Array - I Linear data structure.

Swap two elements

```
swap(a, b)
    a = a + b
    b = a - b
    a = a - b
```

Q. Rotate the array k times

Ex:

	1	2	3	4	5
k = 1	5	1	2	3	4
k = 2	4	5	1	2	3

```
void rotate(arr, k)                    O(k*n)
for(i = n-2 to 0)                      temp = arr[n-1]
    arr[i+1] = arr[i]
a[0] = temp
```

n = 6

k=2	5	6	1	2	3	4	1	2	3	4	5	6
k=3	4	5	6	1	2	3	3	4	5	1	2	
k=4	3	4	5	6	1	2						

```
if(k > n/2)
for(i = 0 to k)        X
    swap(arr[i], arr[(i+k)%n])
```

Reverse:

a = [2, 3, 5, 7, 11, 13, 17, 19, 23]

reverse(a) = [23, 19, 17, 13, 11, 7, 5, 3, 2]

k = 3 [17, 19, 23, 2, 3, 5, 7, 11, 13]

```
rev(arr, k)                    tc = O(n+k+(n-k))
rev(arr, 0, n-1)                  = O(2n) = O(n)
rev(arr, 0, k-1)               sc = O(1)
rev(arr, k, n-1)
```

Dynamic Arrays → Size is flexible. (Size is fixed.)

- built on top of static array.

Q: Given an array, calculate the sum of els in a given range, without recalculating the sum for each query.

(-5, 10, 20, 40, 50, -10, 10, -90, -20, -10)

```
def rollarr(arr)
    parr=[0], parr[0] = arr[0]
    for(i: 1 to n-1)
        parr[i] = arr[i] + parr[i-1]

def sumrange(arr, parr, l, r)
return parr[r] - parr[l-1]
```

Generalized: Cumulative sum from start till a particular index.

Prefix sum with O(1) update: update the input.

```
for(i: 1 to n-1)
    a[i] += a[i-1]
```

tc: O(n+q)
sc: O(1)

Q: Given integer array of size n, q queries. For every query find sum of even index elements from l to r.

ex:

	0	1	2	3	4	5
	2	3	1	6	4	5
psum:	2	5	6	12	16	21

	0	1	2	3	4	5
Pevsum:	2	2	3	3	7	7

Queries

l	r	
1	3	1
2	5	5
0	4	7

```
P[0]=0
for(i: 1 to n-1)
    if(i%2) P[i] = P[i-1] + a[i]
    else    P[i] = P[i-1]
```

Q-1 Given string of lowercase letters, return the count of (i, j) s.t. $i < j$ and $s[i] = 'a'$ and $s[j] = 'g'$

- Brute force
- Can we find a better technique.

0	1	2	3	4	5	6
a	b	c	g	a	d	g
1	1	1	1	2	2	2

→ for every g, count # of a before it.

$(0,3), (0,6), (4,6)$ → 1+2

Q-1

0	1	2	3	4	5	6
a	c	g	d	g	a	g
1	1	1	1	1	2	2

$(0,2), (0,4), (0,6), (5,6)$

$1 + 1 + 2 = 4$

Q-2

0	1	2	3	4	5	6	7
b	c	a	g	g	a	a	g
0	0	1	1	1	2	3	3

$2 + 3 = 5$

$(2,3), (2,4), (2,7), (5,7), (6,7)$ → 5

Brute Force : Do exactly what the question asked to do.
- without any thought of optimization.

TC : $O(n^2)$

SC : $O(1)$

```
ans = 0
for (i = 0 to n-1)
    for (j = 0 to n-1)
        if (i < j && s[i] == 'a' && s[j] == 'g')
            ans++;
return ans;
```

How to optimize ? → Try to convert part of linear code to a data structure.

```
ans = 0
for(i = 0 to n-1)                          // Count the # of 'g' after s[i];
    for(j = 0 to n-1)
        if(i < j && s[i] == 'a' && s[j] == 'g')
            ans++;
return ans;
```

```
psum = [];   // replace with an array.
if(s[n-1] == 'g') psum[n-1] = 1;
else psum[n-1] = 0;                        Tc : O(n)
                                           Sc : O(n)
for(i = n-2 to 0)
    if(s[i] == 'g') psum[i] = (1 + psum[i+1]);
    else psum[i] = psum[i+1];
```

```
ans = 0, cnt = 0                           Tc : O(n)
for(i = n-1 to 0)                          Sc : O(1)
    if(s[i] == 'g') cnt++;                 // Calculate the data
    if(s[i] == 'a') ans += cnt;            // use the data
return ans;
```

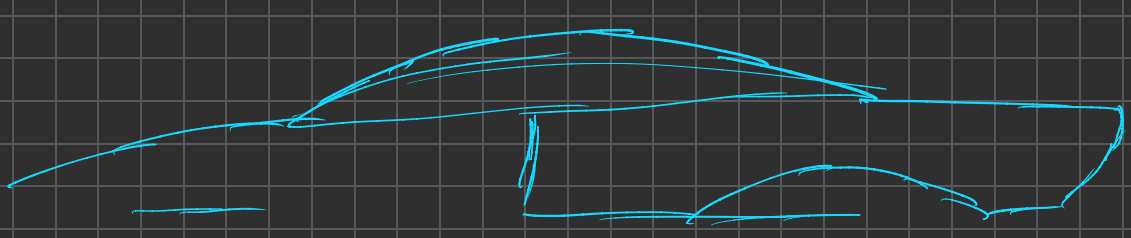
→ Carry Forward Technique.

