Day-4

1> Two Sum

eg:- [2, 7, 11, 15] target = 9 o/p:- [0, 1]

① **Brute force:-** for every a[i] check if in the remaining part of array (target - a[i]) exist or not, if yes then return the answer

TC :- $O(n^2)$
SC :- $O(1)$

② **using Hashing:-** while traversing on array, check if map contain (target - a[i]), if yes, then return answer(indexes) else put a[i] on the map. (nums[i] as key & index (i) as value)

#code

```
v<i> twosum (v<i>& nums, target){
    unordered_map < i, i > m;
    v<i> ans;
    for(i=0 — i<nums.size){
        if (m.find(target - nums[i]) != m.end()){
            ans.p_b(target - nums[i]);
            ans.p_b(i);
            break;
        }
        else m[nums[i]] = i;
    }
    return ans;
}
```

TC :- O(n)
SC :- O(n)

ii) **four Sum** :— $0 <= a, b, c, d < n$, such that, Give all such unique quadruplets, $nums[a] + nums[b] + nums[c] + nums[d] = target$.

eg:- [1, 0, -1, 0, -2, 2] target = 0

O/P:- {[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]}

① **Brute force** :— Sort → 3ptr + BS.

i j k k k k

[-2 -1 -1 0 0 1 1 2 2 4]

target = 9

TC :- $O(n^3 \log n) + n \log n$
SC :- $O(1)$

```
i = 0
    j = i+1
        k = j+1
```

→ After these 3 loop, do BS on remaining part of array and check for [target − a[i] − a[j] − a[k]], if it is, then put it on set DS (to get unique only) so on.

Sum (a[i]+a[j]+a[k])	remaining target (target − sum)		set(DS) a(i, j, k, ...)
3	6	X	
4	5	X	
4	5	X	
5	4	✓	(1, 1, 3, 4)
1			

② Optimised approach

target = 9

→ first sort the array
→ so, we'll keep 2 pointers i and j, and on remaining right part we'll use 2 pointer technique (left, right) and find if there is or not. new_target (target − a[i] − a[j])

```
 i  j  left                    right
[1  1  1  2  2  3  3  4  4  4]
```

new target = 9 − 1 − 1 = 7

1 + 4 < 7 (left++)
2 + 4 < 7 (left++)

→ now to reduce time, we'll now, move left to 3.

3 + 4 == 7 → Put in DS. [1, 1, 3, 4]

and so, on,

TC:- O(n³)
SC:- O(1)

→ To remove duplicacy, we'll everytime move each of our pointer (i, j, left, right) such that their current value is not equal to next value.

#code

```
vv<i> quad(4,0);
quad[0] = nums[i];
quad[1] = nums[j];
quad[2] = nums[front];
quad[3] = nums[back];
res.push_back(quad);
```

```
vv<vi> foursum(vi& nums, int t){
    vv<vi> res;
    if(nums.size()<4) return res;
    n = nums.size();
    sort(nums.begin(), nums.end());
    for(i=0 ——— i<n){
        for(j=i+1 ——— j<n){
            t2 = t1 = nums[j] - nums[i];
            front = j+1;
            back = n-1;
            while(front<back){
                ts = nums[front] + nums[back];
                if(ts<t2) front++;
                else if(ts>t2) back--;
                else {
                    while(front<back and nums[front]==quad[2])
                        ++front;
                    while(front<back and nums[back]==quad[3])
                        --back;
                }
            }
            while(j+1<n and nums[j+1]==nums[j]) ++j;
        }
        while(i+1<n and nums[i+1]==nums[i]) ++i;
    }
    return res;
}
```

iii) **Longest Consecutive Sequence** :- Given unsorted array, return the length of the longest consecutive ele.

eg:- [100, 4, 200, 1, 3, 2] . o/p:- 4 {1, 2, 3, 4}

eg:- [102, 4, 100, 1, 101, 3, 2] o/p:- 4.

