Liding Window

- Problems / Concept

(21) Given an Array of integers and a number k, find the maximum sum of k consecutive elements ?

Naive → O(nk)
efficient → O(n)

```
int f(int k, vector<int> &a, int n)
{
    int ms=0;
    for(int i=0; i<k; i++)
        ms += a[i];
    int sum=ms;
    for(int i=k; i<n; i++)
    {
        sum += a[i]-a[i-k];
        ms = max(ms, sum);
    }
    return ms;
}
```

I/P → {1,8,30,-5,20,7}, k=3
O/P → 45

[1,8,30] -5, 20, 7
window slides by 1.

(22) Given Array A, & k (Window size) find if there is a subarray of sum equal to given sum

Solution → same approach as above

** (23) Given an Unsorted Array of non-negative integers. Find if there is a subarray with given sum.

Naive → O(n²) eg: {1,4,20,3,10,5} sum=33
efficient → O(n) Approach we maintain 2 pointers start(s) end(e)

→ we increment start

1, 4, 20, 3, 10, 5
↑
s, e

→ when current sum(cs) increases given sum(k) then we move start ahead (st++) and remove the a[s] element. until cs <= k.

∴ while loop.

```
bool f(int a[], int k)
{
    int s=0, e, cs=a[0];
    for(e=1; e<n; e++)
    {
        while(cs>k && e>=s)        // start never cross end
        {                          // O(n) in total
            cs = cs - a[s];
            s++;
        }
        if(cs==k)
            return true;
        cs = cs + a[e];
    }
    return (cs==k);
}
```

while loop is written just b/z say {0,5,3,1,4} k=6
a[0] > k
∴ we have to remove it then add other numbers.

O(2n) → O(n)

* This approach don't work for negative elements
eg: {4,7,-3,1,2} sum = 9
(9)
but 4+7 = 11 > 9, we with this approach we will remove 4. causing no answer.

(24) N-bonacci Numbers:

→ Fibonacci is 2-bonacci, i.e every element is sum of two previous element.

→ when N=4 means every element is sum of previous 4 element and number starting 4 elements must be given and those are {0,0,0,1}

→ when N=3 starting no. are {0,0,1}

→ ___ N=5 ___ {0,0,0,0,1}

Q. Print first m N-bonacci numbers.

```
void f(int n, int m)
{
    int a[m];
    for(int i=0; i<n-1; i++)
        a[i]=0;
    a[n-1]=1;
    int cs=1;
    for(int i=n; i<m; i++)
    {
        a[i]=cs;
        cs=cs+a[i]-a[i-n];
    }
    for(int i=0; i<m; i++)
        cout<<a[i]<<" ";
}
```

↓

Window Sliding Technique

O(4) both

```
void f(int n, int m)
{
    int a[m];
    for(int i=0; i<n-1; i++)
        a[i]=0;
    a[n-1]=a[n]=1;
    int cs=1;
    for(int i=n+1; i<m; i++)
    {
        a[i]=2*a[i-1]-a[i-n-1];
    }
    for(int i=0; i<m; i++)
        cout<<a[i]<<" ";
}
```

↓

n=3, m=9

0, 0, 1, 1, 2, 4, 7, 13, 24

13 = 2*7 - 1

24 = 2*13 - 2

(27) Find the duplicate Number:

Given an array of (n+1) integers, where each integer is in range [1,n]

I/P → 5, 4, 3, 2, 1, 2

O/P → 2

(I) Using sorting O(nlogn) SC → O(1)

(II) " Hashing O(n) SC → O(n)

(III) Tortoise Method:

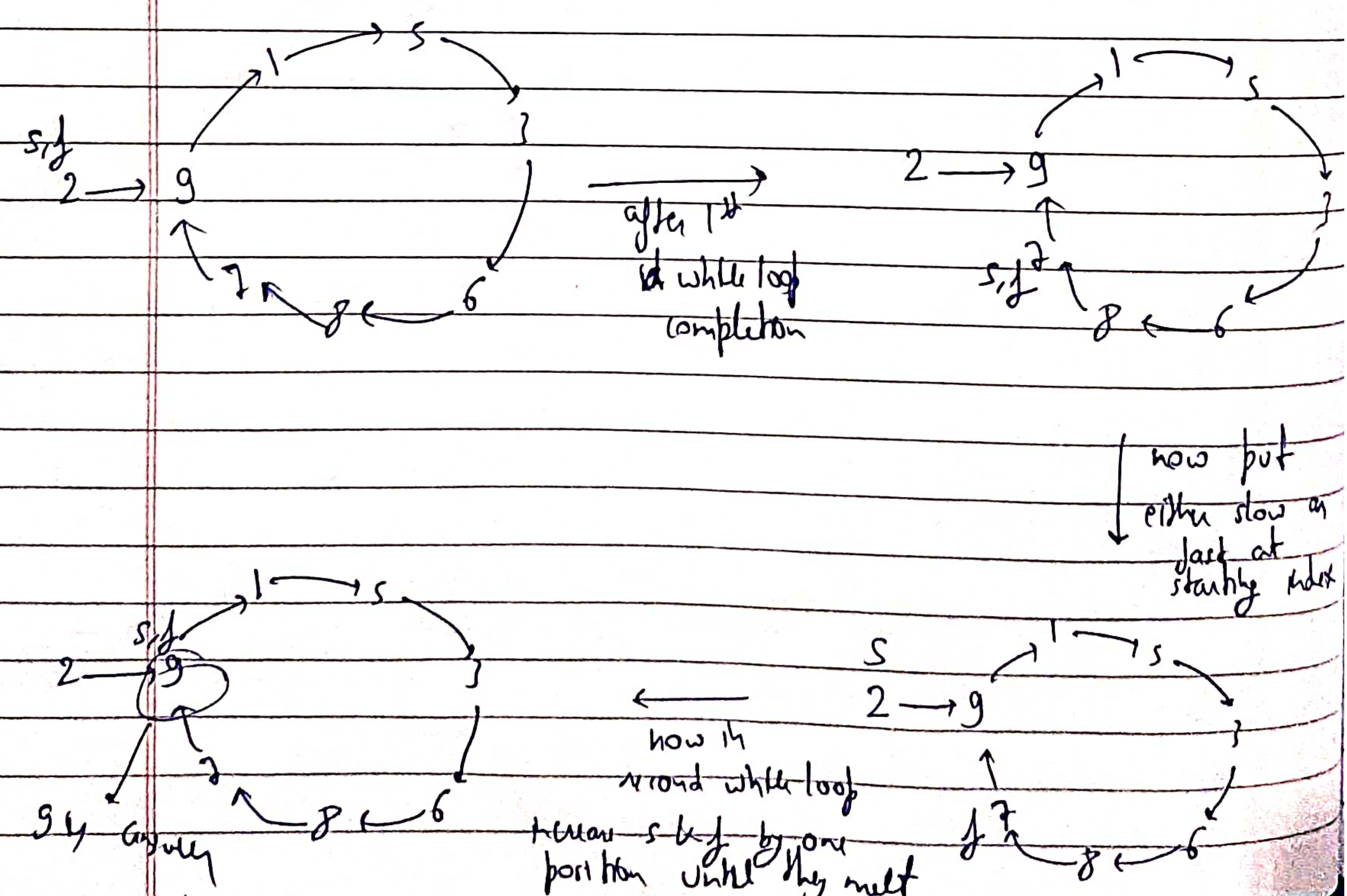
we maintain two pointer slow & fast

eg: {2, 5, 9, 6, 9, 3, 8, 9, 7, 1}
(indices 0 1 2 3 4 5 6 7 8 9; s, f at index 0)

→ in first loop we find when they (s, f) meet for first time.

