(14) Maximum Consecutive 1's in a Binary Array :

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

O/P → 4 | 0 | 3

Naive → O(n²)

efficient → O(n)

```
int f(int a[], int n)
{
    int res=0; c=0;
    for(int i=0;i<n;i++)
    {
        if(a[i]==0)
            c=0;
        else{
            c++;
            res=max(res,c);
    }
    return res;
}
```

(15) Maximum Sum SubArray :

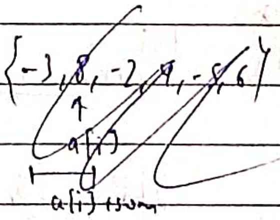
I/P → {-3, 8, -2, 4, -5, 6}

O/P → ~~3~~ 11

✓ a) Kadane's Algo O(n)

b) Naive O(n²)

✓ c) efficient O(n)



✓ ** push this

```
int maxSum(int a[], int n)
{
    int res=a[0], sum=a[0];
    for(int i=1; i<n; i++)
    {
        sum=max(a[i], a[i]+sum);
        res = max(res, sum);
    }
    return res;
}
```

(16) Maximum length Even-Odd Subarray :

→ pairs can be even-odd / odd-even both valid

→ I/P → {10, 12, 14, 7, 8} O/P → 3 (e o e) | I/P {7, 10, 13, 14} (o e o e) O/P → 4 | I/P {4, 6, 8} (e) O/P → 1

Naive → O(n^2) → break is used (hint)

efficient → O(n).

```
int maxlength(int a[], int n)
{
        int res=1, c=1;
        for(int i=1; i<n; i++)
        {
              if((a[i]%2==0 && a[i-1]%2!=0) ||
                 (a[i]%2!=0 && a[i-1]%2==0))
              {
                    c++;
                    res=max(res,c);
              }
              else
                    c=1;
        }
    return res;
}
```

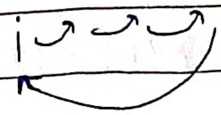
★★★ (17) Maximum Circular SubArray Sum :

I/P →	{5,-2,3,4}	{2,3,-4}	{8,-4,3,-5,4}	{-3,4,6,-2}
O/P →	12	5	12	10

a) Naive Solution $O(N^2)$

n=4 5, -2, 3, 4 → to again go to index 0 we do $(i\%n)$ 4%4=0

0 1 2 3



```
int f(int a[], int n)
{
    int ans = a[0];
    for( int i=0; i<n; i++)
    {
        sum = a[i], res = a[i];
        for( j=1; j<n; j++)
        {
            sum += a[(i+j)%n];
            res = max(res, sum);
        }
        ans = max(ans, res);
    }
    return ans;
}
```

for i=0

5, -2, 3, 4
0 1 2 3
i=0

we ~~[illegible]~~ go through 5→-2→3→4

for i=1
we go -2→3→4→5

for i=2
we go 3→4→5→-2
Similary other

★ for every i we do +1 through j loop to access index in circular manner

★★

b) efficient $O(n)$

logic

SubArray → Normal SubArray, Circular SubArray

Step I We will find max SubArray Sum from Normal SubArray say res1
Step II Circular " say res2
Step III Then Do Max of results from Step I & Step II
ans = max(res1, res2);

One Concept for Step II Calculation

a[] = {8, (-4, 3, -5), 4}

we know max Circular SubArray Sum = ~~8~~ 4+8 = 12

Remaining elements are the minimum SubArray Sum of Normal SubArray

→ So basically we find (total sum of Array) - (minimum SubArray)

prefer this *

eg: To find MSS

Method (I) a[] = {8, -4, 3, -5, 4} ⇒ Multiply -1 ⇒ {-8, 4, -3, 5, -4}

⇓ Now find max SubArray Sum

return -6 ⇐ 6

Method (II)

```
int sum = a[0], ans = a[0];
~~Kada~~ for(int a[], int n)
{
        for( i → 0 to n-1) {
        ~~int~~ sum += a[i];
            ~~ans~~ res = max(res, sum);
            if (sum>0)
                sum=0;  }
}
return ~~sum~~ res.
```

```
int CirculanSum( int a[], int n)
{
        int ans1 = f(a,n);      → Normal SubArray Sum
        if(ans1<0)
            return ans1;        ] → when all elements are -ve

        int total=0;
        for( int i=0; i<n; i++)
        {
                total += a[i];
                a[i] = - a[i];
        }
        int  ans2 = f(a,n);     → " (Minimum SubArray Sum)
        ans2= total+ ans2;
        return  max(ans1, ans2);
}

int f(int a[], int n)
{
        int  sum=a[0], res=a[0];
        for(int i=1; i<n; i++)
        {
                sum= max (a[i], a[i]+sum);
                res = max (res, sum);
        }
        return res;
}
```

(Moore's Voting Algo)

(18) Majority Element I

→ Given array of size n, find majority element.

→ Only 1 soln max^m possible.

majority element ↓ element that occurs more than ⌊n/2⌋ value.