① **Brute force** :- Sort the array. [1, 2, 3, 4, 100, 101, 102]

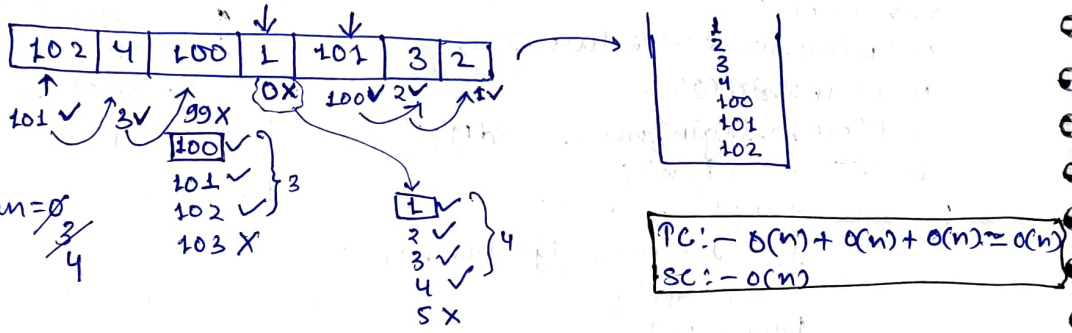
Then check for consecutive sequence. → longest ⇒ 4 Ans

TC :- O(n log n) + O(n)
SC :- O(1) or O(n) → for merge sort.

② **optimised approach** :- 1) Traverse in array and put ele's in set, DS.

2) Again traverse and for every a[i], check if (a[i] - 1) exist in set or, not, → if yes → do nothing,
else → Then do check for (a[i] + 1) exist in set or not, till it doesn't exist.
→ maintain cnt for this ~~and update~~
update max_length with this count.

→



max_len = 0, 3, 4

TC :- O(n) + O(n) + O(n) ≈ O(n)
SC :- O(n)

Intution :- Everytime, checking for a[i] - 1, because we want to start cnting for consecutive sequence and for that, we need to go lowest value of that sequence.

#code

```
{ unordered_set<int> s;
  for (i=0 — i<nums.size()){
     s.insert(nums[i]);
  }
  int mx_len=0;
  for(i=0 — i<nums.size()){
     if (s.find(nums[i]-1)==s.end()){
        int j=1;
        while(s.find(nums[i]+j)!=s.end()) j++;
        mx_len = max(mx_len, j);
     }
  }
  return mx_len;
```

iv) Largest Subarray with 0 sum :-

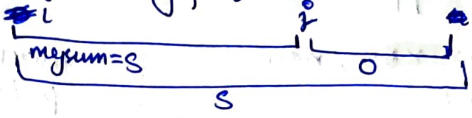
eg:- [15, -2, 2, -8, 1, 7, 10, 23] O/P: 5

① **Brute force:-** Generate all subarrays and check which have sum = 0 and largest in length.

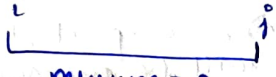
TC :- O(n3) → this can be optimised to O(n²)
SC :- O(1)

② **Optimal:-(hashing):** → using prefix sum, and hashing,

① ~~Prefix sum~~



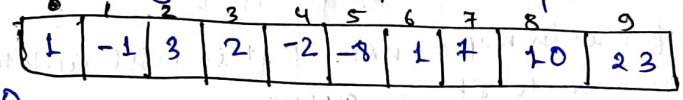
then, len = max(len, k - j);

② 

if till j sum = 0 and till k also s = 0 which means, after j there is something added to s(i.e) which again gave sum as 0.

→ while traversing we'll keep a sum variable and check if that sum exist in map or not, if not then add to map (sum, index).

If already exist in map, then update length, with (curindex - map me jo th uska index)



S=0
sum → 1 0 3 5 3 -5 -4 3

(36, 9)
(13, 8)
(-4, 6)
(5, 5)
(5, 3)
(3, 2) ← (4-2=2) (7-2=5)
(1, 0)

maxi = 2 2 5

→ If we find that sum in map, then we'll not update it because to get max len.

