W2

P1 **Prefix - Sum concept :-**

Find answer to Q query on a static array A of size N :

? L, R → Find the sum of elements in the range L to R in the array.

A :	4	2	3	1	-5	6
	0	1	2	3	4	5

? 24 = 3+1+(-5)

? 04 = 4+2+3+1+(-5)

for looping from L to R for each query → O(N)
∴ T = O(Q·N)

Q how to do better than this ?

Let us create another assay P[n]

$$P[i] = \sum_{j=0}^{i} A[j]$$

$$P[i+1] = P[i] + A[i+1]$$

⇒ P[i] = A[i] + P[i-1]

P[n] array can be created in O(n)

A :	4	2	3	1	-5	6

P :	4	6	9	10	5	11

now

? 24 = P[4] - P[1]

? LR = P[R] - P[L-1] | for L=0 ; ?0R = P[R]

Now, Time complexity = $O(N) + O(Q)$
$= O(N+Q)$ ——.

P2 Partial Sum concept :- (Range update)

Given an array A of size N, initially all 0s.
There are Q queries of the form:
+ L R X → Add X to all the index in the range L to R.
After all the queries You need to find the final array.

eg:

A	0	0	0	0	0	0
	0	1	2	3	4	5

+ 2 4 1

A	0	0	1	1	1	0

+ 1 5 3

A	0	3	4	4	4	3

+ 0 2 2

A	2	5	6	4	4	3

} Final array

One obv. way is loop over L to R ⇒
Time = $O(Q \cdot N)$ } not good ——.

↑ Brute force, we want to do better than this.

Claim

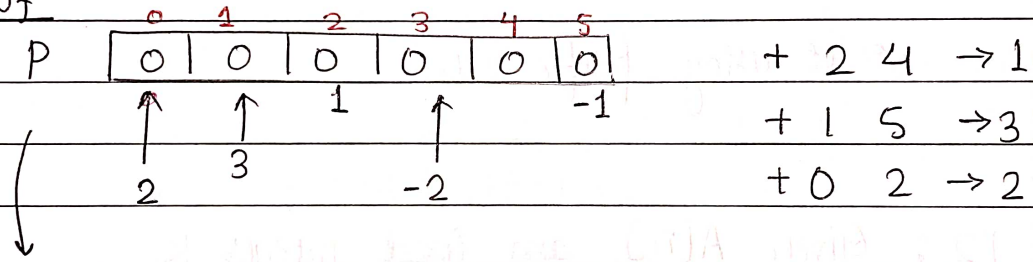
Step 1: Create a new array P of size N

Step 2: for each query + L R X
do
→ P[L] += X
→ P[R+1] -= X

Step 3: after all query, build prefix sum array of P (in place)

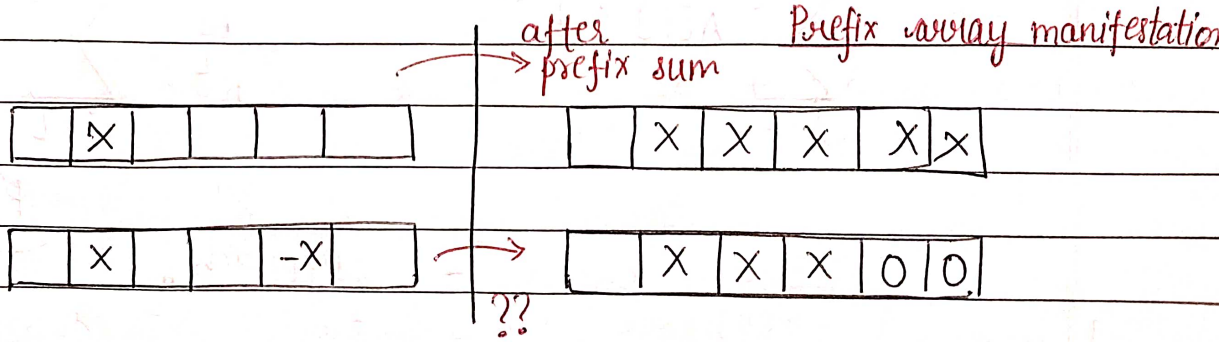
Final P u get is the Answer.

Proof



Q Why is it working?

Prefix sum contribution :-



This reduces our Time
Time complexity = $O(N+Q)$

P1: Given A[N]
for Q queries ; take modulo 10^9+7
? L R = $A[L] + A[L+1]*2 + \dots + A[R]*(R-L+1)$

soln:
$$\sum_{i=L}^{R} A[i]*(-L+i+1)$$

$$= \sum_{i=L}^{R} A[i]*i \;+\; (1-L)\sum_{i=L}^{R} A[i]$$

solve using prefix sum.

P2: Given A[n] and fixed integer K
Find for Q queries
? L R → take modulo 10^9+7

$$A[L] + A[L+1]*K + \dots + A[R]*(R-L)$$

soln:
$$\sum_{i=L}^{R} A[i]*K^{(i-L)} = \frac{1}{K^L}\sum_{i=L}^{R} A[i]*K^i$$

$\sum_{i=L}^{R} A[i]*K^i$ → $P[R] - P[L-1]$

approach.

$\sum_{i=0}^{(R-L)} A[i+L] * K^i$ → cannot be precomputed as it is dependent on L

$$Ans = \left(\frac{\sum_{i=L}^{R} A[i] * K^i}{K^L} \right) \bmod (1e9+7)$$

v prefix

powers

if (L != 0)

$$= \left(\frac{v[R] - v[L-1]}{powers[L]} \right) \bmod (1e9+7)$$

$\left(\frac{a}{b}\right)$ % mod

(ab^{-1}) % mod

if (L = 0)

$= v[R] \bmod (1e9+7)$

Note:

(ab^{-1}) % mod $= [(a \% mod) * (b^{mod-2} \% mod)]$ % mod

(mod: prime)

P3: Given an arr[] of n integers, inially all 0's
Process q queries of the form

+ A D L R → Add the AP of first term A & CD = D to the range LR

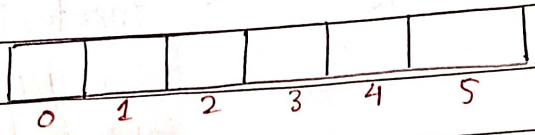
arr[L] += A

arr[L+1] += A+D

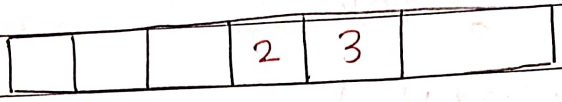
arr[L+2] += A+2D ---- arr[R] += A + (R-L)*D

Find the final array after all queries modulo $10^9 + 7$.

