LEETCODE STL+BITS

Q1) https://leetcode.com/problems/intersection-of-two-arrays/description/

```
class Solution {
public:
    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {
        int n=nums2.size();
        unordered_set<int>s2;
        unordered_set<int>s1(nums1.begin(),nums1.end());
        for(int i=0;i<n;i++){
            if(s1.find(nums2[i])!=s1.end()){
                s2.insert(nums2[i]);
            }
        }
        return vector<int>(s2.begin(),s2.end());
    }
};
```

Q2) https://leetcode.com/problems/subarray-sum-equals-k/

```
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        int result = 0;
        int sum = 0;
        map<int, int> mp;
        mp.insert({0,1});
        int n = nums.size();
        for(int i = 0; i<n; i++) {
            sum += nums[i];

            if(mp.find(sum-k)!=mp.end())
                result += mp[sum-k];

            mp[sum]++;
        }
        return result;
    }
};
```

Q3) https://leetcode.com/problems/longest-consecutive-sequence/solutions/

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        if(nums.empty()) {
            return 0;
        }
        int n=nums.size();
        int count,maxi=1;
        unordered_set<int>s(nums.begin(),nums.end());
        for(int i=0;i<n;i++){
            if(s.find(nums[i]-1)==s.end()){
                count=1;
                while(s.find(nums[i]+count)!=s.end()){
                    count++;
                    maxi=max(maxi,count);
                }
            }
        }
        return maxi;
    }
};
```

Q4) https://leetcode.com/problems/contiguous-array/

```
class Solution {
public:
    int findMaxLength(vector<int>& nums) {
        int n=nums.size();
        //all the zeros are converted to -1
        for(int i=0;i<n;i++){
            if(nums[i]==0){
                nums[i]=-1;
            }
        }
        //now it becomes largest subarray sum question
        //here we we use the prefix sum concept
        int res=0,sum=0,resi;
        unordered_map<int,int>m;
        for(int i=0;i<n;i++){
            sum=sum+nums[i];
            if(sum==0){
                resi=i+1;
                res=max(res,resi);
                continue;
            }
            if(m.find(sum)!=m.end()){
                resi=i-m[sum];
                res=max(res,resi);
            }else{
                m.insert({sum,i});
            }
        }
    return res;
    }
};
```

Q5) https://leetcode.com/problems/maximum-sum-of-distinct-subarrays-with-length-k/

```
class Solution {
public:
    long long maximumSubarraySum(vector<int>& a, int k) {
        long long sum = 0, maxSum = 0;
        int i = 0;
        unordered_set<int> s;

        for (int j = 0; j < a.size(); j++) {
            // Shrink window if necessary
            while (s.count(a[j]) || s.size() == k) {
                sum -= a[i];
                s.erase(a[i]);
                i++;
            }

            // Expand window
            sum += a[j];
            s.insert(a[j]);

            // Update maxSum if we have a valid window
        if (s.size() == k)
            maxSum = max(maxSum, sum);
    }
    return maxSum;
  }
};
```

Q6) https://leetcode.com/problems/find-players-with-zero-or-one-losses/description/

```
class Solution {
public:
  vector<vector<int>>
findWinners(vector<vector<int>>& matches) {
    unordered_map<int, int> lost;

    for(auto &it : matches) {
      int lose = it[1];
      lost[lose]++;
    }

    vector<int> notLost;
    vector<int> oneLos;

    for(auto &it : matches) {
      int lose = it[1];
      int win  = it[0];

      if(lost[lose] == 1) {
        oneLos.push_back(lose);
      }
      if(lost.find(win) == lost.end()) {
        notLost.push_back(win);
        lost[win] = 2;//AVOID duplicates
      }
    }

    sort(begin(notLost), end(notLost));
    sort(begin(oneLos), end(oneLos));

    return {notLost, oneLos};

  }
};
```

Q7) **sort characters by frequency**

```
class Solution {
public:
 static bool compare(pair<int,
char>&a,pair<int, char>&b){
  return a.first>b.first;
}
  string frequencySort(string s) {
    unordered_map<char, int> m;
    int n = s.length();
    for (int i = 0; i < n; i++) {
      m[s[i]]++;
    }

    vector<pair<int, char>> freqVec;
    for (auto& entry : m) {
            freqVec.push_back({entry.second, entry.first});
        }

