracted Position	`words` Array
Read `"is2"`	`"is2"`	`2`	`words[2] = "is"`
Read `"sentence4"`	`"sentence4"`	`4`	`words[4] = "sentence"`
Read `"This1"`	`"This1"`	`1`	`words[1] = "This"`
Read `"a3"`	`"a3"`	`3`	`words[3] = "a"`

82) Partitioning Into Minimum Number Of Deci-Binary Numbers (Max - digit) :

```
class Solution {
public:
    int minPartitions(string n) {
        char maxDigit = '0';
        for(char digit : n){
            maxDigit = max(maxDigit, digit);
        }
        return maxDigit - '0'; } };
```

83) Minimum Number of Operations to Move All Balls to Each Box :

Brute Force :

```
class Solution {
public:
    vector<int> minOperations(string boxes) {
        int n = boxes.size();
        vector<int> op(n);
        int sum = 0;

        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
                if(boxes[j] == '1'){
                    sum += abs(j - i);   //Finding distance
                }
            }
            op[i] = sum;
            sum = 0;
        }
        return op; } };
```

Optimal :

Forward ad backward traverse : (two pointers) :

```
class Solution {
public:
    vector<int> minOperations(string boxes) {
        int n = boxes.size();
        vector<int> op(n);
        int sum = 0;
        int moves = 0;
        int ball = 0;

        for(int i = 0; i < n; i++){
            op[i] = moves + ball;
            moves += ball;
            ball += boxes[i] - '0';
        }
        ball = 0;
        moves = 0;

        for(int i = n-1; i >= 0; i--){
            op[i] += moves + ball;
            moves += ball;
            ball += boxes[i] - '0';
        }
        return op; } };
```

Example input : “110”

- `moves` is updated **before** `ball` in each iteration.

i	boxes[i]	`op[i]` (stored)	`moves` (updated before `ball`)	`ball` (updated after)
0	'1'	0	0	1
1	'1'	1	1	2
2	'0'	3	3	2

i	boxes[i]	`op[i]` (before update)	`op[i]` (after update)	`moves` (updated before `ball`)	`ball` (updated after)
2	'0'	3	3	0	0
1	'1'	1	1	0	1
0	'1'	0	1	1	2

84) Find words containing char :

```
class Solution {
public:
    vector<int> findWordsContaining(vector<string>& words, char x) {
        vector<int> op;
        for(int i = 0; i < words.size(); i++){
            string word = words[i];
            if(word.find(x) != string::npos){
                op.push_back(i);
            }
        }
        return op; } };
```

85) Defanging IP address :

3ms time :

```
class Solution {
public:
    string defangIPaddr(string address) {
        string op;

        for(int i = 0; i < address.size(); i++){
```

```
        if(address[i] == '.'){
            op += “[.]”;      // append() is faster than +=
        }
        else op += address[i];
    }
    return op; } };
```

0ms (Append):

```
class Solution {
public:
    string defangIPaddr(string address) {
        int n = address.size();
        string result;
        result.reserve(n + 6);

        for (char c : address) {
            if (c == '.') {
                result.append("[.]");
            } else {
                result.push_back(c);
            }
        }
        return result; } };
```

86) Jewels and stones :

```
class Solution {
public:
    int numJewelsInStones(string jewels, string stones) {
        int n = stones.size();
        int m = jewels.size();
        int count = 0;

        for(int i = 0; i < n; i++){
            for(int j = 0; j < m; j++){
                if(stones[i] == jewels[j])    count++;
            }
```

```
        }
        return count; } };
```

Using set(member function) :

```
#include <unordered_set>
class Solution {
public:
    int numJewelsInStones(string jewels, string stones) {
        unordered_set<char> jewelSet(jewels.begin(), jewels.end());
        int total = 0;

        for (char stone : stones) {
            if (jewelSet.count(stone))  total++;   //Count is a function
        }
        return total;
    }
};
```

cpp Copy Edit

```cpp
unordered_set<char> jewelSet = {'a', 'A'};
cout << jewelSet.count('a');  // Output: 1
cout << jewelSet.count('b');  // Output: 0
```

87) Convert Date to Binary :

```
class Solution {
public:
    string convertDateToBinary(string date) {
        int n = date.size();
        string word, op;

        for(int i = 0; i < n; i++) {
            if(date[i] == '-') {
                op.append(toBinary(word));
                op += '-';
                word.clear();
```

```
        } else {
            word += date[i];
        }
    }
    op.append(toBinary(word));

    return op; }

string toBinary(string word) {
    int num = stoi(word);
    string binaryStr;
         int dig = 0;
    while(num > 0) {
         dig = num % 2;
        binaryStr = char('0' + dig) + binaryStr;
        num /= 2;
    }
     if(binaryStr.empty()) return "0";
     else return binaryStr; } };
```

Convert 29 to Binary

1. 29 ÷ 2 = 14 → 1
2. 14 ÷ 2 = 7 → 0
3. 7 ÷ 2 = 3 → 1
4. 3 ÷ 2 = 1 → 1
5. 1 ÷ 2 = 0 → 1

vbnet Copy Edit

```vbnet
Input:      "2080-02-29"
             ├── 2080 → 100000100000
             ├── 02   → 10
             └── 29   → 11101
Output:    "100000100000-10-11101"
```

