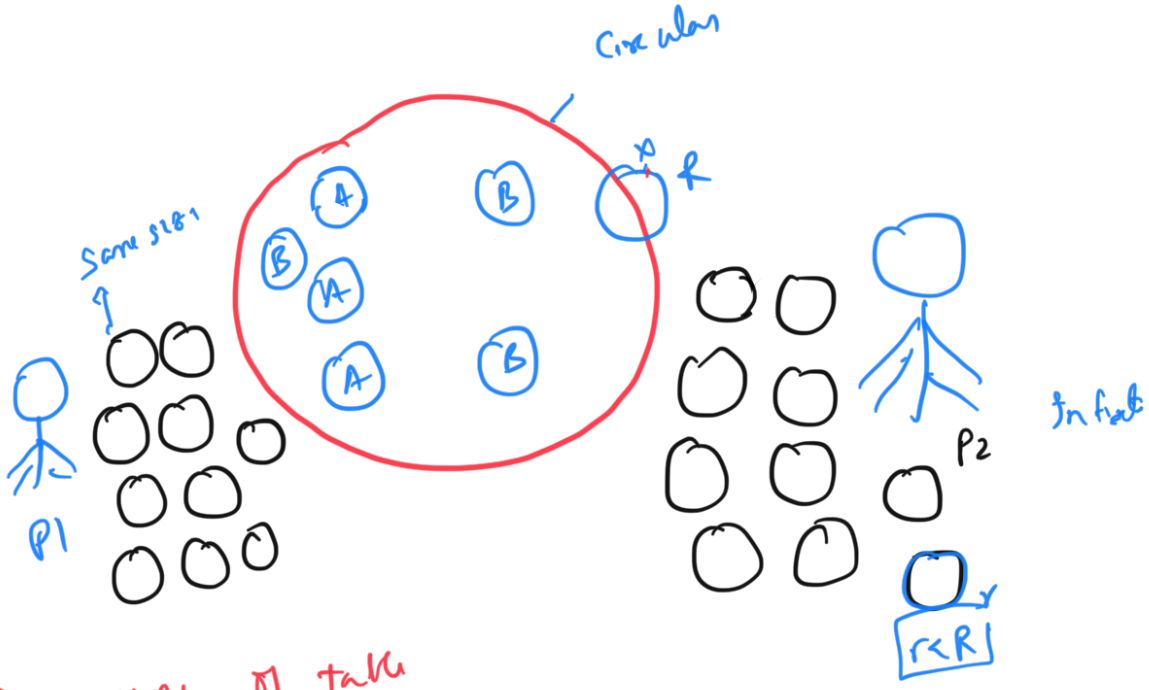
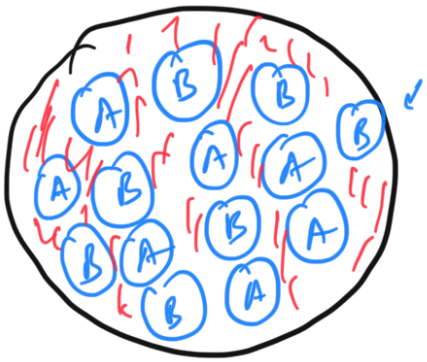
Problem - Solving -1

Question :

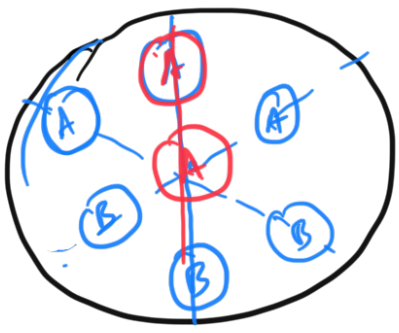


Dimensions of table



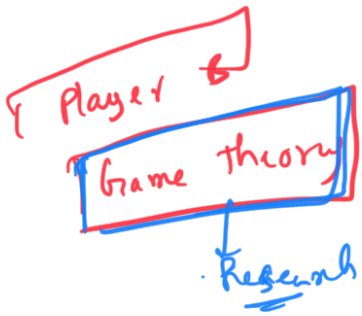
Remaining space can contribute to more disks

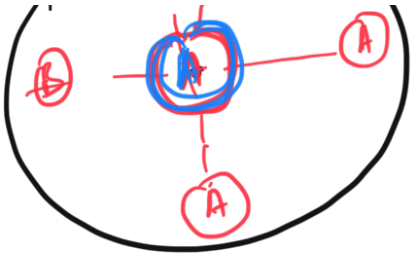
Copy- Cat Technique : Make same moves as your opponent.