→ in second " ——————— second "

→ second time meeting is the answer



```
int f(vector<int> &v)
{
    int slow = v[0], fast = v[0];
    do
    {
        slow = v[slow];
        fast = v[v[fast]];
    } while (slow != fast);
    slow = v[0];        // or fast = v[0];
    while (slow != fast)
    {
        slow = v[slow];
        fast = v[fast];
    }
    return slow;    // or return fast;
}
```

→ inwardly slow by 1 & fast by 2

→ inwardly slow & fast by 1

(28) Sort 0's &, 1's & 2's :

→ Given an array of 0, 1, & 2 sort it

I/P → {2, 0, 2, 1, 1, 0}

O/P → {0, 0, 1, 1, 2, 2}

(I) Sort it O(nlogn) SC → O(n)

(II) Using Counting Sort O(N)+O(N) → O(N)

→ count no of 0's, 1's & 2's in first pass

→ in second pass simply start writing in an array (same or different array) starting from 0 to 2

(III) Dutch National Flag Algorithm :

→ we consider three pointer low, high, mid

```
void f(vector<int> &v)
{
    int low=0, mid=0, high=v.size()-1;
    while(mid<=high)
    {
        if(v[mid]==0)
        {
            swap(v[mid], v[low]);
            mid++;
            low++;
        }
        else if(v[mid]==1)
            mid++;
        else if(v[mid]==2)
        {
            swap(v[mid], v[high]);
            high--;
        }
    }
}
```

logic: basically mid के left में 0 तक रहेंगे और high से left right में 2 तक रहेंगे।

[0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2]
↑
mid, high (after loop ends)

(29) Find the missing & Repeating Numbers:

→ Given unsorted array of size n, array element in range {1 to n}, one no. in set {1, 2, ... n} is missing & one number is occurring twice. Find these two numbers

→ I/P → {4, 3, 6, 2, 1, 1}
O/P → 5 1
missing repeating

o/p in form of first print repeating no. then missing no.

(1) Using Hashing:

```
int * f(int *a, int n)
{
    static int ans[2];   // or int *ans = new int[2];
    vector<int> v(n+1, 0);
    for(int i=1; i<=n; i++)
        v[a[i-1]]++;
    for(int i=1; i<=n; i++)
    {
        if(v[i]==2)
            ans[0]=i;
        if(v[i]==0)
            ans[1]=i;
    }
    return ans;
}
```

use properly.

TC → O(N) + O(N) = O(2N) ≈ O(N)

SC → O(N)

negation of index approach modifies the array, never use this approach

classmate Date Page

(II) Using Math:

$$s = \frac{n(n+1)}{2}$$

$$s^2 = \frac{n(n+1)(2n+1)}{6}$$

division in C++ is of math, no rule extra

eg: {4, 3, 6, 2, 1, 1}

(i) s - total = (1+2+3+4+5+6) - (4+3+6+2+1+1)

s - total = s - 1 = 4 ⇒ x - y = 4 ——— (I)

s → missing number

(ii) s^2 - total^2 = $(1^2+2^2+3^2+4^2+5^2+6^2) - (4^2+3^2+6^2+2^2+1^2+1^2)$

$s^2 \to p$ = $s^2 - 1^2 = 24$

$x^2 - y^2 = 24$ ——— (II)

Solve these two equation for ans.

```
int *f(int *a, int N)
{   static int ans[2];
    long long int n=(long long int) N;
    long long int s, p, x, y;
    s = (n*(n+1))/2;
    p = (n*(n+1)*(2*n+1))/6;
    for(int i=0; i<n; i++)
    {
        s -= (long long int) a[i];
        p -= (long long int) a[i] * (long long int) a[i];
    }
    x = (s + p/s)/2;
    y = x - s;
    ans[0] = y;
    ans[1] = x;
    return ans;
}
```

Using long long b/c square of no. is large.

TC → O(N)

SC → O(1)

5 101
0 000
5 101 (XOR with 0 = no. itself)