88) Score of a string :

```
class Solution {
public:
   int scoreOfString(string s) {
      int n = s.size();
      int op = 0;

      for(int i = 0; i < n -1; i++){
         op += abs(s[i] - s[i+1]);
      }

    return op; } };
```

Step 1 :

```
'h' = 104
'e' = 101
'l' = 108
'l' = 108
'o' = 111
```

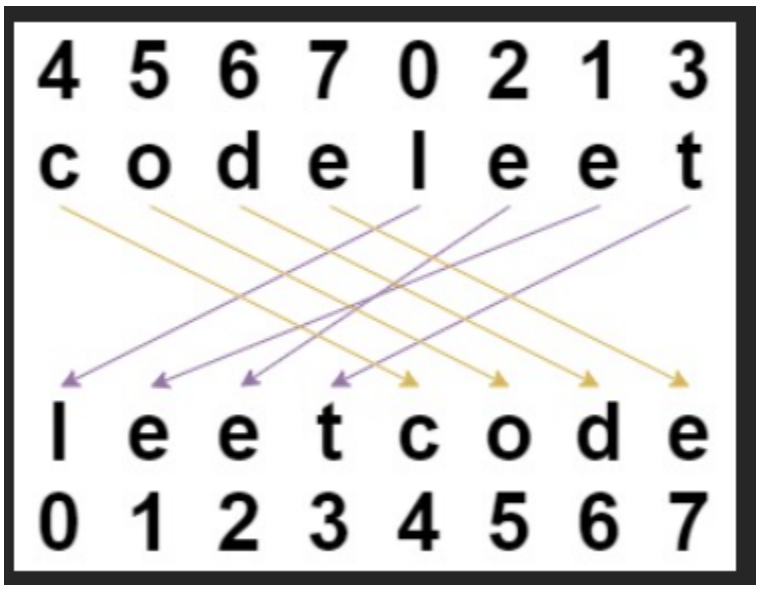
Step 2 :

```
|104 - 101| = 3
|101 - 108| = 7
|108 - 108| = 0
|108 - 111| = 3
```

Step 3 :

```
3 + 7 + 0 + 3 = 13
```

89) Shuffle String :



Brute Force solution :

```
class Solution {
public:
    string restoreString(string s, vector<int>& indices) {
        int n = s.size();
        string op = "";

        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){
```

```
            if(i == indices[j])  op += s[j];
        }
    }

    return op; } };
```

Optimal Solution : (Using map)

```
class Solution {
public:
    string restoreString(string s, vector<int>& indices) {
        int n = s.size();
        string op = "";
        map<int, char> charWithIndices;

        for(int i = 0; i < n; i++){
            charWithIndices[indices[i]] = s[i];
        }
        for(int i = 0; i < n; i++){
            op += charWithIndices[i];
        }
        return op; } };
```

More optimal(Without extra space) :

```
class Solution {
public:
    string restoreString(string s, vector<int>& indices) {
        int n = s.size();
        string op(n, ' ');

        for(int i = 0; i < n; i++){
            op[indices[i]] = s[i];
        }
        return op; } };
```

```ini
op = "     "   // (5 empty spaces)
```

i (index in `s`)	s[i]	indices[i]	Resulting `op`
0	'a'	3	" a "
1	'i'	1	" i a "
2	'o'	4	" i a o"
3	'h'	2	" ihao"
4	'n'	0	"niaho"

90) Check if two strings are equivalent :

```
class Solution {
public:
   bool arrayStringsAreEqual(vector<string>& word1, vector<string>&
word2) {
      int n1 = word1.size();
      int n2 = word2.size();

      string words1, words2;
      for(int i = 0; i < n1; i++){
         words1.append(word1[i]);
      }

      for(int i = 0; i < n2; i++){
         words2.append(word2[i]);
      }
      return (words1 == words2); } };
```

91) Truncate sentence :

```
class Solution {
public:
   string truncateSentence(string s, int k) {
      int n = s.size();
```

```
        string op;
        int space = 0;

        for(int i = 0; i < n; i++){
            if(s[i] == ' ') space++;
            if(space == k) return op;
            op += s[i];
        }
        return op; } };
```

92) Lemonade Change :

With using map:

```
class Solution {
public:
    bool lemonadeChange(vector<int>& bills) {
        int n = bills.size();
        map<int, int> change;
        int price = 5;
        int changeNeeded;

        for (int i = 0; i < n; i++) {
            if (bills[i] > 5) {
                changeNeeded = bills[i] - price;

                while (changeNeeded > 0) {
                    if (changeNeeded >= 20 && change[20] > 0) {
                        change[20]--;
                        changeNeeded -= 20;
                    }
                    else if (changeNeeded >= 10 && change[10] > 0) {
                        change[10]--;
                        changeNeeded -= 10;
                    } else if (changeNeeded >= 5 && change[5] > 0) {
                        change[5]--;
```

```
                changeNeeded -= 5;
            } else {
                return false;
            }
        }

    }
        change[bills[i]]++;


    }
    return true; } };
```

