otes

Day - 4

Problem Name : 2-Sum Problem

nums := [2, 7, 11, 15] | output : [0, 1]
target = 9

```
// first Approach → Brute force
// for (i to n)
   for (j = i+1 to n)
   if (a[i] + a[j] == target)
      res.push_back(i)
      res.push_back(j)
```

Remarks return res

TC : $O(N^2)$
SC : $O(1)$

//Second Approach : Two pointer
TC: O(NlogN) & SC: O(N)

// Third approach: TC: O(N)
OC: O(N)

```
// using map
  vector<int> res;
  unordered_map<int, int> mp;

  for(int i=0; i<n; i++)
   {
     if (mp.find(target-nums[i])!=
                          mp.end()) {

        res.emplace_back(i);
        res.emplace_back(mp[target-nums[i]]
        return res;
   }
   mp[nums[i]]=i;
  }
   return res;
```

Remarks

Problem Name: 4 Sum Problem

arr[] = [1, 0, -1, 0, -2, 2] , target = 0

output: [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]

// First approach: Brute force

// logic: target - (nums[i] + nums[j] + nums[k])

// apply Binary search in right half!

```
// i = 0 -> n
   j = i+1 -> n
   k = j+1 -> n

int x = target - (n[i] + n[j] + n[k])

if( binary_search (nums.begin() + k+1,
                   nums.end(), x)) {
```

Remarks
```
vector<int> v;
v.push (i)
v.push (j)
v.push(k)
v.push(x)
```

te__/__/______

```
vector<vector<int>> res(S.begin(), S.end());
return res;
```

Tc: O(NlogN + N^3 logN)
Sc: O(m-x1)

// Second Approach → optimized approach

// logic: Sort the array, fix two pointers, and in remaining sum will be (target - (nums[i] + nums[j]))

// Sort code:

```
→ sort
 → i=0; i<n
 → target3 = target - num[i]
→ j=i+1; j<n
   target2 = target3 - num[j]
→ int front = j+1;
→ int back = n-1;
→ while (front < back)  {
Remarks
  int twosum = num[front] + num[back]
  if ( twosum < target2 )  front ++;
  else ( twosum > target )  back --;
  else {
```

www.lawrencemayo.ac.in

Notes

Problem Name: LCS in an Array

Input: [100, 200, 1, 3, 2, 4]
output: 4

// First approach → Brute force.

```
// Sort the array
// take var ans=1, curr=1, prev = nums[0]
// for (int i=1 ; i<n)
//   if (nums[i]= = prev+1 ) {
        curr++;
  }
// else if (nums[i] ! = prev ) {
        curr=1;
}
   prev = nums[i];
   ans = max(ans, curr)
   return ans.
                )
```

TC: O(N log N)
SC: O(N)

Remarks

Date__/__/______ No

Second approach → optimal

```
//using set
//for( int num:nums) {
      hashSet.insert (num);
}
// int ls = 0;
// for(int num : nums) {
    if (!hashSet.count (num-1)) {
       int currNum= Num;
        int currStreak = 1;
while (hashSet.count (currNum+1)) {
      currNum += 1;
      currStreak += 1;
  }
    ls = max (ls, currStreak)
return ls;
```

Tc: O(N)
Sc: O(N)

Remarks