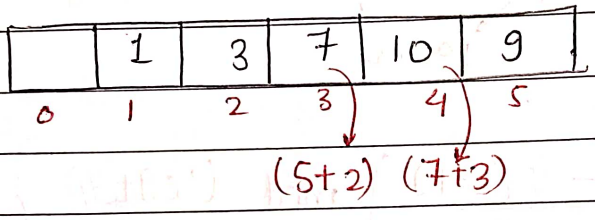
Example:



Q1: +, 2, 1, 3 4 → 2 3 4 - - - (AP)



Q2: +, 1 2, 1 5 → 1 3 5 7 9 - - (AP)



sol^n: Looking at contribution carefully
What are we adding at each index?

+ A D L R.

↓

arr[i] += A + D * (i - L) ; (i ≥ L)

= (A - DL) + i * D

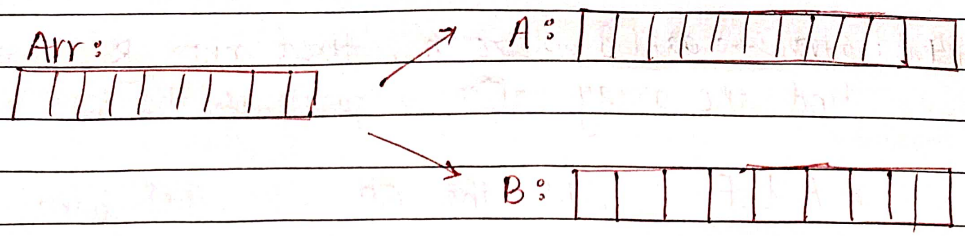
(A - DL): constant for a query

i * D: Dependent on position. → (??)

can be done through partial sum

we need to make it independent of position to apply partial sum technique.

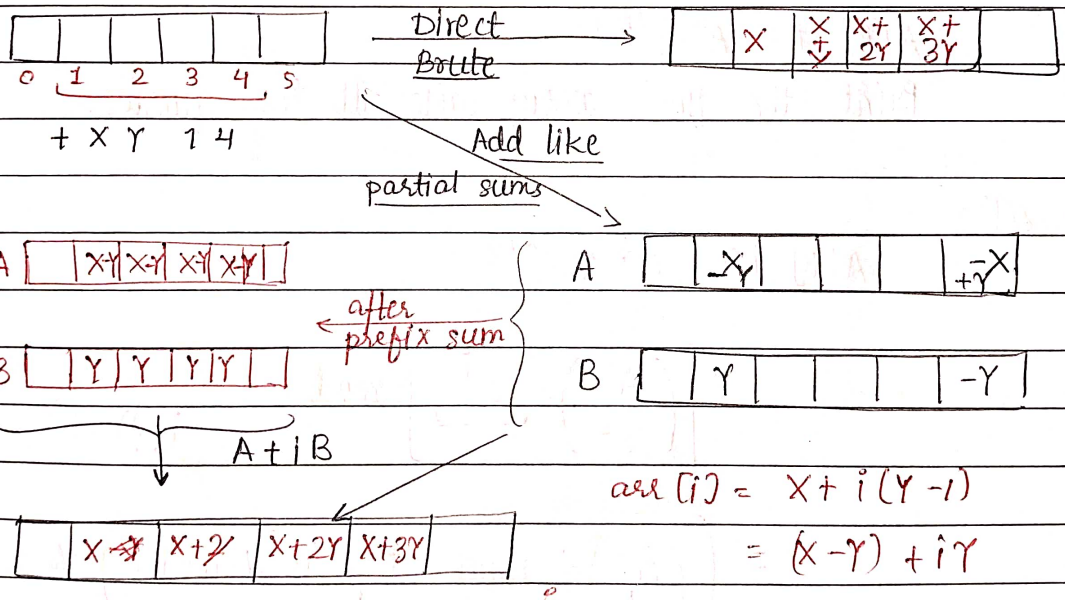
arr[i] += (A + DL) + (iD)



$arr[i] = A[i] + B[i] * i$

$arr[i] = \underbrace{(A - LD)}_{\text{make } A[i]} + \underbrace{D * i}_{\text{make } B[i]}$

let's look at an example



$arr[i] = X + i(Y - 1)$

$= (X - Y) + iY$

Hence,

we will process all queries first, i.e, we add respective arrays A and B.

After this we make prefix sum of A and B, then combine them to get arr[i] values.

P4 : Given → arr[] → size n, fixed num = R
Find the array after q queries of the form:

+ A L R → Add the GP with first term A and common ratio r to the range L to R.

i.e,

A[L] += A

A[L+1] += A*r

A[L+2] += A*r²

⋮

A[R] += A*r^(R-L)

Print the final array after all the queries.

Sol^n:

$$A[i] \mathrel{+}= A * r^{(i-L)}$$

$$= \left[\left(\frac{A}{r^L}\right) \cdot r^i\right] \% \text{mod}$$

(??)

$$= \left(\left(\frac{A}{r^L}\right) \% \text{mod} * (r^i \% \text{mod})\right) \% \text{mod}$$

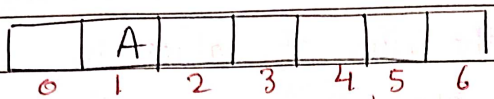
powR[L] powR[i]

((A % mod) * (powinverseR[L] % mod)) % mod

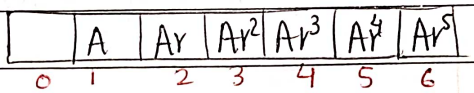
((A % mod) * modRinverse[L]) % mod

This cannot be expressed as sum of two or more arrays.