Customer Pays	Change Needed	Available Change	Action Taken	Change Map After Transaction
$5	$0	`{}`	Accept $5	`{5:1}`
$5	$0	`{5:1}`	Accept $5	`{5:2}`
$10	$5	`{5:2}`	Give $5 as change	`{5:1, 10:1}`
$10	$5	`{5:1, 10:1}`	Give $5 as change	`{5:0, 10:2}`
$20	$15	`{5:0, 10:2}`	❌ No $5 available → Return false	❌

Output:

cpp Copy Edit

```cpp
false
```

Without Using map class Solution {

```
public:
    bool lemonadeChange(vector<int>& bills) {
        int five = 0, ten = 0;

        for (int bill : bills) {
```

```
        if (bill == 5) five++;
        else if (bill == 10) {
            if (five == 0) return false;
            five--; ten++;
        }
        else {
            if (ten > 0 && five > 0) {
                ten--; five--;
            } else if (five >= 3) {
                five -= 3;
            } else return false;
        }
    }
    return true; } };
```

93) Concatenation of array :

```
class Solution {
public:
    vector<int> getConcatenation(vector<int>& nums) {
        int n = nums.size();
        vector<int> op(2*n);
        for(int i = 0; i < nums.size(); i++){
            op[i] = nums[i];
            op[n + i] = nums[i];
        }
        return op; } };
```

94) Number of good pairs :

```
class Solution {
public:
    int numIdenticalPairs(vector<int>& nums) {
        int n = nums.size();
        int count = 0;

        for(int i = 0; i < n; i++){
```

```
        for(int j = i+1; j < n; j++){
            if(nums[i] == nums[j]) count++;
        }
    }
    return count; } };
```

95) Find Minimum Operations to Make All Elements Divisible by Three :

```
class Solution {
public:
    int minimumOperations(vector<int>& nums) {
        int n = nums.size();
        int count = 0;

        for(int i = 0; i < n; i++){
            if(nums[i] % 3 != 0){
                count++;
            }
        }
        return count; } };
```

96) Max sum of the subarray :

```
class Solution {
public:
    int maximumWealth(vector<vector<int>>& accounts) {
        vector<int> wealth;
        int n = accounts.size();
        int richest = 0;
        int sum = 0;

        for(int i = 0; i < n; i++){
            for(int j = 0; j < accounts[i].size(); j++){
                sum += accounts[i][j];
            }
            richest = max(richest, sum);
            sum = 0;
```

```
        }

        return richest; } };
```

97) Reverse words in a string :

Using vector :

```
class Solution {
public:
    string reverseWords(string s) {
        string word = "";
        string op = "";
        int n = s.size();
        int ptr = 0;
        vector<string> tempStr;

        for(int i = 0; i < n; i++){
            if((s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <= 'Z') || (s[i] >= '0' &&
s[i] <= '9')){
                word += s[i];
            }
            else{
                if(word != ""){
                    tempStr.push_back(word);
                }
                word = "";
            }
        }
        if(word != ""){
            tempStr.push_back(word);
        }
        word = "";

        int wordCount = tempStr.size();
        reverse(tempStr.begin(), tempStr.end());
        for (int i = 0; i < wordCount; ++i) {
```

```
        op.append(tempStr[i]);
        if (i < wordCount - 1) {
            op += " ";
        }
    }

    return op; } };
```

Without extra space (without vector) :

```
class Solution {
public:
    string reverseWords(string s) {
        int n = s.size();
        int left = 0, right = n - 1;

        while (left <= right && s[left] == ' ') left++;
        while (right >= left && s[right] == ' ') right--;

        reverse(s.begin() + left, s.begin() + right + 1);

        int wordStart = left;
        for (int i = left; i <= right; i++) {
            if (s[i] == ' ' || i == right) {
                if (i == right) i++;
                reverse(s.begin() + wordStart, s.begin() + i);
                wordStart = i + 1;
            }
        }

        int i = left, j = left;
        while (i <= right) {
            while (i <= right && s[i] == ' ') i++;
            while (i <= right && s[i] != ' ') s[j++] = s[i++];
            while (i <= right && s[i] == ' ') i++;
            if (i <= right) s[j++] = ' ';
```

```
    }

    return s.substr(left, j - left); } };
```

Step-by-Step Visualization

Step	Operation	String Transformation
1	Trim spaces	`"hello world"`
2	Reverse whole string	`"dlrow olleh"`
3	Reverse each word	`"world hello"`
4	Remove extra spaces	`"world hello"`

Step 1: Trim Leading & Trailing Spaces

📌 **Initial Input:**

sql Copy Edit

```sql
"   hello   world   "
   ↑               ↑
  left           right
```

Step 2: Reverse the Entire Trimmed String

We reverse the whole string in-place.

