Todays Content

1. Count of subarrays with sum = k
2. longest increasing sequence

28 Given ar[N] & k

Calculate & return no: of subarrays with sum = k.

k = 4	0	1	2	3	4	5	6	7	8
ar[] = {	3	4	1	3	-4	5	-1	-5	-6 }

Note: write [s..e]

Subarrays: {1 3} {4} {5 -1} {1 3 -4 5 -1} {4 1 3 -4} ans = 5

Idea: Generate all subarrays calculate sum & sum == k: cnt

Way 1: TC: O(N3)

```
ans = 0;
s=0; s<N; s++) {
    e=s; e<N; e++) {
        #[s..e];
        sum = 0;
        i=s; i<=e; i++){
            sum = sum + ar[i]
        if[sum==k) {
            ans++;
        }
    }
}
return ans;
```

Way 2: TC: O(N^2 * 1) = O(N^2) SC: O(N)

```
Generate pf[N];
ans = 0;
s=0; s<N; s++) {
    e=s) e<N; e++) {
        #[s..e];
        sum = 0;
        if[s==0) { sum = pf[e]}
        else{sum = pf[e) - pf[s-1]}
        if[sum==k){
            ans++;
        }
    }
}
return ans;
```

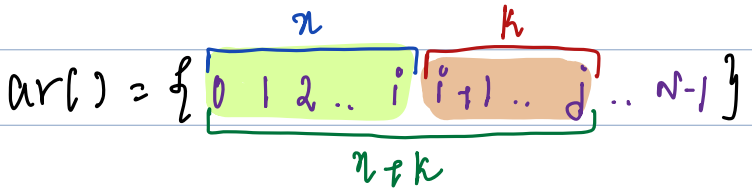
Idea2: Create pf[]

	0	1	2	3	4	5	6	7	8	9
ar[] = {	4	-2	4	-1	4	3	6	-4	1	5 }
psum[] = {	4	2	6	5	9	12	18	14	15	20 }

k = 10

obs: $pf[j] - pf[i] = k \Rightarrow sum[i+1..j] = k$

Assume $pf[i] = x$, $pf[j] = x + k$



Qn: Calculate no: of pairs (i, j) such that $pf[j] - pf[i] = k$ & $j > i$

Dry Run:

	0	1	2	3	4	5	6	7
ar[] = {	1	2	3	2	-7	2	3	2 }

$P[j] - P[i] = 5$ $j > i$

k = 5

pf[] = { 1 3 6 8 1 3 6 8 }

6 8 11 13 6 8 11 13

cnt-4 : 0

cnt-2 : 0

cnt 1 : 1

cnt 3 : 1

cnt-4 : 0

cnt-2 : 0

cnt 1 : 2

cnt 3 : 2

Ans = 6.

Idea:

Idea: For every pf[j];
Calculate freqy of pf[j]-k in left of j; [0..j-1]

Opt; Use hashmap
At pf[j]: We only iterate in [0..j-1]
Note: At pf[j]: Hashmap should only contain [0..j-1]

Dry Run:

	0	1	2	3	4	5	6	7	8	
ar[] =	1	2	3	-1	3	-7	2	3	2	
k=5										
	0	1	2	3	4	5	6	7	8	j
pf[] = 0	1	3	6	5	8	1	3	6	8	ans = 7
Target =	-4	-2	1	0	3	-4	-2	1	3	
cnt =	0	0	1	0/1	1	0	0	2	2	

HashMap:

<1:2>
<3:2>
<6:2>
<0:1>
<5:1>
<8:2>

↳ Edge Case: 0 already exists at start, before pf[], initialize hashmap with <0,1>

```
int subarrays(vector<int>& arr, long k){ TC: O(N+N) = O(N)
                                         SC: O(N+N) = O(N)
    long pf[N];
    long sum=0;
    for(int i=0; i<N; i++){
        sum = sum + arr[i];
        pf[i] = sum;
    }

    unordered_map<long,int> hm;
    hm[0] = 1;
    int ans=0;
    for(int j=0; j<N; j++){
        # pf[j], Target = pf[j] - k.
        if(hm.find(pf[j]-k) != hm.end()){
            ans = ans + hm[pf[j]-k]
        }

        # Insert pf[j]
        hm[pf[j]]++;
    }

    return ans;
}
```

28 Given ar[N] elements, calculate length of longest subset which can be re-arranged in strictly Increasing Order by 1.

Note: Pick any no: of ar[] elements in any order.