Date
Page

(III) Using XOR

T(→ O(N) + - - O(N) = O(N) (shtmy)
S(→ O(1)

```
int *f(int *a, int n)
{   static int ans[2];
    int xo = a[0];
    for(int i=1; i<n; i++)
        xo = xo ^ a[i];
    for(int i=1; i<=n; i++)
        xo = xo ^ i;
    int s = xo & ~(xo-1);      // Gives rightmost set bit number (eg: 100 ⇒ 4)
    int xo1=0, xo2=0;
    for(int i=0; i<n; i++)
    {
        if((a[i]&s)!=0)  or  if(a[i]&s)
            xo1 = xo1 ^ a[i];
        else
            xo2 = xo2 ^ a[i];
    }
    for(int i=1; i<=n; i++)
    {
        if(i&s)
            xo1 = xo1 ^ i;
        else
            xo2 = xo2 ^ i;
    }
    int c=0;
    for(int i=0; i<n; i++)
    {
        if(xo1 == a[i])
            c++;
    }
    if(c==0)
    {   ans[0] = xo2;
        ans[1] = xo1;
        return ans;
    }
    ans[0] = xo1;
    ans[1] = xo2;
    return ans;
}
```

→ dividing element of array by using s (say s=4 → 100)
eg: 4, 3, 6, 2, 1, 1
4→100, 3→011, 6→110
2→010 → 1→001

xo1	xo2
6, 4	1, 1, 2, 3

→ dividing element from 1 to n in xo1 & xo2

xo1	xo2
6, 5, 4, 6, 4	3, 2, 1, 1, 2, 3

xo1 → 5, xo2 → 1

to detect which is missing no. & which repeating.

(30) Merge Intervals :

→ Given collection of intervals merge all overlapping intervals.

→ I/P → [[1,3], [8,10], [2,6], [15,18]]
O/P → [[1,6], [8,10], [15,18]]

(I) Brute force :

→ first sort the vector
→ check for all intervals that merge with ith interval & merge it all. A linear scan for every interval is done

(II) efficient O(n)

→ sort first
→

```
vector<vector<int>> f(vector<vector<int>> &v)
{
    vector<vector<int>> ans;
    sort(v.begin(), v.end());
    vector<int> ds = v[0];
    for(int i=1; i<v.size(); i++)        //i=0 or i=1
    {
        if( ds[1] >= v[i][0])
            ds[1] = max( ds[1], v[i][1]);
        else
        {
            ans.push_back(ds);
            ds.clear();
            ds = v[i];
        }
    }
    ans.push_back(ds);
    return ans;
}
```

In-place - no extra space other than the given one.

(II) Set Matrix Zeros :

→ Given nxm matrix, if an element is 0, set its entire row & column to 0.

→ I/P → [[1,1,1], [1,0,1], [1,1,1]] O/P → [[1,0,1], [0,0,0], [1,0,1]]

Ask Interviewer

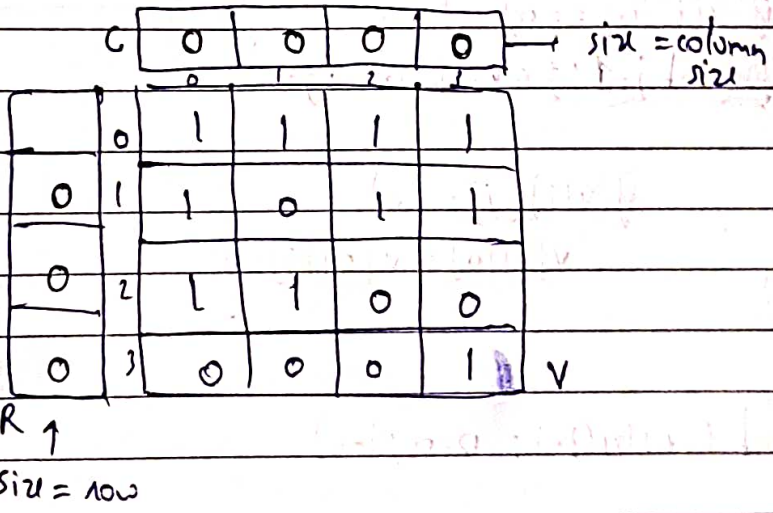
(I) Brute force : O(n x m x (n+m))

→ if you find zero, do all its row & column to -1 (value which can't be in array)

→ at last change -1 to 0.

(II) Two Dummy Array Approach :

→ We take two dummy array.