Lets try to tweek the idea of prefix sum



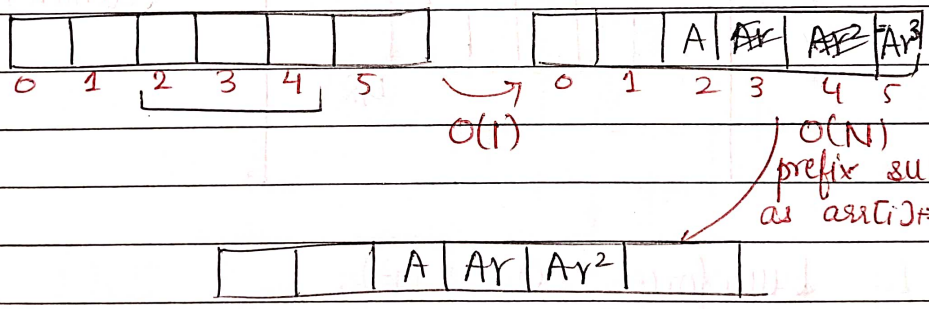

```
for (int i=1 ; i<n ; i++)
    arr[i] += arr[i-1] * r
```



for L → R

add A to (L) and $-(Ar^{R-L+1})$ to (R+1)

for + A 2 4



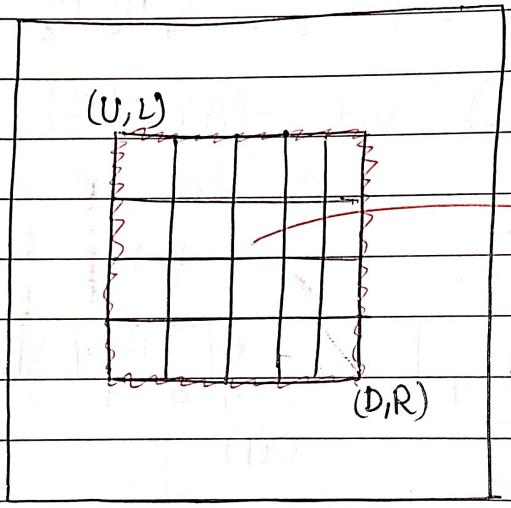
we will process the sum & then take prefix sum (with multiplication by r) to get final answer.

P3 2D Prefix sum :-

Given a 2D matrix Arr[][], with values in each cell
There are Q queries of the form

? L R U D : Find the sum of values in rectangle formed by
$U <= i <= D$ and $L <= j <= R$

i → index of row
j → index of column



By bruteforce : $O(Q * N^2)$

Optimization 1 :

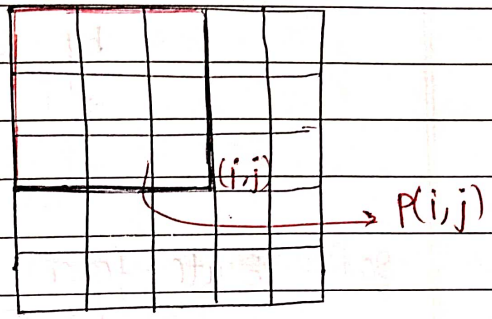
We can form prefix sum arrays for each row,
the call prefix sum arrays (D-U) times to get the answer.
That would be

Time = $O(\max(N^2, Q * N))$

Optimization 2 :

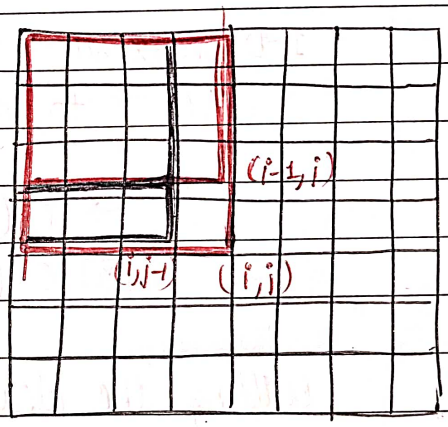
The idea of 2D prefix sum

P[i][j] → sum of top left corner to (i,j) rectangle.



We will build this in row major order.

$$P[i][j] = P[i][j-1] + P[i-1][j] - P[i-1][j-1] + arr[i][j]$$

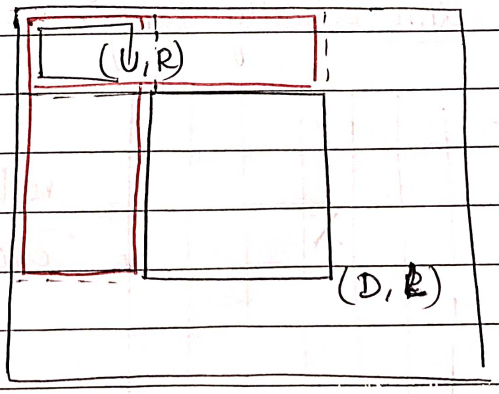


Time complexity

$O(M * N)$

$$ans = P[D][R] - P[D][L] - P[U][R] + P[U][L]$$

$O(1)$ per query.



Hence,

Time = $O(N \cdot M + Q)$

P4: 2D Partial Sum

Given a 2D matrix Arr[][], with 0 as initial value in each cell. There are Q queries of the form:

+ L,R,U,D,X : Add X to the cell values in rectangle formed by $U <= i <= D$ and $L <= j <= R$
→ i index of row, j is index of column.