📌 **Before Reverse:**

arduino Copy Edit

```arduino
"hello   world"
```

📌 **After Reverse:**

arduino Copy Edit

```arduino
"dlrow   olleh"
```

Reverse each words alone :

arduino Copy Edit

```
"world   hello"
```

Step 4: Remove Extra Spaces

Before Removing Spaces:

arduino Copy Edit

```
"world   hello"
```

➡️ **Collapse multiple spaces into one:**

arduino Copy Edit

```
"world hello"
```

98) Largest odd number in string :

```
class Solution {
public:
    string largestOddNumber(string num) {
        int n = num.size();

        for(int i = n-1; i >= 0; i--){
            int currentNum = (num[i] - '0');
            if(currentNum % 2 == 1){
                return num.substr(0, i + 1);
            }
        }

        return ""; } };
```

99) Isomorphic string (mapping two str) : Using map (4ms)

```
class Solution {
public:
    bool isIsomorphic(string s, string t) {
```

```
        int n1 = s.size();
        int n2 = t.size();

        if(n1 != n2) return false;
        unordered_map<char, char> mpp1;
        unordered_map<char, char> mpp2;

        for(int i = 0; i < n1; i++){
            if (mpp1.count(s[i]) && mpp1[s[i]] != t[i]) return false;
            else mpp1[s[i]] = t[i];
            if (mpp2.count(t[i]) && mpp2[t[i]] != s[i]) return false;
            else mpp2[t[i]] = s[i];
        }
        return true; } };
```

With char ASCII (0ms) :

```
class Solution {
public:
    bool isIsomorphic(string s, string t) {
        if (s.size() != t.size()) return false;

        char sMap[256] = {0};
        char tMap[256] = {0};

        for (int i = 0; i < s.size(); i++) {
            if (sMap[s[i]] == 0 && tMap[t[i]] == 0) {
                sMap[s[i]] = t[i];
                tMap[t[i]] = s[i];
            }
            else if (sMap[s[i]] != t[i] || tMap[t[i]] != s[i]) {
                return false;
            }
        }
        return true; } };
```

We iterate through each character of `s` and `t` simultaneously.

Index (i)	s[i]	t[i]	sMap (s → t)	tMap (t → s)	Check Condition
0	'e'	'a'	`sMap['e'] = 'a'`	`tMap['a'] = 'e'`	No conflict, store mapping
1	'g'	'd'	`sMap['g'] = 'd'`	`tMap['d'] = 'g'`	No conflict, store mapping
2	'g'	'd'	**Already mapped**	**Already mapped**	✅ No conflict, continue

All mappings are valid, so we return **true** ✅.

cpp Copy Edit

```cpp
s = "foo"
t = "bar"
```

Index (i)	s[i]	t[i]	sMap (s → t)	tMap (t → s)	Check Condition
0	'f'	'b'	`sMap['f'] = 'b'`	`tMap['b'] = 'f'`	No conflict, store mapping
1	'o'	'a'	`sMap['o'] = 'a'`	`tMap['a'] = 'o'`	No conflict, store mapping
2	'o'	'r'	**sMap['o'] is 'a' but found 'r'** ❌	**Mismatch, return false!**	

Here, `o` was mapped to `a`, but now it's conflicting with `r`, so we return **false** ❌.

100) Count pair whose sum < target :

Brute force :

```
class Solution {
public:
    int countPairs(vector<int>& nums, int target) {
        int n = nums.size();
        int count = 0;

        for(int i = 0; i < n; i++){
            for(int j = i + 1; j < n; j++){
                if((nums[i] + nums[j]) < target) count++;
            }
        }
```

```
        return count; } };
```

Using 2 pointer :

```
class Solution {
public:
    int countPairs(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());
        int n = nums.size();
        int count = 0;
        int left = 0, right = n -1;

        while(left < right){
            if((nums[left] + nums[right]) < target){
                count += (right - left);
                left++;
            }
            else{
                right--;
            }
        }
        return count; } };
```

Given Input:

cpp

```cpp
nums = [2, 3, 1, 5, 4], target = 6
```

Step	Left Pointer (`nums[left]`)	Right Pointer (`nums[right]`)	Sum (`nums[left] + nums[right]`)	Action
1	`1` (index 0)	`5` (index 4)	`1 + 5 = 6` (Not `< 6`)	Move `right--` (now at index 3)
2	`1` (index 0)	`4` (index 3)	`1 + 4 = 5` (< 6 ✅)	`count += (3 - 0) = 3`, move `left++`
3	`2` (index 1)	`4` (index 3)	`2 + 4 = 6` (Not `< 6`)	Move `right--` (now at index 2)
4	`2` (index 1)	`3` (index 2)	`2 + 3 = 5` (< 6 ✅)	`count += (2 - 1) = 1`, move `left++`
End	`left == right`	Stop		

How `count += (right - left)` Works Visually

At Step 2, when `nums[left] = 1` and `nums[right] = 4`:

```css
Sorted: [1, 2, 3, 4, 5]
          ↑         ↑
       left(0)   right(3)
```

Since `1 + 4 = 5` (which is **less than** `target`),

- **All pairs** `(1,2)`, `(1,3)`, `(1,4)` are valid.
- Since indices 1, 2, 3 are **between** `left` **and** `right`, we **directly add** `(right - left) = 3` **to count**.