Naive → $O(n^2)$ Brute force $O(n^2)$ | HashMap $O(n\log n)$, S(→ O(n))

efficient → $O(n)$

I/P → [8,3,4,8,8]	{3,7,4,7,7,5}	{3 4 4}
O/P → 8	-1	4

```
int MajorityElement( int a[], int n)
{
        int c=0, num;
        for(int i=0;i<n;i++)
        {
                if(c==0)
                        num=a[i];
                if(num=a[i])
                        c++;
                else
                        c--;
        }
        int c1=0;
        for(int i=0;i<n;i++)
        {
                if(a[i]==num)
                        c1++;
        }
        if(c1>n/2)
                return num;
        return -1; → when no ME present
}
```

Moore Voting Algo

→ num is our majority element, but only when there is ME exist.

→ counting num occurrence

here count (c) becomes zero, and also num is assigned a new value

eg: 7, 7, 5, 7, 5, 1 | 5, 7 | 5, 5, 7, 7, | 5, 5, 5, 5

जब भी count=0 होगा हम new num(ME) assign करो

$c = \cancel{0}\,\cancel{1}\,\cancel{2}\,\cancel{1}\,\cancel{2}\,\cancel{1}\,\cancel{0}\,\cancel{1}\,\cancel{0}\,\cancel{1}\,\cancel{2}\,\cancel{1}\,\cancel{0}\,\cancel{1}\,\cancel{2}\,\cancel{3}\,4$

$num = \cancel{7}\,\cancel{5}\,\cancel{5}\,5$

इस part को ध्यान से देखो

{7, 7, 5, 7, 5, 1}

→ every time num value change we consider that element as our new ME.

we consider 7 as our layered element (ME)
here (majority == minority) they cancel each other

(19) Majority Element II :

→ Given array find ME (element that occurs $> \lfloor \frac{n}{3} \rfloor$ times)
→ Remember → at max only two sol^n are possible.

Naive → $O(n^2)$ SC → $O(1)$ | Hashmap → $O(n\log n)$ SC → $O(n)$
efficient → $O(n)$

eg: {1, 4, 5 | 4, 4, 5, 5, 5, 3, 4, 7} n = 11

c1, c2 = 0

$c1 = \cancel{0}\,\cancel{1}\,\cancel{0}\,\cancel{1}\,\cancel{2}\,\cancel{3}\,\cancel{2}\,1$

$c2 = \cancel{0}\,\cancel{1}\,\cancel{0}\,\cancel{1}\,\cancel{2}\,\cancel{1}\,\cancel{2}\,1$

$num1 = \cancel{1}\,\cancel{1}\,5$

$num2 = \cancel{1}\,4$

Q. in {1, 4, 5}
$n = 3/3 = 1$
no element occurs more than 1

5, 4 are required ME

→ Using num1, num2 we keep track of two ME.

```
class Solution{
 public:
    vector<int> f(vector<int> &a)
    {
         int c1=0, c2=0 ,num1=-1, num2=-1;
         for( int i=0; i<a.size(); i++)
         {
              if( a[i]==num1)  c1++;
              else if(a[i]==num2)  c2++;
              else if(c1==0)
              {
                   num1=a[i]
                   c1++;          // or c1=1;
              }
              else if(c2==0)
              {
                   num2=a[i]
                   c2++;          // or c2=1;
              }
              else
              {
                   c1--; c2--;
              }
         }
         c1=c2=0;
         for(int i=0; i<a.size(); i++)
         {
              if( a[i]==num1)
                   c1++;
              if( a[i]==num2)
                   c2++;
         }
```

→ calculating occurrenc
of num1 &num2

```
    vector<int> v;
    if(c1 > a.size()/3)
        v.push_back(num1);
    if(c2 > a.size()/3)
        v.push_back(num2);

    return v;
  }
};
```

20 Minimum Group flip to make Binary Array Same :

Brute force → O(n²)

efficient O(n)

I/P → 0,0,1,1,0,0,1,1,0
0 1 2 3 4 5 6 7 8

O/P → flip 2 to 3
flip 6 to 7

Concept

→ Group of 0 & 1 differ by atmost 1. (either same or differ by one)

① Group differ by 1

11000111001 → count of 1 group = 3 (3-2) = 1
" " 0 " = 2

0101001111100 → count of 1 —— = 3 (4-3) = 1
0 —— = 4

When a[0] & a[n-1] are same then group differ by 1

② Group counts same

001111000011 → count of 0 == count of 1 == 2

10101100110 → —— = 4

When a[0] & a[n-1] are different then group count is same.

Remember we will solve by taking second different group ie say 11000101100111 for able to ~~solve~~ write code for both case.

SDG

```
void f(int a[], int n)
{
        for(int i=1; i<n; i++)
        {
                if(a[i]! a[i-1])
                {
                        if(a[i] ==a[0])
                            cout<< i-1 <<endl;
                        else
                            cout<<"from "<<i<<" to ";
                }
        }
        if(a[n-1]! a[0])
            cout<< n-1;          ] when both grp count same
}                                      110001011000
```

Brute force :

Sliding Window

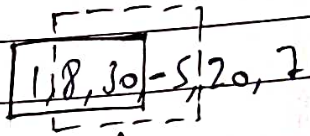
- Problems / Concept

(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 res=0;
        for(int i=0; i<k; i++)
                res += a[i];
        int sum=res;
        for(int i=k; i<n; i++)
        {
                sum += a[i]-a[i-k];
                res = max(res, ~~max~~ sum);
        }
        return res;
}
```

I/P → {1,8,30,-5,20,7}, k=3
O/P → 45



window slides by 1.

(22)