Soln: Brute-force → O(Q*N*M)

M = no. of rows
N = no. of ele. in each row

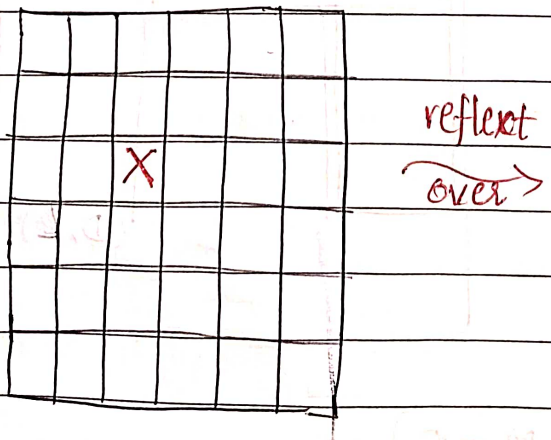
1-D prefix sum

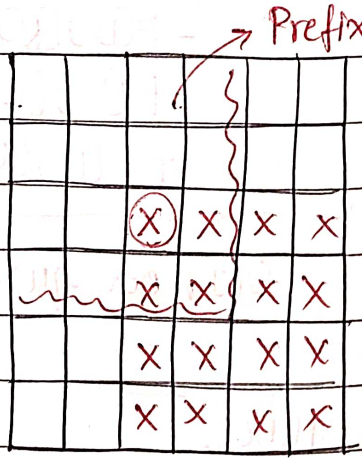
Add X to arr[i][L] & -X to } for i → U to D

→ O(N*M + Q*M)

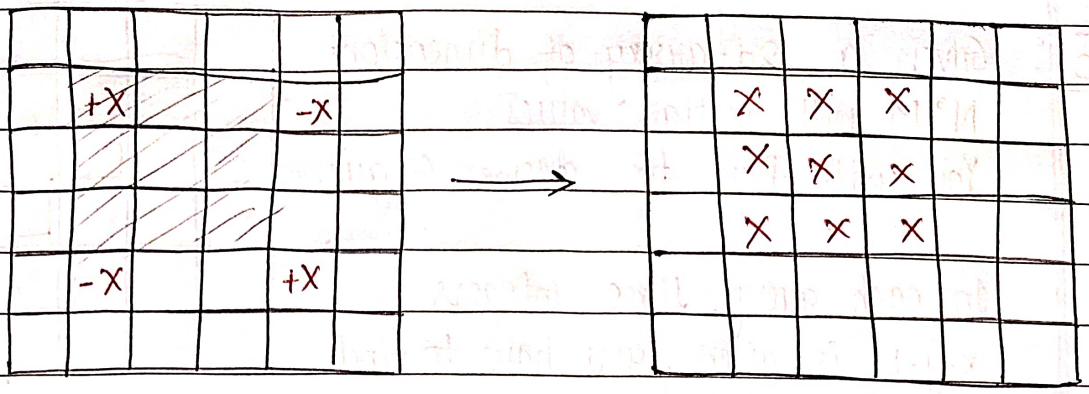
or O((N+Q)M)

Building the idea of 2D partial sum, what is Prefix 2D Sum actually Doing?





Date __/__/____



arr [U] [L] += X

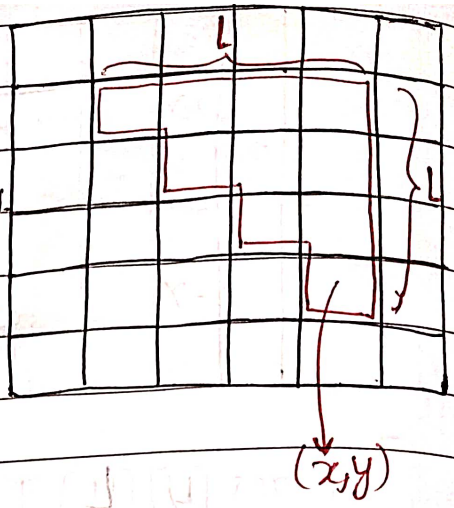
arr [D+1] [L] -= X } if (D+1 != m)

arr [U] [R+1] -= X } if (R+1 != n)

arr [D+1] [R+1] += X } if (D+1 != m && R+1 != n)

↳ O(N · M)

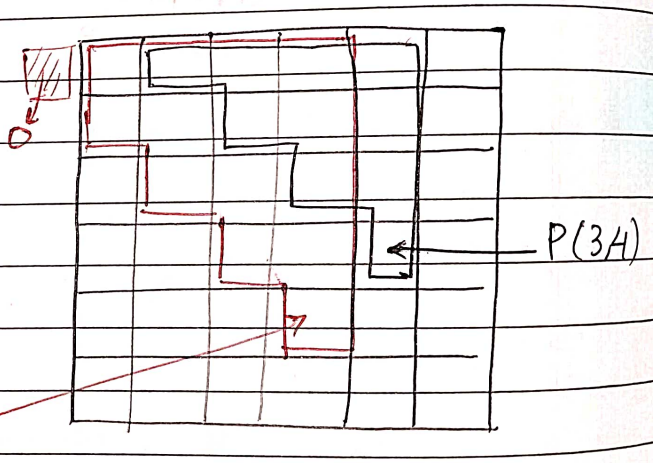
P5: Given a 2d-array of dimensions N*M with initial values
You will have to answer q queries.



In each query three integers x,y,L is given, you have to find sum of triangle given as per the illustration.
Calculate all the values modulo 10^9+7.

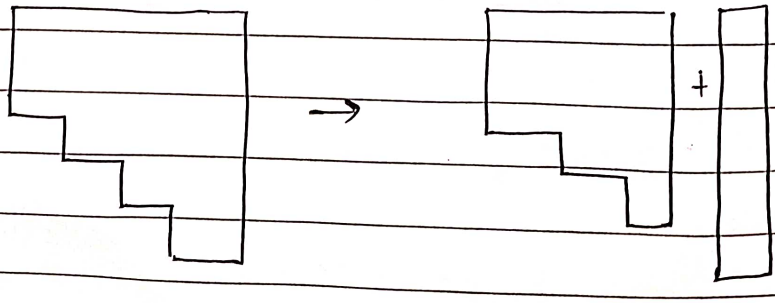
If you go outside the cell then consider the value of the cell to be 0.

soln: let try to conquese this using triangular prefix sum.