101) Find the Prefix Common Array of Two Arrays: Using vector (6ms) :

```cpp
class Solution {
public:
    vector<int> findThePrefixCommonArray(vector<int>& A, vector<int>& B)
{
```

```
        int n1 = A.size();
        int n2 = B.size();
        vector<int> freq(51, 0);
        vector<int> op(n1, 0);
        int count = 0;

        for(int i = 0; i < n1; i++){
            freq[A[i]]++;
            if(freq[A[i]] == 2) count++;
            freq[B[i]]++;
            if(freq[B[i]] == 2) count++;
            op[i] = count;
        }

        return op; } };
```

Bitset Array :

```
class Solution {
public:
    vector<int> findThePrefixCommonArray(vector<int>& A, vector<int>& B)
{
        int n = A.size();
        bitset<51> seen;
        int freq[51] = {0};
        vector<int> op(n);
        int count = 0;

        for (int i = 0; i < n; i++) {
            freq[A[i]]++;
            if (freq[A[i]] == 2) count++;
            freq[B[i]]++;
            if (freq[B[i]] == 2) count++;

            op[i] = count;
        }
```

return op; } } ;

```
A = [1, 3, 2, 4]
B = [3, 1, 2, 4]
```

```
A[0] = 1, B[0] = 3
```

1. `freq[1]++` → `freq[1] = 1`
2. `freq[3]++` → `freq[3] = 1`
3. No number has appeared **twice**, so `count = 0`
4. `op[0] = 0`

```
A[1] = 3, B[1] = 1
```

1. `freq[3]++` → `freq[3] = 2` (✅ Appeared twice, `count++`)
2. `freq[1]++` → `freq[1] = 2` (✅ Appeared twice, `count++`)
3. `count = 2`
4. `op[1] = 2`

State :

```
freq = [0, 2, 0, 2, 0, ...]
op = [0, 2, _, _]
count = 2
```

Step-by-Step Execution Table

Iteration (i)	A[i]	B[i]	freq[A[i]] Before	freq[A[i]] After	freq[B[i]] Before	freq[B[i]] After	Count Before	Count After	op[i]
0	1	3	0	1	0	1	0	0	0
1	3	1	1	2 (✅)	1	2 (✅)	0	2	2
2	2	2	0	1	1	2 (✅)	2	3	3
3	4	4	0	1	1	2 (✅)	3	4	4

102) Find Original array of prefix XOR :

```
class Solution {
public:
    vector<int> findArray(vector<int>& pref) {
        int n = pref.size();
        vector<int> op(n);
        op[0] = pref[0];

        for(int i = 1; i < n; i++){
            op[i] = pref[i-1] ^ pref[i];
        }
        return op; } };
```

103) Remove outermost parentheses :

```
class Solution {
public:
    string removeOuterParentheses(string s) {
        int n = s.size();
        string op = "";
        int balance = 0;
        int fp = 0;
```

```
        for(int i = 0; i < n; i++){
            if(s[i] == '(') balance++;
            else if(s[i] == ')') balance--;

            if(balance == 0){
                op.append(s.substr(fp + 1, (i - fp - 1)));
                fp = i + 1;
            }
        }
        return op; } };
```

104) Rotate String :

```
class Solution {
public:
    bool rotateString(string s, string goal) {
        int n = s.size();

        for(int i = 0; i < n; i++){
            char temp = s[0];
            s = s.substr(1) + s[0];

            if(s == goal) return true;
        }
        return false; } };
```

105) Sort characters by frequency :

```
class Solution {
public:
    string frequencySort(string s) {
        int n = s.size();
        map<char, int> freq;
        string op = "";

        for(int i = 0; i < n; i++){
            freq[s[i]]++;
        }
```

```
    vector<pair<char, int>> vec(freq.begin(), freq.end());

    sort(vec.begin(), vec.end(), [](const pair<char, int> &a, const
pair<char, int> &b){
        return a.second > b.second;
    });

    for (int i = 0; i < vec.size(); i++) {
        while(vec[i].second > 0){
            op += (vec[i].first);
            vec[i].second--;
        }
    }

    return op; } };
```

Character	Count
't'	1
'r'	1
'e'	2

cpp Copy Edit

```cpp
freq = { {'t', 1}, {'r', 1}, {'e', 2} };
```

Before Sorting:

arduino Copy Edit

```arduino
{ 't', 1 }, { 'r', 1 }, { 'e', 2 }
```

After Sorting (Descending by Frequency):

arduino Copy Edit

```arduino
{ 'e', 2 }, { 't', 1 }, { 'r', 1 }
```

Final Code Execution Flow (Diagram)

rust

Copy Edit

```
Input: "tree"

1 Frequency Count:
  { 't' -> 1, 'r' -> 1, 'e' -> 2 }

2 Store in Vector:
  [ ('t', 1), ('r', 1), ('e', 2) ]

3 Sort by Frequency (Descending):
  [ ('e', 2), ('t', 1), ('r', 1) ]

4 Build Output String:
  "eert"
```

106) Sum of beauty of all substrings :

```
class Solution {
public:
    int beautySum(string s) {
        int n = s.size();
        int opCount = 0;

        for(int i = 0; i < n; i++){
            vector<int> freq(26, 0);
            for(int j = i; j < n; j++){
                freq[s[j] - 'a']++;
                int maxFreq = 0, minFreq = INT_MAX;

                for(int k = 0; k < 26; k++){
                    if(freq[k] > 0){
                        maxFreq = max(maxFreq, freq[k]);
                        minFreq = min(minFreq, freq[k]);
                    }
                }
```

```
        opCount += (maxFreq - minFreq);
    }
}


return opCount; } };
```

We extract **all possible substrings** from `s = "aabcb"`.