→ Now iterate whole array, if for any index (R[i]==0 || C[j]==0) then assign zero at that index

TC O(N x m + N x m) → bz we iterate twice, one for assign zero in dummy array & one for iterator to change index value

⇓

O(N x M)

SC O(N) + O(m)

O(N×M)

III Taking two Dummy arrays inside the given 2D Matrix :

1	1	1	1
1	0	1	1
1	1	0	1
0	0	0	1

2 Dummy Array

⇒

0	0	0	1
0	0	1	1
0	1	0	1
0	0	0	1

⇓

0	0	0	1
0	0	0	0
0	0	0	0
0	0	0	0

```
void f(vector<vector<int>> &v)
{  int col=1;
   for(int i=0; i<v.size(); i++)
   {
        if(v[i][0]==0) col=0;
        for(int j=1; j<v[i].size(); j++)
        {
              if(v[i][j]==0)
                  v[i][0] = v[0][j] = 0;
        }
   }

   for(int i=v.size()-1; i>=0; i--)
   {
        for(int j=v[i].size()-1; j>=1; j--)   → उलटा check कर रहे so that ऊपर वे निचे वाले को chang न कर पाए
        {
              if(v[i][0]==0 || v[0][j]==0)
                  v[i][j]=0;
        }
        if (v[i][0   if(col==0)
                  v[i][0] = 0;
   }
}
```

**

	0	1	2	3
0	1	1	0	1
1	1	1	1	1
2	1	0	1	1
3	1	1	1	0

⇒ final.

0	0	0	0
1	0	0	0
0	0	0	0
0	0	0	0

~~col~~ col variable इस लिए ले रहे क्योंकि,
v[0][2] की जगह से v[0][0] में जाएगा assign होगा 0 (dummy array accessed)
→ और जब हम check करेंगे तो देखेंगे v[0][0] == 0 है जिससे पूरा (col=0) zero हो जाएगा

→ इसीलिए column 0 के लिए एक अलग से col variable बना लिए।

(32) Pascal Triangle:

```
          1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
```

row=5 col=3

(I) To calculate element at R^{th} row & C^{th} column

TC → O(N)
SC → O(1)

$$^{R-1}C_{C-1}$$

eg: row=5, col=3
$^4C_2 = 6$

calculation of this also takes O(n) time

(II) To print n^{th} row of PT:

→ we use this formula

eg: n=5
O/P → 1 4 6 4 1 → Calculating every index values takes O(n) time, in total ~~O(n+1)~~ time will ~~take~~ ~~be equal to~~ ~~than~~ ~~O(n×n)~~ $O(n^2)$

~~TC = O(~~

~~cout<<~~ next_permutation(v.begin(), v.end())
↓
modifies the vector to next permutation

R-1 C C-1
R=5

eg: 1 4 6 4 1
$^4C_0$ $^4C_1$ $^4C_2$ $^4C_3$ $^4C_4$
1 $\frac{4}{1}$ $\frac{4\times3}{2\times1}$ $\frac{4\times3\times2}{3\times2\times1}$ $\frac{4\times3\times2\times1}{4\times3\times2\times1}$

$$^nC_r = \frac{n-1}{r-1} \cdot {}^{n-1}C_{r-1}$$

$$^4C_3 = \frac{4}{3} {}^3C_2$$
$$= \frac{4}{3} \times \frac{3}{2} \times \frac{2}{1}$$

```
int main()
{
    int n;
    ~~cin>>~~ cin>>n;
    int ans=~~n-~~1;
    cout<<ans<<" ";
    ans=n-1;
    for(int i=2;i<=n;i++)
    {
        cout<<ans<<" ";
        ans=ans*(n-i);
        ans=ans/(i);
    }
}
```

TC → O(N)
SC → O(N) (if string values will be printing)

(III) Print all n rows of PT: TC → O(n²)
SC → O(n²)

```
~~#~~ vector<vector<int>> f(int n)
{   vector<vector<int>> v(n);
    for(int i=0;i<n;i++)
    {
        v[i].resize(i+1);
        v[i][0]=v[i][i]=~~0~~1;
        for(int j=1;j<i;j++)
            v[i][j]=v[i-1][j-1]+v[i-1][j];
    }
} return v;
```