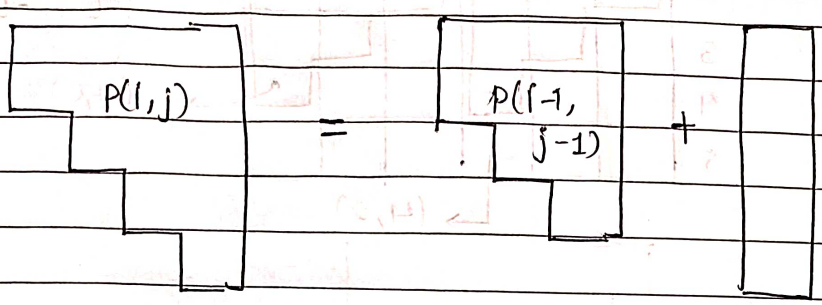
we keep the sum as triangular region upto the first row.

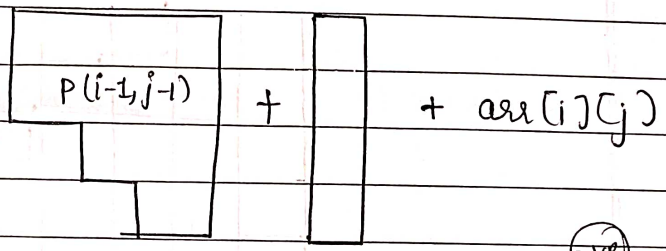
METHOD 1:
using rectangular sums



METHOD 2:

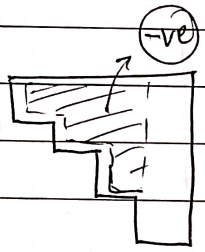
Using only prefix sums previous values





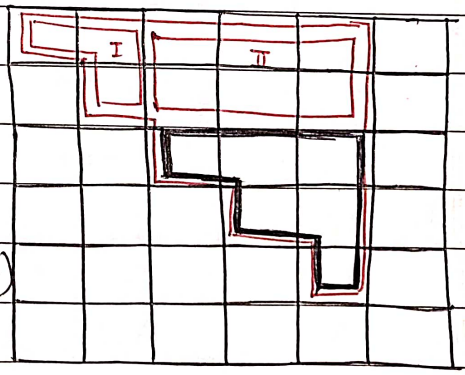
Hence,

$P[i][j] = arr[i][j] + P[i-1][j-1] + (P[i-1][j] - P[i-2][j-1])$



How will this help us?

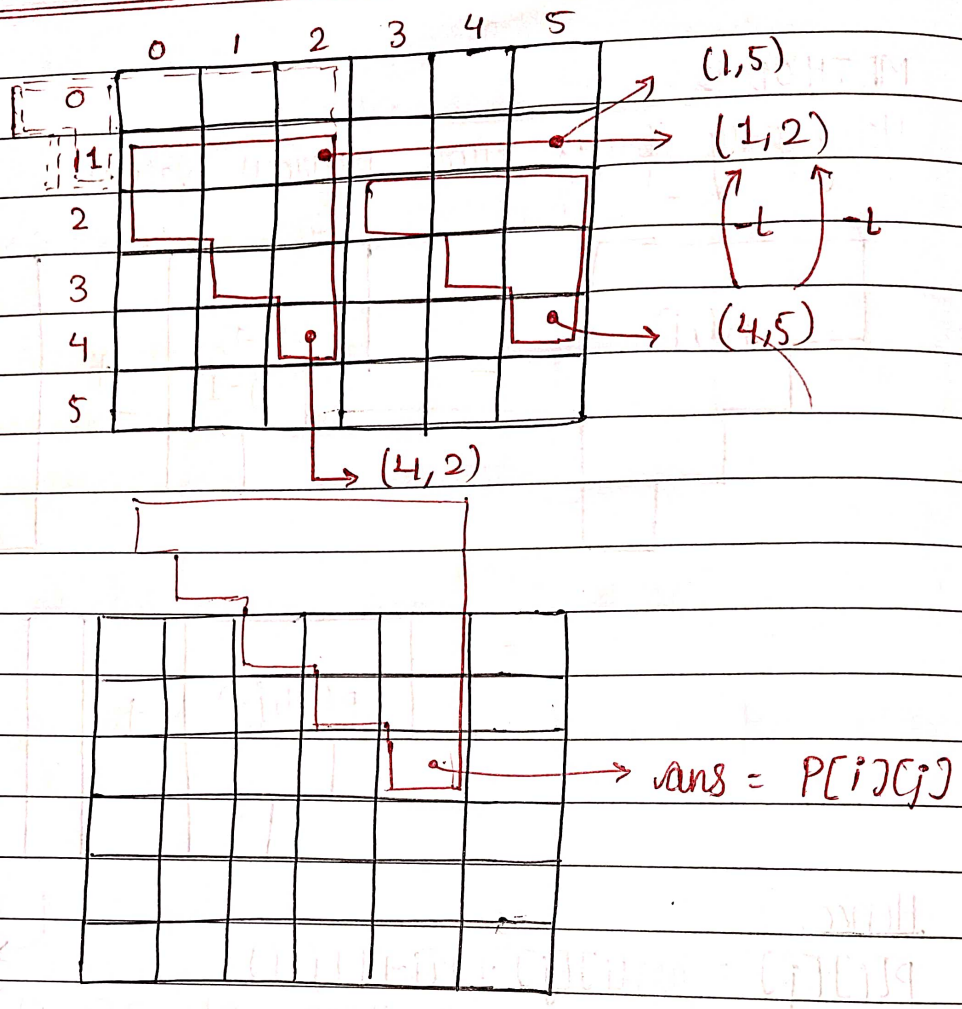
region II → by rectangular prefix sum



Ans → $P[x][y] - P[x-1][y-1] - (R[x-1][y] - R[x-1][y-1])$

Time complexity : $O(N*M+Q)$

Date __/__/____



Bit-manipulation :-

$(10)_{10}$ ← decimal (10 to the base 10)

Binary → 1/0

$(10)_{10} = (1010)_2$

2	10	0
2	5	1
2	2	0
2	1	1
	0	

1 0 1 0

$(14)_{10} = (1110)_2$

2	14	0
2	7	1
2	3	1
2	1	1
	0	

Now,

$(1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 8 + 2 + 1$

$= 11$

1×2^0

1×2^1

0×2^2

1×2^3

(+) = (14)

Operators :-

* & operator → all 1 → 1 ; any 0 → 0

- a = 5 , b = 6

a & b

```
  101
& 110
  100  ⟶  4
```

hence,

a & b = 4

- a = 5 , b = 7 , C = 8

a & b & C = 0

```
  0101
  0111
  1000
  0000
```

Time complexity for bit-wise operators

Even for very large number say, largest value for int is $2^{32}-1$.