Starting Index (`i`)	Substrings (Expanding `j`)
`i = 0` → `"a"`	`"a"`, `"aa"`, `"aab"`, `"aabc"`, `"aabcb"`
`i = 1` → `"a"`	`"a"`, `"ab"`, `"abc"`, `"abcb"`
`i = 2` → `"b"`	`"b"`, `"bc"`, `"bcb"`
`i = 3` → `"c"`	`"c"`, `"cb"`
`i = 4` → `"b"`	`"b"`

Substrings from `i = 0` (Starting from `"a"`)

1. `"a"` → { `a`:1} → **Beauty = 0**
2. `"aa"` → { `a`:2} → **Beauty = 0**
3. `"aab"` → { `a`:2, `b`:1} → **Beauty = 2 - 1 = 1** ✅
4. `"aabc"` → { `a`:2, `b`:1, `c`:1} → **Beauty = 2 - 1 = 1** ✅
5. `"aabcb"` → { `a`:2, `b`:2, `c`:1} → **Beauty = 2 - 1 = 1** ✅

Substrings from `i = 1` (Starting from `"a"`)

1. `"a"` → { `a`:1} → **Beauty = 0**
2. `"ab"` → { `a`:1, `b`:1} → **Beauty = 1 - 1 = 0**
3. `"abc"` → { `a`:1, `b`:1, `c`:1} → **Beauty = 1 - 1 = 0**
4. `"abcb"` → { `a`:1, `b`:2, `c`:1} → **Beauty = 2 - 1 = 1** ✅

Continues, i = 2, 3, 4

107) Permutation difference between two strings :

```
class Solution {
public:
    int findPermutationDifference(string s, string t) {
        int n = s.size();
        int count = 0;
        unordered_map<char, int> freq;

        for(int i = 0; i < n; i++){
            freq[t[i]] = i;
        }
        for(int i = 0; i < n; i ++){
            count += abs(i - freq[s[i]]);
        }

        return count;
    }
};
```

📌 **Index Differences:**

java Copy Edit

```
s[0] = 'a' → t index = 2 → |0 - 2| = 2
s[1] = 'b' → t index = 0 → |1 - 0| = 1
s[2] = 'c' → t index = 1 → |2 - 1| = 1
```

108) How many no smaller than current index element :

```
class Solution {
public:
    vector<int> smallerNumbersThanCurrent(vector<int>& nums) {
        int n = nums.size();
        vector<int> sortedNums = nums;
        sort(sortedNums.begin(), sortedNums.end());
        unordered_map<int, int> countMap;

        for(int i = 0; i < n; i++){
            if(countMap.find(sortedNums[i]) ==  countMap.end()){
```

```
        countMap[sortedNums[i]] = i;
    }
}

vector<int> result;
for(int num : nums){
    result.push_back(countMap[num]);
}
return result; } };
```

```cpp
sortedNums = {1, 2, 2, 3, 8};
```

`sortedNums[i]`	Index `i`	Meaning
1	0	0 numbers are smaller
2	1	1 number is smaller
2	2	(Already seen `2`, so keep previous index `1`)
3	3	3 numbers are smaller
8	4	4 numbers are smaller

cpp

```cpp
nums = {8, 1, 2, 2, 3}
```

Using `countMap`:

- `8 → 4`
- `1 → 0`
- `2 → 1`
- `2 → 1`
- `3 → 3`

Final Output

cpp

```cpp
result = {4, 0, 1, 1, 3}
```

109) Find indices of stable mountain :

```
class Solution {
public:
    vector<int> stableMountains(vector<int>& height, int threshold) {
        int n = height.size();
        vector<int> indices;

        for(int i = 1; i < n; i++){
            if(height[i-1] > threshold){
                indices.push_back(i);
            }
        }
        return indices; } };
```

110) Running sum of 1D Array :

```
class Solution {
public:
    vector<int> runningSum(vector<int>& nums) {
        int n = nums.size();
        int sum = 0;
        vector<int> op(n);

        for(int i = 0; i < n; i++){
            sum += nums[i];
            op[i] = sum;
        }
        return op; } };
```

111) Find common elements bw two arrays :

```
class Solution {
public:
    vector<int> findIntersectionValues(vector<int>& nums1, vector<int>&
nums2) {

        unordered_set<int> set1(nums1.begin(), nums1.end());
        unordered_set<int> set2(nums2.begin(), nums2.end());
```

```
        int count1 = 0, count2 = 0;

        for(int num : nums1){
            if(set2.find(num) != set2.end()) count1++;
        }

        for(int num : nums2){
            if(set1.find(num) != set1.end()) count2++;
        }

        return {count1, count2}; } };
```