	0	1	2	3	4	5	6	7	8
s :	a	c	g	a	g	k	a	g	c

cnt : 0 → 1 → 1 → 2 → 3

ans : 0 → 1 → 3 → 6

Subsequence → Continuous subsequence of an array.



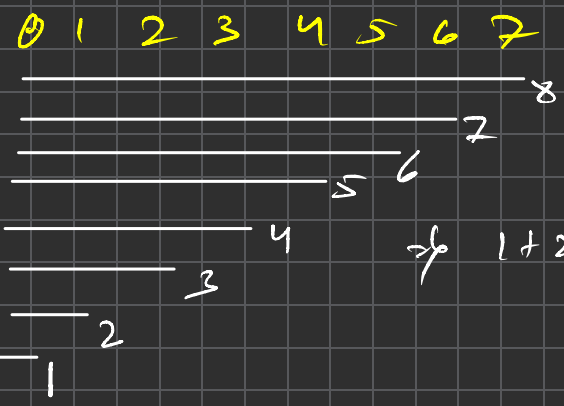
6 + 7

0, (0,1) (0,2), (0,3), (0,4), (0,5), (0,6)

1 2 3 4 5 6 7

Q) Sum of all Subarray Sums -

Q) Total # of subarrays in an array of size n.



$\Rightarrow 1+2+3+\dots n = \frac{n(n+1)}{2}$.

Sum of all subarray sums -

Brute Force -

```
for(i = 0 to n-1)
    for(j = i to n-1)
        sum = 0
        for(k = i to j)
            sum += a[k];
        ans += sum;
```

Subarray Sum.

Prefix Sum -

```
for(i = 0 to n-1)
    for(j = i to n-1)
        if (i == 0) sum += P[j];
        else sum = P[j] - P[i-1];
        ans += sum;
```

→ use prefix sum to calculate subarray sum.

```
// Prefix Sum -
P[0] = a[0]
for(i = 1 to n-1)
    P[i] = P[i-1] + a[i];
```

Cummy Trick → Calculate & use without storing in memory.

```
for(i:= 0 to n-1)
  sum=0
  for(j:= i+1 to n-1)
    sum += a[j];
    ans += sum;
return ans;
```

ex → 3, 2, 5

	0	1	2

i,	j,	sum,	ans
0		0	0
	0	3	3
	1	5	8
	2	10	18
1		0	
	1	2	20
	2	7	27
2		0	
	2	5	32

Contribution Technique → If an element is contributing multiple times in answer, then answer can be calculated as

$$ans = \sum_{\forall i} \text{occurrence}[a[i]] * a[i]$$

# of subarrays with index i →

start positions → 0 → i → (i+1)

end positions → (n-i)

```
contribution[i] = a[i] * (i+1)(n-i);
for(i:= 0 to n-1)
  contribution[i] = a[i] * (i+1) * (n-i);
  ans += contribution.
return ans;
```

Q- Count of Subarrays of length K -

Brute Force -

```
ans = 0;
for(i = 0 to n-1)
    for(j = i to n-1)
        if(j - i + 1 == k)
            ans++;
```

Sliding Window -

```
ans = 0; i = 0, j = 0;
while(i <= n-k)
    if(j - i + 1 < k) j++;
    if(j - i + 1 == k) j++, ans++;
return ans;
```

# of Sub-arrays of size K in an array of size n = n-k+1

Q- Minimum Sum Sub-array -

Brute Force -

```
for(i = 0 to n-k)
    sum = 0;
    for(j = i to i+k-1)
        sum += a[j];
    ans = min(ans, sum);
```

T(n) = O(n²)
S(n) = O(1)

Sliding window -

```
i = 0, j = 0, sum = ans = Int.Min;
while(i < n-1)                    T(n) = O(n)
    sum += a[j];
    while(j - i + 1 > k)          S(n) = O(1)
        sum -= a[i];
        i++;
    if(j - i + 1 == k)
        ans = min(ans, sum);
    j++;
return ans;
```

Array-3

Q- Maximum sum subarray -

Brute -

```
for(i:=0 to n-1)
    for(j:= i to n-1)
        sum=0
        for(k:= i to j)
            sum += a[k]
        if(sum > ans) ans:= sum
return ans
```

TC → $O(n^3)$
SC → $O(1)$

Carry Forward -

```
for(i:= 0 to n-1)
    sum=0
    for(j:= i to n-1)
        sum = a[j];
        ans = max(ans, sum);
return ans;
```

TC → $O(n^2)$
SC → $O(1)$

→ Kadane's Algorithm →

7 5 -3 4

12

9

13

3, 5, -9 short-mein

8

-1

→ overall negative in right, don't consider this sub-array.