Even after doing bit-wise operation, it is T(n) = 32

Hence,

bit-wise operators have time complexity of O(1)

* | operator (OR) → any 1 → 1, all 0 → 0

* ^ operator (XOR) → even 1 → 0, odd 1 → 1

(a = 5) ^ (b = 7) = 2

```
  101
  111
  010
```

a = 5 , b = 5 ⟶ 101
101
000

a^b = 0

P1: If same numbers ~~of~~ are xored then the result will be 0

P1: a^a = 0 ; P2: a^0 = a

* ~ operator (negation) (One's complement)

a = (101)$_2$ = 5
b = ~a
= (010)$_2$
= 2

~3 (11)
= (1 (1 1 1 - - - 0 0))
-ve ↳ flip ↳ even
(-4) { 000 - - 1 1
+1

* >> right shift operator

a = 5 → (101)$_2$

a >> 1 (a right shift of 1)
↳ last bit will go off
↳ = (10)$_2$ = 2

a = 10 (1010)

a >> 3 (right shift of 3)
↓ ↓
(1 ~~0~~ ~~1~~ ~~0~~) → (0001)$_2$ = 1

Actually

$5 >> 1 = 2$

$\hookrightarrow \frac{5}{2}$

$10 >> 3 = 1$

$\hookrightarrow \frac{10}{2}(5) \rightarrow \left(\frac{5}{2}\right) \rightarrow \left(\frac{2}{2}\right)$

↳ dividing 10 by 2 for 3 time

Why?

$10 = (1010)_2$

$\rightarrow 1 \times 2^3$

$\rightarrow 0 \times 2^0$

$\rightarrow 1 \times 2^1$

$\rightarrow 0 \times 2^2$

change by 2^3

$10 >> 3 = (0001)_2$

$\hookrightarrow (1 \times 2^0)$

* << left shift operator

int a = 5

↳ 32 bits

$(5)_{10} = \underbrace{(0000 \ldots 0101)_2}_{\text{total 32 bits}}$

$5 << 2 ; (0000 \ldots 0101)_2$

$\hookrightarrow (10100)_2 = 2^4 + 2^2 = 20$

$5 << 2 = 20$

↳ 5×2^2 ↑

Hence,

$a << p = a \times 2^p$

Q1 arr[] → {2, 1, 2, 5, 6, 5, 7, 7, 6} ← example array

every integer occurs twice & one integer occurs once.

Print that one integer. (1 ← example ans)

Ans → Just take XOR of all numbers.

Q2 Swapping of 2 numbers using XOR.

Ans → step I : $a = a \wedge b$; (b ?)

$b = a \wedge b$

∴

$(b = a \wedge b \wedge b)$

∴

$\boxed{b = a}$

now

$a = a \wedge b$

$= (a \wedge b) \wedge a$

$\boxed{a = b}$

eg: $a = 5, b = 7$ → without taking int temp.

→ SI: $a = a \wedge b \ (5 \wedge 7)$

→ SII: $b = a \wedge b \ (5 \wedge 7 \wedge 7) = 5$

→ SIII: $a = a \wedge b \ (5 \wedge 5 \wedge 7) = 7$ ✓

Q3 Given N, print the XOR of all nos. b/w 1 to N without using loop.

I/P: 5

O/P: 1^2^3^4^5 = 1

Ans → ~~1^2^3~~ Always try to find some pattern

~~001~~

~~010~~

~~011~~

N = 1 Ans → 1

N = 2 Ans → 3

N = 3 Ans → 0

N = 4 Ans → 4

N = 5 Ans → 1

N = 6 Ans → 7

N = 7 Ans → 0

N = 8 Ans → 8

pattern of 4 forming.

Q4 Given a range (L-R), print the XOR (L^(L+1)^(L+2)^---^(R-1)^R) in O(1).

I/P: 2 4

O/P: 2^3^4 = 5

Soln: (L - L) → L

L - (L+1) → L^L+1

1^2 = 1	4^5 = 1
2^3 = 1	5^6 = 3
3^4 = 7	

this is not a pattern

XOR (1-(L-1)) = 1^2^3^ ... ^(L-1)

XOR (1-R) = 1^2^3^ ... ^(L-1)^L^ ... ^R

XOR (L-R) = XOR(1-(L-1)) ^ XOR (1-R)

Q5 Given N, check N if N is odd or not

soln

```
if (N%2 == 0)
    even
else
    odd
```

```
if (N & 1 == 0)
    even
else
    odd
```

more efficient

Time: (a) > (b), a > b

bitwise operators are much faster than %, *, /, +, -

Basic Stuffs

(1) (N, i) → check if the ith bit is set

eg: N = 13, $(1101)_2$

i = 2 (3 2 1 0 ← i)

↳ yes

(i = 1 → not)

(i = 0 → yes)

```
if (N>>i & 1)
{
    ____
}
```

best approach

(N>>i) } now; check if N is odd or not

Aliter;

checking this bit

say for 1 1 0 0 1 } & }
mask 0 1 0 0 0 }
0 0 0 0 0

N & mask == 0 } ⇒ bit is set
else bit is not set.

`mask = 1 << i` → if N is long long, then this might give error as 1<<i will overflow for i > 32.

and

`bool set = mask & N` ← Any

`mask = 1LL << i`

(2) extract the i^{th} bit of N.

N = 13, i = 3 ; $13 = (1101)_2$

3 2 1 0

↳ (1) ← Any

same as previous question.