112) Count number of pairs with absolute difference K :

```
class Solution {
public:
    int countKDifference(vector<int>& nums, int k) {
        int n = nums.size();
        vector<int> freq(101, 0);
        int count = 0;

        for(int num : nums){
            freq[num]++;
        }

        for(int i = 0; i < freq.size() - k; i++){
            if(freq[i] > 0 && freq[i + k] > 0){
                count += (freq[i] * freq[i + k]);
            }
        }

        return count; } };
```

113) Sort minimum (2nd min first, first min 2nd) :

```
class Solution {
public:
```

```
    vector<int> numberGame(vector<int>& nums) {
        int n = nums.size();
        sort(nums.begin(), nums.end());

        for(int i = 0; i < n; i += 2){
            int temp = nums[i];
            nums[i] = nums[i+1];
            nums[i+1] = temp;
        }
        return nums; } };
```

114) 2161. Partition Array According to Given Pivot

```
class Solution {
public:
    vector<int> pivotArray(vector<int>& nums, int pivot) {
        int n = nums.size();
        vector<int> beforePivot;
        vector<int> afterPivot;
        vector<int> res;
        int pivotCount = 0;

        for(int i = 0; i < n; i++){
            if(nums[i] < pivot){
                beforePivot.push_back(nums[i]);
            }
            else if(nums[i] > pivot){
                afterPivot.push_back(nums[i]);
            }
            else{
                pivotCount++;
            }
        }
        for(int i = 0; i < beforePivot.size(); i++){
            res.push_back(beforePivot[i]);
        }
```

```
        while(pivotCount > 0){
            res.push_back(pivot);
            pivotCount--;
        }
        for(int i = 0; i < afterPivot.size(); i++){
            res.push_back(afterPivot[i]);
        }
        return res; } };
```

115) First bad version :

```
class Solution {
public:
    int firstBadVersion(int n) {
        int left = 1, right = n;

        while(left < right){
            int mid = left + (right - left) /2;

            if(isBadVersion(mid)){
                right = mid;
            }
            else{
                left = mid + 1;
            }
        }
        return left;  } };
```

116) Sum of powers of three :

```
class Solution {
public:
    bool checkPowersOfThree(int n) {

        while(n > 0){
            if(n % 3 == 2) return false;
            n /= 3;
        }       return true; } };
```

117) Convert temperature :

```
class Solution {
public:
    vector<double> convertTemperature(double celsius) {
        double kelvin = 0.00000;
        double fahrenheit = 0.00000;

        kelvin = celsius + 273.15;
        fahrenheit = (celsius * 9.00 / 5.00) + 32.00;

        return {kelvin, fahrenheit}; } };
```

118) Strictly Palindromic :

```
class Solution {
public:
    bool isStrictlyPalindromic(int n) {
        return false;
    }
};
```

😭

```
class Solution {
public:
    bool isStrictlyPalindromic(int n) {

        for(int i = 2; i <= n-2; i++){
            string binaryFormat = "";

            int copy = n;
            while(copy > 0){
                int dig = copy % i;
                binaryFormat += to_string(dig);
                copy /= i;
            }
            string reversedFormat = binaryFormat;
            reverse(binaryFormat.begin(), binaryFormat.end());
```

```
        if(reversedFormat != binaryFormat) return false;
    }

    return true; } };
```

119) Count Number of Distinct Integers After Reverse Operations :

```
class Solution {
public:
  int countDistinctIntegers(vector<int>& nums) {
    int n = nums.size();
    vector<int> reversedArray = nums;
    int count = 0;

    for(int i = 0; i < n; i++){
        int copy = nums[i];
        int rev = 0;
        while(copy > 0){
            int dig = copy % 10;
            rev = (rev*10)+ dig;
            copy /= 10;
        }
        reversedArray.push_back(rev);
    }

    unordered_set<int> revHash;

    for(int i = 0; i < reversedArray.size(); i++){
        revHash.insert(reversedArray[i]);
    }

    count = revHash.size();

    return count; } };
```

Another method ;

```
class Solution {
public:
    int countDistinctIntegers(vector<int>& nums) {
        int n = nums.size();
        int count = 0;
        unordered_set<int> revHash;

        for(int num : nums){
            revHash.insert(num);
        }

        for(int i = 0; i < n; i++){
            int copy = nums[i];
            int rev = 0;
            while(copy > 0){
                rev = (rev*10) + (copy % 10);
                copy /= 10;
            }
            revHash.insert(rev);
        }
        return revHash.size(); } };
```

120) Find triangular sum of an array :

```
class Solution {
public:
    int triangularSum(vector<int>& nums) {
        int n = nums.size();
        if(n == 1) return nums[0];

        while(n > 1){
            for(int i = 0; i < n - 1; i++){
                nums[i] = (nums[i] + nums[i+1]) % 10;
            }
            n--;
        }
```

```
        return nums[0]; } };
```