For any subarray

n = 3

0	1	2	3	4	5	6	7	8	9
5	2	-3	-8	6	1	-5	11	0	-3

sum → 5 7 4 -4 6 7 2 13 0 -3

ans → 5 7 7 7 7 7 7 13 13 10

ans = 13

Code → st=0, l=0, h=0, ans = -∞, sum=0;

for(i=0 to n-1)

sum += a[i];

if(sum > ans)

h = i; l = st;

ans = sum;

if(sum < 0) sum=0, st = i+1;

return ans

TC → O(n)
SC → O(1)

To keep track of temporary start.

Q → Given a row & col wise sorted matrix, check if given element K is present or not.

Always start with top right or bottom left position.

ascending → default.

Increasing

	0	1	2	3
0	-5	-2	1	13
1	-4	0	3	14
2	-3	2	6	18

outside, ans = false;

K = 5

while(0 ≤ i < n && 0 ≤ j < n)

if(a[i][j] == K)

return true;

if(a[i][j] < K)

i++; // go down

else j--; // go left

return false

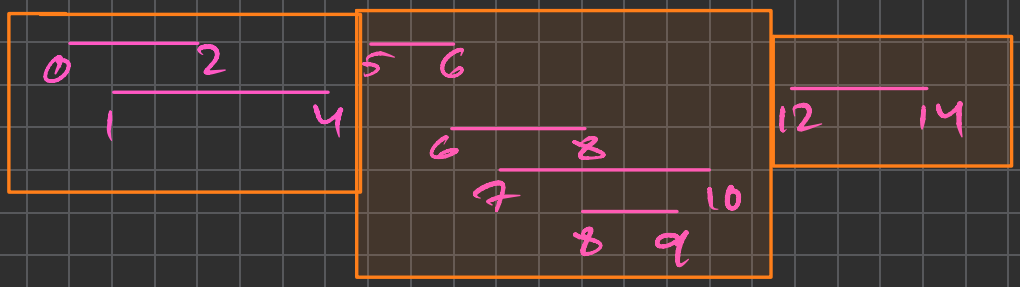
TC → O(n+m)
SC → O(1)

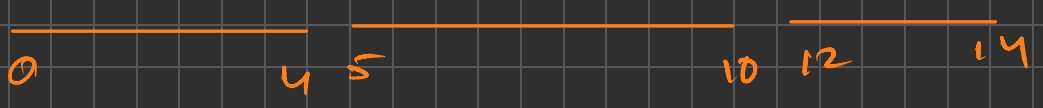
Start from bottom left ↓

```
while( 0 <= i < n && 0 <= j < m )      TC → O(n+m)
    el = a[i][j];                      SC → O(1)
    if( el == k ) return true;
    if( el < k ) j++;
    else i--;
return false;
```

Q: Given a list of intervals, sort it w.r.t start time.
- Merge all overlapping intervals.
- Return the sorted list of non-overlapping intervals.

```
bool overlap( i1, i2 )
    if( (i1.start <= i2.start <= i1.end)
        or
        (i1.start <= i2.end <= i1.end) )
        return true;
    return false;
```

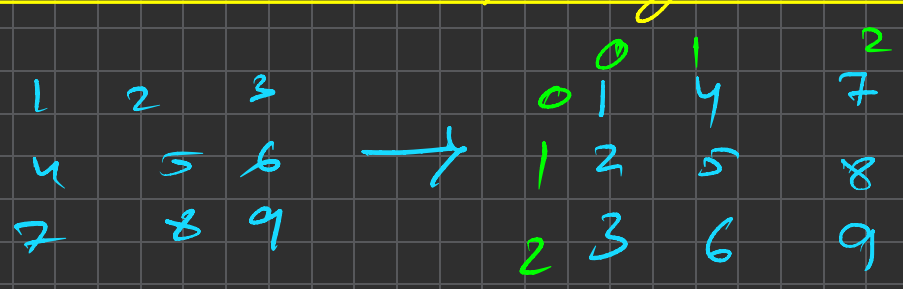





```
for( i = n-2 to 0 )
    if( overlap(a[i+1], a[i]) )
        l = min(a[i].first, a[i+1].first)
        r = max(a[i].end, a[i+1].end)
        a.pop(), a.pop();
        a.insert([l,r]);
```

TC → O(n)
SC → O(1)

Q: Find the transpose of the given matrix :-




```
if(i != j)
    swap(a[i][j], a[j][i])
```

Q: Rotate the sq matrix by 90° clockwise :-

Ex:

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