        // Step 3: Sort the vector by frequency in descending order
        sort(freqVec.begin(), freqVec.end(), compare);

        // Step 4: Build the resulting string
        string result;
        for (auto& entry : freqVec) {
            result += string(entry.first, entry.second);
        }

        return result;
    }
};
//stl method
class Solution {
public:
    string frequencySort(string s) {
        unordered_map<char, int> m;
        int n = s.length();
        for (int i = 0; i < n; i++) {
            m[s[i]]++;
        }

        vector<pair<int, char>> freqVec;
        for (auto& entry : m) {
            freqVec.push_back({entry.second, entry.first});
        }

        // Step 3: Sort the vector by frequency in descending order
        sort(freqVec.begin(), freqVec.end(),[](const pair<int,char>a,const pair<int,char>b){
            return a.first>b.first;
        });

        // Step 4: Build the resulting string
        string result;
        for (auto& entry : freqVec) {
            result += string(entry.first, entry.second);
        }

        return result;
    }
};
```

Q8) https://leetcode.com/problems/word-pattern/

```
class Solution {
```

```
public:
   bool wordPattern(string pattern, string s) {
      vector<string> temp;
      stringstream ss(s);
      string token;
      int count = 0;
      while(getline(ss, token, ' ')) {
         temp.push_back(token);
         count++;
      }

       int n = pattern.size();

       if (count != n)
          return false;

       unordered_map<string, char> mp;
       set<char> used;
       for (int i = 0; i < n; i++){
           if (mp.find(temp[i]) == mp.end() &&
used.find(pattern[i]) == used.end()) {
               used.insert(pattern[i]);
               mp[temp[i]] = pattern[i];
           }
           else if (mp[temp[i]] != pattern[i]) {
               return false;
           }
       }
       return true;
   }
};
```

Q9) https://leetcode.com/problems/number-of-good-pairs/description/

```
class Solution {
public:
   int numIdenticalPairs(vector<int>& nums) {
      int result = 0;
      unordered_map<int, int> mp;

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

      for(auto &it : mp) {

         int count = it.second;
         result += (count * (count-1))/2;

      }

      return result;
```

```
    }

};
```

Q10) https://leetcode.com/problems/count-nice-pairs-in-an-array/description/

```
class Solution {

public:

    int M = 1e9+7;

    int reverse(int num) {

        int rev = 0;



        while(num > 0) {

            int rem = num%10;

            rev = rev*10 + rem;

            num /= 10;

        }



        return rev;

    }



    int countNicePairs(vector<int>& nums) {

        int n = nums.size();



        unordered_map<int, int> mp;



        //nums[i] - rev(nums[i]) ==  nums[j] - rev(nums[j])

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

            nums[i] = nums[i] - reverse(nums[i]);

        }



        int result = 0;

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

            result = (result + mp[nums[i]]) % M;

            mp[nums[i]]++;

        }



        return result;

    }

};
```

Q11) https://leetcode.com/problems/check-if-array-pairs-are-divisible-by-k/

```
//unordered multiset use

#include <vector>

#include <unordered_set>

using namespace std;



class Solution {

public:

    bool canArrange(vector<int>& arr, int k) {

        int n = arr.size();
        // If the number of elements is odd, we
can't pair all of them.

        if (n % 2 == 1) {

            return false;

        }



        unordered_multiset<int> st;



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

            // Compute the remainder of the
current element.

            int r1 = arr[i] % k;

            if (r1 < 0) {

                r1 += k; // Ensure remainder is non-
negative

            }



            // The other part of the pair should
sum with r1 to be a multiple of k.

            int r2 = (k - r1) % k;



            // Check if the counterpart of this
remainder exists in the multiset.

            auto it = st.find(r2);

            if (it != st.end()) {

                st.erase(it); // If found, remove one
counterpart.

            } else {

                st.insert(r1); // Otherwise, add the
current remainder.