(3) set the i^{th} bit of N.

say for 1 1 0 0 1 (N)
mask 0 1 0 0 0 (mask = 1 << i)

`N = N | mask` now the ith bit will be set

(4) clear the i^{th} bit of N.

soln : `mask = 1 << i ;`

```
if (mask & N != 0)
{
    // we don't have to do anything
}

if (mask & N == 0)
{
    N = ~~N^mask;~~   N^mask ;
}
```

Alter

say: ..0 0 1 1 0 0 1 0 1

..& 1 1 1 1 1 1 0 1 1 → mask

1 1 0 0 0 0 1

mask = ~(1<<i)

N = N & mask

* (5) Remove the last set bit

eg: 1 1 0 1 1 0

↓ ↑ last set bit

1 1 0 1 0 0

soln: say n = 13 ; 1 1 0 1

(n-1) = 12 ; 1 1 0 0

Ans

N = N & (N-1)

✓ n & (n-1) = 1100

n = 12 ; 1 1 0 0

n_2 = 11 ; 1 0 1 1

} n & (n-1) = 1000

(6) Check if number is power of 2

Sol^{n}: If num is power of 2 ⇒ 0.0001 0000...0 (32 bit)

~~setting~~ removing the last bit will make that number 0

⇒ N → To check N

```
if (N & (N-1) == 0)
{
    it is power of 2;
}
else
{
    not a power of 2;
}
```

(7) Count the number of set bits in a num N?
eg: 14 = 1110 → (Ans = 3)

Sol^{n}: M-I

```
while (n != 0)
{
    if (n & 1 == 1)
    { cnt++; }
    n = n >> 1
}
print (cnt)
```

Time complexity
= O(no. of digits in binary of n)
≈ O(1)

O(no. of bits)
O(position of most signifi. bit

M - II

```
while (n != 0)
{
    n = n & (n-1);   ← setting off (removing) last set bit
    cnt++;
}
```

Time complexity = O(set bits)

M - III

```
cout << __builtin_popcount(n) << "\n";
```

Precedence

1 << 3 - 1

gives 4

Binary operators have less precedence. Hence, use () to make things work in correct manner.

Bitset:

bitset <1000> bs

Faster than arrays (read on your own)

NOTE: 2's complement

~x → one's complement

eg: 5 = 00000101 } 8 bit representation

-x → two's complement

eg: -5 = (~5) + 1

= 11611010 + 1

-5 = 11111011

} How to represent -ve number

int n; cin >> n;

bitset <64> arr(n); } converts 'n' into binary & stores in arr of size 64.

for (int i = 63; i >= 0; i--) cout << arr[i];

↗

// most significant bit

eg: 5 - 3

5 = 00---.0101] 32 bits

-3 = (~3) + 1 = ~(00...11) + 1

= (111...101)

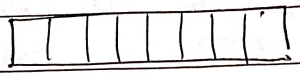
5 + (-3) = 000....0101 1+1 → 0 + carry 1

111....1101 1+0 → 1

Last carry thrown off ← (1 carry) 000---0010

2

Q2 $N \leq 10^5$

arr 

$arr[i] \leq 10^9$

$$\sum_{i<j} arr[i] \wedge arr[j]$$

soln:

```
ans = 0
for (i ∈ [1,N])                    O(N^2)
    for (j ∈ [i+1,N])              ↳ TLE
        ans += arr[i]^arr[j]
```

XOR is not distributive over any operation

eg: N = 3

1 3 7 → 1^3 + 1^7 + 3^7

2 + 6 + 4 = 12

	1	3	7
MSB	0	0	1
	0	1	1
LSB	1	1	1

$3 \wedge 7 = (0 \wedge 1) \times 2^2 + (1 \wedge 1) \times 2^1 + (1 \wedge 1) \times 2^0$

similary

$= (1 \wedge 3) + (3 \wedge 7) + (1 \wedge 7)$

$= (0 \wedge 0) * 2^2 + (0 \wedge 1) * 2^1 + (1 \wedge 1) * 2^0$ → Line 1

$+ (0 \wedge 1) * 2^2 + (1 \wedge 1) * 2^1 + (1 \wedge 1) * 2^0$

$+ (0 \wedge 1) * 2^2 + (0 \wedge 1) * 2^1 + (1 \wedge 1) * 2^0$ ↓ Line 2

Instead of summing along line 1 then line 2, we can change it as summing along line 2 then line 1

for i^{th} bit, we take sum of all pairs then multiply it with 2^i; then sum it over i to get the answer

i	1	3	7		@	b	©
2	0	0	1	→	0	0	1
1	0	1	1	=	(0^0) +	(0^1) +	(0^1)
0	1	1	1				

0	0	1

how to quickly find sum of pairs xored

= (# of 0's) * (# of 1's) → O(n)

= (# of 0's) * (# of 1's) * 2^i

we will do this for all the bits then sum of these over i will give the ans

Time complexity reduces to O(32 x n)

Crux

$$\sum f(x,y) = \underbrace{\sum \underbrace{(\sum)}_{O(-)} * 2^i}_{32 \times O(-)}$$

solve the problem in each bit, then sum up the solution.

Q3 Suppose in previous problem we have

$$\sum_{i<j} arr[i] \,\&\, arr[j]$$

Sol^n: say N = 3

i	1	3	7
MSB 2	0	0	1
1	0	1	1
LSB 0	1	1	1

$(1 \& 3) + (3 \& 7) + (1 \& 7)$ $(= 5)$

$= (0\&0) * 2^2 + (0\&1) * 2^2 + (0\&1) * 2^2$
$(0\&1) * 2^1 + (1\&1) * 2^1 + (0\&1) * 2^1$
$(1\&1) * 2^0 + (1\&1) * 2^0 + (1\&1) * 2^0$

$1\&3$ $3\&7$ $1\&7$

$$= \sum \left(\sum (pairs\&) \times 2^i \right)$$

↳ no. of ones C_2

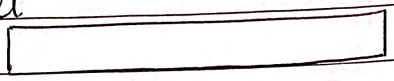
Similarly for OR (|) → $(\#0s * \#1s) + (\#1s C_2)$

Or

$^nC_2 - {}^{\#0s}C_2$

- Bit expressions are sum independent on each bit.