121) Min operations to make an array equal :

```
class Solution {
public:
    int minOperations(int n) {
        int steps = 0;
        vector<int> arr;
        for(int i = 0; i < n; i++){
            arr.push_back((2 * i) + 1);
        }
        int median = arr[n/2];
        for(int i = 0; i < n; i++){
            if(arr[i] < median){
                steps += (median - arr[i]);
            }
            else{
                steps -= (median - arr[i]);
            }

        }
        return steps / 2; } };
```

122) Minimum Number of Operations to Reinitialize a Permutation :

```
class Solution {
public:
    int reinitializePermutation(int n) {
        int pos = 1;
        int count = 0;

        do {
            if (pos % 2 == 0) {
                pos /= 2;
            } else {
                pos = n / 2 + (pos - 1) / 2;
```

```
        }
        count++;
    } while (pos != 1);

    return count; } };
```

Iteration	Current `pos`	Calculation	New `pos`	Operations Count
1	1 (odd)	`n/2 + (1-1)/2 = 3`	3	1
2	3 (odd)	`n/2 + (3-1)/2 = 4`	4	2
3	4 (even)	`4/2 = 2`	2	3
4	2 (even)	`2/2 = 1`	1 (reset)	4

123) Minimum Average of Smallest and Largest Elements :

```
#include <bits/stdc++.h>
class Solution {
public:
    double minimumAverage(vector<int>& nums) {
        int n = nums.size();
        vector<double> op;

        sort(nums.begin(), nums.end());
        int left = 0;
        int right = n-1;
        while(left < right){
            op.push_back((nums[left] + nums[right]) / 2.0);
            left++;
            right--;
        }
        double mini = numeric_limits<double>::max();
        for(double i : op){
            mini = min(mini, i);
        }

        return mini; } };
```

124) Max no of coins you can get :

```
class Solution {
public:
    int maxCoins(vector<int>& piles) {
        int max = 0;
        int n = piles.size();

        for(int i : piles){
            if(i > max) max = i;
        }
        vector<int> freq(max + 1, 0);
        for (int i : piles) {
            freq[i]++;
        }

        int chance = n / 3;
        int coins =0;
        int turn = 1;
        int i = max;

        while(chance != 0){
            if(freq[i] > 0){
                if(turn == 1) turn = 0;
                else{
                    chance--;
                    turn = 1;
                    coins += i;
                }
                freq[i]--;
            }
            Else i–;
        }

        return coins; } };
```

125) Sort 2D matrix with k :

```
class Solution {
public:
    vector<vector<int>> sortTheStudents(vector<vector<int>>& score, int k)
{
        int n = score.size();

        sort(score.begin(), score.end(), [k](const vector<int> &a, const
vector<int> &b){
            return a[k] > b[k];
        });

        return score; } };
```

126) Min sum of 4 digit after splitting :

```
#include <bits/stdc++.h>
class Solution {
public:
    int minimumSum(int num) {
        string nums = to_string(num);
        string temp = "";
        int sum = 0;

        sort(nums.begin(), nums.end());
        int n = nums.size();

        for(int i = 0; i < n - 2; i++){
            temp = nums[i];
            temp += nums[i+2];
            sum += stoi(temp);
        }

        return sum; } };
```

127) Widest Vertical Area Between Two Points Containing No Points :

```
class Solution {
public:
   int maxWidthOfVerticalArea(vector<vector<int>>& points) {
      int n = points.size();
      int maxGap = 0;
      int dist = 0;
      vector<int> xCordinates;

      sort(points.begin(), points.end(), [](const vector<int>& a, const
vector<int>& b) {
         return a[0] < b[0];  // Compare by x-coordinate (first element)
      });

      for(int i = 1; i < n; i++){
         dist = points[i][0] - points[i - 1][0];
         maxGap = max(maxGap, dist);
      }
      return maxGap; } };
```

128) Sort vowels in a string :

```
class Solution {
public:
   string sortVowels(string s) {
      string t = s;
      int n = s.size();
      string vowels;

      for(int i = 0; i < n; i++){
         if(s[i] == 'a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'o' || s[i] == 'u' ||
           s[i] == 'A' || s[i] == 'E' || s[i] == 'I' || s[i] == 'O' || s[i] == 'U') {
            t[i] = '-';
            vowels += s[i];
```

```
        }
    }
    sort(vowels.begin(), vowels.end());
    int index = 0;
    for(int i = 0; i < n; i++){
        if(t[i] == '-'){
            t[i] = vowels[index];
            index++;
        }
    }
    return t; } };
```

Using set :

```
class Solution {
public:
    string sortVowels(string s) {
        int n = s.size();
        string vowels;
        vowels.reserve(n);
        unordered_set<char> vowelSet = {'a','e','i','o','u','A','E','I','O','U'};

        for(char c : s){
            if(vowelSet.count(c)){
                vowels += c;
            }
        }
        if(vowels.empty()) return s;
        sort(vowels.begin(), vowels.end());
        int index = 0;
        for(int i = 0; i < n; i++){
            if(vowelSet.count(s[i])){
                s[i] = vowels[index++];
            }
        }
        return s; } };
```