            }

        }



        // If the multiset is empty, it means all
pairs are correctly formed.

        return st.empty();

    }

};



//map use



class Solution {

public:

    bool canArrange(vector<int>& arr, int k) {

        int n = arr.size();



        // If the number of elements is odd, we
can't pair all of them.

        if (n % 2 == 1) {

            return false;

        }



        unordered_map<int, int> count;



        for (int num : arr) {
```

```
      int remainder = num % k;
      if (remainder < 0) {
        remainder += k; // Adjust negative
remainder
      }
      count[remainder]++;
    }

    for (auto& pair : count) {
      int remainder = pair.first;
      int freq = pair.second;

      if (remainder == 0 || 2 * remainder ==
k) {
        // Check if the count of this
remainder is even
        if (freq % 2 != 0) {
          return false;
        }
      } else {
        // Check if counterpart remainder
exists and matches in count
        int counterpart = k - remainder;
        if (count[counterpart] != freq) {
          return false;
        }
      }
    }

    return true;
  }
};
```

Q12) https://leetcode.com/problems/hand-of-straights/description/

```
class Solution {
public:
  bool isNStraightHand(vector<int>& hand,
int groupSize) {
    int n = hand.size();
    if(n % groupSize) {
      return false;
    }
    map<int, int> mp;
    for(int &handNumber : hand) {
      mp[handNumber]++; //O(nlogn)
    }
    while(!mp.empty()) { //O(n*groupSize)
      int curr = mp.begin()->first; //->second
: frequency
      for(int i = 0; i < groupSize; i++) {
        if(mp[curr + i] == 0) {
          return false;
        }
```

```
                mp[curr+i]--;

                if(mp[curr+i] < 1) {

                    mp.erase(curr+i);

                }

            }

        }

        return true;


    }

};
```

Q13) https://leetcode.com/problems/replace-words/

```
class Solution {

public:



    string findRoot(string& word, unordered_set<string>& st) {



        //try all length substring starting from 0th index

        for(int l = 1; l <= word.length(); l++) {

            string root = word.substr(0, l);

            if(st.count(root)) {

                return root;

            }

        }

        return word;


    }


    string replaceWords(vector<string>& dictionary, string sentence) {

        unordered_set<string> st(dictionary.begin(), dictionary.end());


        stringstream ss(sentence);

        string word;

        string result;


        while(getline(ss, word, ' ')) {

            result += findRoot(word, st) + " ";

        }


        result.pop_back();

        return result;


    }

};
```

Q14) https://leetcode.com/problems/relative-sort-array/description/

```
//Approach-1 (Using counting sort)
```

```
//T.C : O(nlogn)
//S.C : O(n)
class Solution {
public:
    vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {
        map<int, int> mp;

        for(int &num : arr1) {
            mp[num]++;
        }

        int i = 0;
        for(int &num : arr2) {
            while(mp[num]-- > 0) {//current value before decrement
                arr1[i++] = num;
            }
        }

        for(auto &it : mp) {
            int freq = it.second;
            while(freq > 0) {
                arr1[i++] = it.first;
                freq--;
            }
        }

        return arr1;
    }
};
```

```
//Approach-2 (Using lambda)
//T.C : O(nlogn)
//S.C : O(n)
class Solution {
public:
    vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {
        unordered_map<int, int> mp;

        for(int i = 0; i < arr2.size(); i++) {
            mp[arr2[i]] = i;
        }

        for(int &num : arr1) {
            if(!mp.count(num)) {
                mp[num] = 1e9;
            }
        }
        auto lambda = [&](int &num1, int
&num2){

            if(mp[num1] == mp[num2]) { //1e9

                return num1 < num2;

            }



            return mp[num1] < mp[num2];

        };



        sort(begin(arr1), end(arr1), lambda);



        return arr1;

    }

};
```

Q15) https://leetcode.com/problems/minimum-number-of-moves-to-seat-everyone/description/

```
class Solution {

public:

    int minMovesToSeat(vector<int>& seats,
vector<int>& students) {

        int n = students.size();

        vector<int> position_seat(101, 0);

        vector<int> position_stud(101, 0);



        for(int &x : seats) {

            position_seat[x]++;

        }

        for(int &x : students) {

            position_stud[x]++;

        }



        int i = 0;

        int j = 0;

        int result = 0;



        while(i <= 100 && j <= 100) {

            if(position_seat[i] == 0) i++;

            if(position_stud[j] == 0) j++;



            if(i <= 100 && j <= 100 &&
position_seat[i] != 0 && position_stud[j] != 0)
{

                result += abs(i-j);

                position_seat[i]--;

                position_stud[j]--;

                n--;