#code

```
int maxLen(int A[], int n){
    unordered_map<int,int> mpp;
    int maxi=0, sum=0;
    for(i=0 ——— i<n) {
        sum += A[i];
        if(sum==0) maxi = i+1;
        else {
            if(mpp.find(sum)!=mpp.end())
                maxi = max(maxi, i-mpp[sum]);
            else
                mpp[sum] = i;
        }
    }
    return maxi;
}
```

TC :- O(nlogn)
SC :- O(n)

v) Count number of subarray with given xor K :-

eg:- {4, 2, 2, 6, 4} m=6 o/p:- 4

① **Brute force** :- Generate all subarrays and do xor of all subarrays if xor found to be 'm', cnt++

TC :- $O(n^3)$ → $O(n^2)$ (can be reduced to this with few observations)
SC :- $O(1)$

② **Optimised** :- (approach is bit similar to, subarray sum equals K)

We'll have a prefix xor, now

K ⊕ my curxor

If I xor something with my curxor, let say Y and, if :

(curxor ^ Y) = K, then that will be our answer.

→ Now to find, Y we'll check in map

equivalent to

Y = K^curxor, if Y is in map or not. If it is will update our cnt by it's frequency and update it's frequency.

~~eg:~~ ~~{4, 2, 2, 6, 4}~~

#code

TC :- $O(n \log n)$
SC :- $O(n)$

```
{  map<int, int> freq;
   cnt = 0, xor = 0;
   for(auto it : A) {
      xor = xor ^ it;
      if (xor == B)
         cnt++;
      if ( freq.find(xor^B) != freq.end())
         cnt += freq[xor^B];
      freq[xor]++;
   }
   return cnt;
}
```

How?

We'll do prefix sum and check for (K - psum) in map, it it exist then that will be one subarray.

Eg. Let psum = 5, K = 12

search (12 - 5 = 7) in map, if map has, then it will be our ans. Eg. (There is subarray with sum 7 and my cursum = 5, and both add up to get K (i.e 12).

Ni> Longest Substring without Repeat:-

eg:- "abcabcbb" o/p:- 3

① **Brute force**:- Generate all substrings, and using hashing check that substring has a repeating characters or not and among those who are without repeat take maximum among all

TC:- O(n³) → O(n²)
SC:- O(n)

② **Optimal approach**:- 2 ptr (variable size sliding window) (i, j)

Traverse through string and check if it exist in map or not if not, then add to set., if it exist, then start moving l ptr till set don't contain s[i]. everytime update the length.

eg:- "a b c a a b c b d a"
0 1 2 3 4 5 6 7 8 9

len = 0 1 2 3 4

(r-l+1) (r-l+1) = 9-6+1 = 4

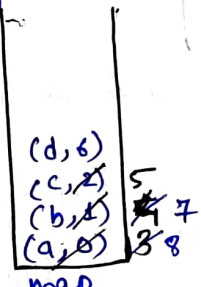
Any

TC:- O(2n) → (There can be a case, when we need to move r till end and after that l till end, eg:- "abcdefghh"
SC:- O(n) → for set DS

③ **most optimal** to reduce in exact O(n). Take map DS to store index of its last occurence. So, while traversing, if we find that value in our map, then instead of moving 'l' ptr by 1 step every time we'll move it to lastoccurence+1,

TC:- O(n)
SC:- O(n)

eg:- "a b c a b c d b a"
0 1 2 3 4 5 6 7 8



len = 0 1 2 3 4

#code

```
int len(string s){
    vi> m(256, -1); //There are 256 different chars.

    left=0, right=0;
    n=s.size, len=0;
    while(right<n){
        if(m[s[right]]!=-1) left=max(m[s[right]]+1, left);
        m[s[right]]=right;
        len=max(len, right-left+1);
        right++;
    }return len;
}
```