Q4 N, target

arr  of N elements

Find no. of (i, j) such that
$((arr[i] \,\&\, arr[j]) + (arr[i] \mid arr[j])) = target$

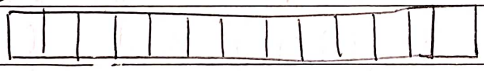
Soln: $(arr[i] \,\&\, arr[j]) + (arr[i] \mid arr[j]) = arr[i] + arr[j]$

NOTE:

$A + B = [(A \,\&\, B) + (A \mid B)]$

$= 2 * (A \,\&\, B) + (A \,\hat{}\, B)$

* Q5 $N \leq 10^5$

arr: 

Find

$\max (arr[i] \,\hat{}\, arr[j])$ & $\min (arr[i] \,\hat{}\, arr[j])$

Soln: It is better to solve in greedy fashion → Trie

try 2 make most significant digit 1 as $(1000)_2 > (0111)_2$

For minimum $(arr[i] \,\hat{}\, arr[j])$, sort (arr, arr+n) & then find minimum among a_i and a_{i+1}.

Q6 Given

$$x_1 + x_2 + \cdots + x_n \leq C$$

find

$$\max \left(x_1^2 + x_2^2 + \cdots + x_n^2\right) \; ; \; \boxed{0 \leq x_i \leq D}$$

~~forget this~~

Sol^{n}:

$x_1 + x_2 = 6 \longrightarrow (0,6)\,;\ (1,5)\,;\ \ldots,(3,3)$

$x_1^2 + x_2^2 = ?$

↳ maximum will occurs at (6,0)

use greedy approach

fill $x_1, x_2, x_3, \ldots$ up to their highest values (D) until we have to take ($<D$)

for $\min (x_1^2 + x_2^2 + \cdots + x_n^2)$

divide equally. $\left\lfloor \frac{C}{N} \right\rfloor$

Q7 Given a positive integer n, find the count of positive int $0 \le i \le n$, such that $n + i = n \wedge i$

Sol^{n}: $0 \le i \le n$ constraints

&

$n + i = n \wedge i$

$\Rightarrow n | i + n \& i = n \wedge i$

$\Rightarrow 2(n \& i) + n \wedge i = n \wedge i$

Hence, we have

$$\boxed{n \& i = 0}$$

$n \rightarrow 1\,0\,1\,1\,0\,0\,0\,1$

$i \rightarrow 0\,\downarrow\,0\,0 \ldots\ldots 0$

1/0 1/0

$\therefore$ count $= 2^{\text{no. of non-set bits}}$ Ans

Learn properties

$a + b = (a \wedge b) + (a \& b)\,2$

$a + b = (a | b) + (a \& b)$

$\therefore$

$a | b = (a \wedge b) + (a \& b)$

$a | b \equiv (A \cup B)$

$A \& b = (a \cap b)$

$(a | b) + (a \& b) \equiv (a \cap b) + (a \& \cup b) = a + b$

Q8 Given an array of N positive integers. You can perform this operation any number of times. choose 2 indices x and y. If arr[x] = a & arr[y] = b, then after operation

1. arr[x] = a OR b
2. arr[y] = a & b

Perform operations optimally such that $\sum_{i=0}^{n-1} arr[i]^2$ is maximized. Print that maximum value.

Sol^n.

$$(a, b) \rightarrow (a|b \ , \ a \& b)$$

$$a + b = (a|b) + (a \& b)$$

so, sum of array will always remain constant.

we, can shift bits to other number, so that, sum will remain constant.

Now,

maximize, $\sum a_i^2$ ← make one number as $\sum a_i$ & all other as 0.

now, $\sum a_i$ = constant

this we already discussed

But our constraint may or may not allow us to shift all the Σa_i into one number, so our strategy is to make one number as large as possible.

↓

then continue doing it for other number

Soln: a b c d

a|b (a&b) c d

a|b|c (a&b) (c&a) d

a|b|c|d (a&b) (a&c) (a&d)

p q r

p|a|r q&p r&p

s t

s|t s&t

a&b&c&d

Brute force

Optimisation

2 →	0	0	1	1
1 →	0	1	0	1
0 →	1	1	1	0
	1	3	5	6

← total no. of one's in each row remains constant.

we want to make numbers are large as possible, we can shift bit from one row to another

final matrix

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

} this is the best we can do

Q9 Given an array A of N positive integers, find the maximum of bitwise ANDs of all subsequences with length equal to X.

Soln: largest bitwise of x number taken from the array.

$a_1 \& a_2 \& \cdots \& a_x$ = ans

make the largest bit as 1

```
int n, x;   cin >> n >> x
vector<int> arr(n);
for (int i=0; i<n; i++)  cin >> arr[i];

int ans = 0;
for (int i=29; i>=0; i--) {
  vector<int> elementBitset;
  for (auto v: arr)
  {
      if (v & (1LL << i))
          elementBitset.push_back(v);
  }
  if (elementBitset.size() >= x) {
      arr = elementBitset;
      ans += (1LL << i);
  }
}
```

Q10 XOR AND OR Query

Given, N, arr[]
Q → no. of queries

each query
(L, R), find

i) min X_1 such that $\sum_{i=L}^{R} A_i \text{ XOR } X_1$ = maximum

ii) min X_2 such that $\sum_{i=L}^{R} A_i \text{ OR } X_2$ = maximum

iii) min X_3 such that $\sum_{i=L}^{R} A_i \text{ AND } X_3$ = maximum

$0 =< X_1, X_2, X_3 < 2^{31}$

Soln: (i) $[A_L \quad A_{L+1} \quad \dots \quad , A_R]$
(X_1)

solving for each bit

[1 2 3 6]

Bits				
2	0	0	0	1
1	0	1	1	1
0	1	0	1	0
	1	2	3	6

(X_1) $a \to 0/1$, $b \to 0/1$, $c \to 0/1$

0^a	0^a	0^a	1^a
0^b	1^b	1^b	1^b
1^c	0^c	1^c	0^c

~~tomo~~ to maximize each bit,

$a = 1$, $b = 0$, $c = 0/1$

$a = 1$: if $\#0^s > \#1^s$

$b = 0$: $\#1^s > \#0^s$

$c = 0/1$: take 0 to minimize x_1