            }

        }



        return result;



    }
};



//Approach-2 (Using sorting)

//T.C : O(nlogn)

//S.C : O(1)

class Solution {

public:

   int minMovesToSeat(vector<int>& seats,
vector<int>& students) {

       sort(begin(seats), end(seats));

       sort(begin(students), end(students));



       int moves = 0;



       int n = seats.size();

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

           moves += abs(seats[i] - students[i]);

       }



       return moves;

   }

};
```

BIT MANIPULATION

Q1) https://leetcode.com/problems/minimum-flips-to-make-a-or-b-equal-to-c/description/

```
class Solution {
public:
    int minFlips(int a, int b, int c) {
        int result = 0;

        while( a != 0 || b != 0 || c != 0) {

            if( (c & 1) == 1) {
                if((a & 1) == 0 && (b & 1) == 0) {
                    result++;
                }
            } else {
                result += (a & 1) + (b & 1);
            }

            a >>= 1;
            b >>= 1;
            c >>= 1;

        }

        return result;
    }
};
```

//Approach-2 (Using inbuild function)

```
class Solution {
public:
    int minFlips(int a, int b, int c) {
        int result = (a | b) ^ c;
        return __builtin_popcount(result) + __builtin_popcount((a & b) & (result));
    }
};
```

Q2) **https://leetcode.com/problems/single-number-ii/description/**

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {

        int result = 0;

        for(int i = 0; i<32; i++) {

            int temp = (1 << i);

            int countOne  = 0;
            int countZero = 0;
            for(int &num : nums) {

                if((num&temp) == 0) {
                    countZero++;
                } else {
                    countOne++;
                }

            }

            if(countOne % 3 == 1) {
                result = (result | temp);
            }

        }

        return result;

    }
};
```

Q3)

https://leetcode.com/problems/sort-integers-by-the-number-of-1-bits/

```
//lambda method
class Solution {
public:
    vector<int> sortByBits(vector<int>& arr) {

        auto lambda = [&](int &a, int &b) {

            int count_a = __builtin_popcount(a);
            int count_b = __builtin_popcount(b);

            if(count_a == count_b)
                return a<b;

            return count_a < count_b;

        };

        sort(begin(arr), end(arr), lambda);

        return arr;
    }
};
//comparator function

class Solution {
public:
 static bool compare(int a, int b) {
    int countA = __builtin_popcount(a);
    int countB = __builtin_popcount(b);
    if (countA == countB) {
        return a < b;
    }
    return countA < countB;
}
    vector<int> sortByBits(vector<int>& arr) {
        sort(begin(arr), end(arr), compare);


        return arr;
    }
};
```

Q4) **https://leetcode.com/problems/find-the-original-array-of-prefix-xor/description/**

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


        vector<int> arr;
        arr.push_back(pref[0]);


        for(int i = 1; i<n; i++) {
            arr.push_back(pref[i] ^ pref[i-1]);
        }



        return arr;
    }
};
```

Q5) **https://leetcode.com/problems/maximum-xor-product/**

```
class Solution {
public:
    int M = 1e9+7;
    typedef long long ll;
    int maximumXorProduct(long long a, long long b, int n) {


        ll xXora = 0;
        ll xXorb = 0;
        /*
            0 <= a, b < 2^50
            So, a and b will be represented by 50 bits only (0th bit to 49th bit)
            Now, what if x value is something which can be represented by only say 3 bits
            So, x = 0000000000000......11 (50 bits)
            So, let's say a = 101......000000000001 (50 bits)
                          x = 0000000000000......11 (50 bits, but only starting 2 bits matter)


            Now, notice that for a^x, From 49th bit to nth bit will be same as what is present in 'a'
            Hence the extra for loop below takes care of that
        */
        for(long long i = 49; i >= n; i--) {

           bool aset = (a >> i) & 1 > 0; //Finding
the ith bit of a

           bool bset = (b >> i) & 1 > 0; //Finding
the ith bit of b



           if(aset)

              xXora ^= (1ll << i);

           if(bset)

              xXorb ^= (1ll << i);