Constraints

$1 <= N <= 10^6$

$-10^9 <= ar[i] <= 10^9$

	0	1	2	3	4	5	6	7
Ex: ar[] = {	-1	8	5	3	10	2	4	9 }

Ex: {8 10 9} = {8 9 10} l = 3

Ex: {5 3 2 4} = {2 3 4 5} l = 4

	0	1	2	3	4	5	6	7	8	9
Ex2: ar[] = {	3	8	2	1	9	6	5	6	7	2 }

Ex: {8 9 6 5 7} = {5 6 7 8 9} l = 5

Ex: {6 5 6 7 8} = {5 6 6 7 8} * invalid.

Ideal:

Idea 1: Sort arr[] & compare adj elements

Dry Run:

0	1	2	3	4	5	6	7
-1	8	5	3	10	2	4	9

arr[] = { -1 2 3 4 5 8 9 10 }

len: 1 4 3 maxl = 4.

0	1	2	3	4	5	6	7	8	9
3	8	2	1	9	6	5	6	7	2

arr[] = { 1 2 2 3 5 6 6 7 8 9 }

len 2 2 2 4 maxl = 4.

arr[] = { 1 2=2 3 | 5 6==6 7 8 9 }

cnt: 1 +1 +1 | 1 +1 +1 +1 +1

len = 3 len = 5. max l = 5.

Idea: TODO

1. Sort arr[] TC: O(N log N + N)
2. Iterate in arr[] & compare adj elements.

if data inc by 1: inc cnt by +1

else if data same: Don inc cnt continue proces

else {

compare cnt with maj

cnt = 0

}

Idea3: Assume at every ar[i] we start a sequence
Calculate sequence length starting at ar[i]

0	1	2	3	4	5	6	7
-1	8	5	3	10	2	4	9

ar[] = { -1 8 5 3 10 2 4 9 }

Start:

-1 : 0* = 1

8 : 9 10 11* = 3

5 : 6* = 1

3 4 5 6* = 3

10 11* = 1

2 3 4 5 6* = 4

4 5 6* = 2

9 10 11* = 2

Note: To optimize searching process store all elements in hashset, to search an element: O(1)

ar[] = { 9 7 6 8 10 } store data hs: { 7 6 8 10 9 }

Start:

9 10 11* = 2

7 8 9 10 11* = 4

6 7 8 9 10 11* = 5

8 9 10 11* = 3

10 11* = 1

Total iterations = 2 + 4 + 5 + 3 + 1
= 1 + 2 + 3 + 4 + 5 = 15

TC: For ar[N]:
Total iterations = 1 + 2 + .. N = $\frac{(N)(N+1)}{2}$

Hint1: We will start sequence from ar[i] only if ar[i]-1 is not present in hashset.

ar[] = { 9 7 6 8 10 }

✓8 9*

✓6 7*

5 6 7 8 9 10 11 = 5

✓7 8*

✓9 10*

Eg1:

	0	1	2	3	4	5	6	7
arr[] = {	6	6	6	6	8	9	7	10 } → hashset { 6 8 9 7 10 }

*5 6 7 8 9 10 11 *

*5 6 7 8 9 10 11 *

*5 6 7 8 9 10 11 *

*5 6 7 8 9 10 11 *

Hint1: In arr[] repeatation possible.
Instead of iterating in arr[], iterate in hashset to avoid repeatitions.

Final Idea: Insert elements in hashset & iterate in hashset.
From an element n:
1. We start sequence if n-1 doesn't exist in hashset
2. Get sequence length starting from n & get overall max

Dry Run:

	0	1	2	3	4	5	6	7	8
arr[] = {	-1	8	5	3	10	2	4	9	2 }

hs: { -1 8 3 5 10 2 9 4 } max = 4

-2* -1: 0* l = 1

7* 8: 9 10 11* l = 3

2 3 *

4 5 *

9 10 *

1* 2 3 4 5 6* l = 4

8 9 *

3 4 *

Total iterating = N + N = O(N)
Outer loop = N
Inner loop = N: a element will atmost come in 1 sequence
Adding all sequence = O(N)

```
int longestSequence(vector<int>& arr) {   TC: O(N+N) = O(N)
    unordered_set<int> hs;                SC: O(N)
    for(int i=0; i<arr.size(); i++) {
        hs.insert(arr[i]);
    }

    int ans=0;
    for(auto n : hs) {
        # if n-1 not present in hs start sequence from n
        if(hs.find(n-1) == hs.end()) {
            int s=n, l=0; # start sequence in s.
            while(hs.find(s) != hs.end()) {   s=2→3→4→5→6*
                l++; # one count              l=0 +1 +1 +1 +1
                s++; # Goto next ele in seq
            }

            ans = max(ans, l)
        }
    }
    return ans;
}
```