        }



        /*

           Given constraint : n = 0 to 50

           So, x = 2^0 to 2^50

           2^50 = 1000000000.......0 (total 50 bits
from 0th bit in right to 49th i.e. (n-1)th bit in
left)

           So, we will check from (n-1)th bit of a
and b as well along with x formation

        */

        for (long long i = n-1; i >= 0; i--) {



           bool aset = (a & (1ll << i)) > 0;
//Finding the ith bit of a

           bool bset = (b & (1ll << i)) > 0;
//Finding the ith bit of b



           //If both ith bit of a and b are same

           if(aset == bset) {

              xXora ^= (1ll << i);

              xXorb ^= (1ll << i);

              continue;

           }



           if(xXora > xXorb) {

              xXorb ^= (1ll << i);

           } else {

              xXora ^= (1ll << i);

           }



        }



        xXora %= M;

        xXorb %= M;

        return (xXora * xXorb) % M;

    }

};
```

Q6) https://leetcode.com/problems/minimum-one-bit-operations-to-make-integers-zero/description/

```
class Solution {

public:

    int minimumOneBitOperations(int n) {

        if(n == 0)

            return 0;
        vector<long long> function(32, 0);

        //function[i] = x

        //Means it will take x operations to make
ith bit 1

        function[0] = 1;

        for(int i = 1; i <= 31; i++) {

            function[i] = 2*function[i-1] + 1;

        }


        int result = 0;

        int sign  = 1;


        for(int i = 30; i >= 0; i--) {


            int ith_bit = ((1 << i) & n);


            if(ith_bit == 0) {

                continue;

            }


            if(sign > 0)

                result += function[i];

            else

                result -= function[i];


            sign *= -1;

        }


        return result;

    }

};
```

Q7) https://leetcode.com/problems/bitwise-and-of-numbers-range/description/

```
class Solution {

public:

    int rangeBitwiseAnd(int left, int right) {

        int shiftCount = 0;


        while(left != right) {

            left >>= 1;

            right >>= 1;

            shiftCount++;

        }


        return left << shiftCount;

    }

};

//better approach

class Solution {

public:

    int rangeBitwiseAnd(int left, int right) {

        while(right > left) {
            right = right & (right-1);

        }



        return right;

    }

};
```

Q8)

https://leetcode.com/problems/minimum-number-of-operations-to-make-array-xor-equal-to-k/description/

```
class Solution {

public:

    int minOperations(vector<int>& nums, int k) {

        int totalXor = 0;



        for(int &num : nums) {

            totalXor ^= num;

        }




        int diff = (totalXor ^ k);



        return __builtin_popcount(diff);



    }

};
```

Q9) https://leetcode.com/problems/number-of-wonderful-substrings/description/

Prefix use in cumulative xor

```
class Solution {

public:

    typedef long long ll;

    long long wonderfulSubstrings(string word) {

        unordered_map<ll, ll> mp;



        mp[0] = 1;

        int cum_xor = 0;



        ll result = 0;





        for(char &ch : word) {



            int shift = ch - 'a';



            cum_xor ^= (1 << shift);



            result += mp[cum_xor];



            for(char ch1 = 'a' ; ch1 <= 'j'; ch1++) {

                shift = ch1 - 'a';
                ll check_xor = (cum_xor ^ (1 <<
shift));

                result += mp[check_xor];
            }

            mp[cum_xor]++;

        }

        return result;
    }
};
```

Q10) https://leetcode.com/problems/count-triplets-that-can-form-two-arrays-of-equal-xor/description/

```
class Solution {
public:
    int countTriplets(vector<int>& arr) {
        vector<int> prefixXor(begin(arr),
end(arr));

        prefixXor.insert(prefixXor.begin(), 0);
//initially the xor cumulative will be 0
        int n = prefixXor.size();

        for(int i = 1; i < n; i++) {
            prefixXor[i] ^= prefixXor[i-1];
        }

        int triplets = 0;

        for(int i = 0; i < n; i++) {
            for(int k = i+1; k < n; k++) {

                if(prefixXor[k] == prefixXor[i]) {
                    triplets += k-i-1;
                }

            }
        }

        return triplets;
    }
};
```