PA

PB

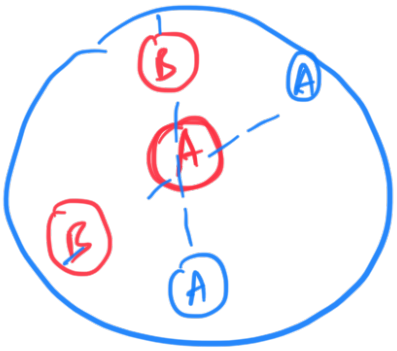




A will copy B.

B will exhaust all moves

A will always win



GCD -

Greatest Common Divisor

Highest Common Factor

$(4, 9)$

$\gcd(4, 9) = 1$

4 = (1, 2, 4)

9 = (1, 3, 9)

4 and 9 co-prime to each

$\gcd(6, 9) = 3$

6 = 1, 2, 3, 6

9 = 1, 3, 9

(x, y)

Two numbers are co-prime if

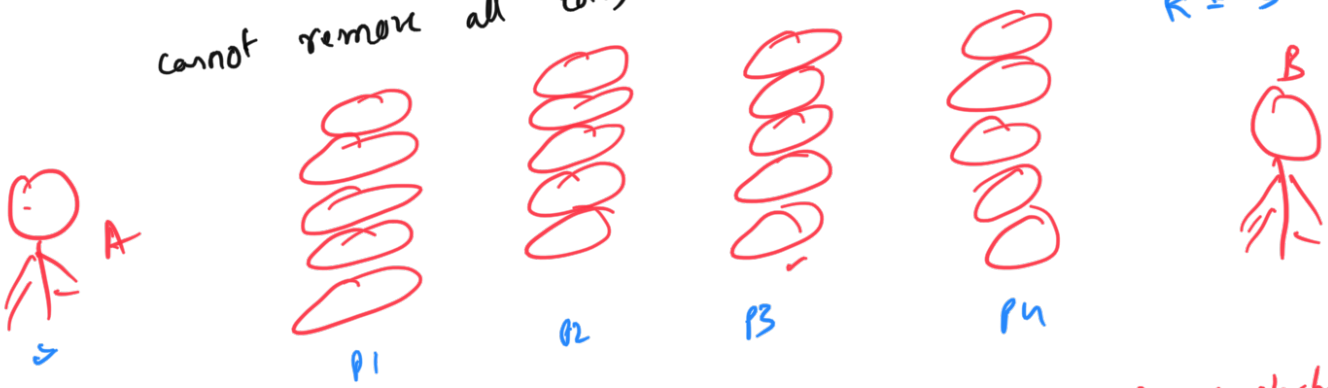
$\gcd(x, y) = 1$

Question: Prime Game

N piles M coins ⇒ each pile has K coins

N = 4
K = 5

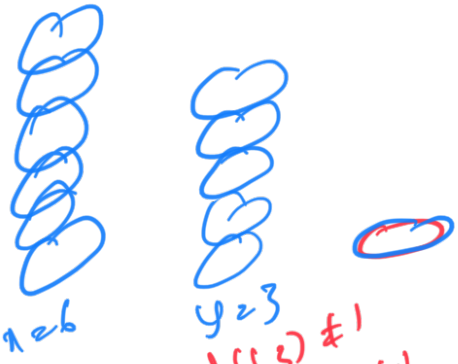
cannot remove all coins



initially = x ✓
finally = y ✓

A is starts

$$\gcd(x, y) = 1$$

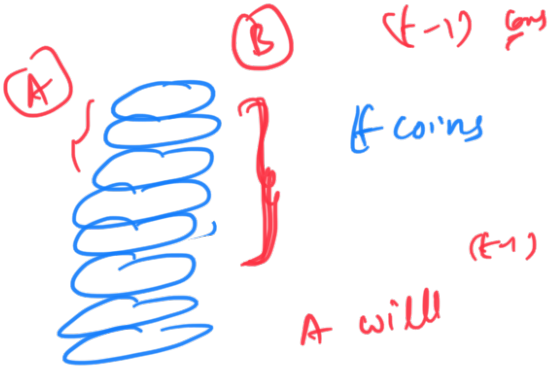


gcd(1,3) ≠ 1
gcd(6,2) ≠ 1
gcd(6,5) = 1

x = 6
y = 5

x = 6
y = 1
gcd(6,1) = 1

1 Pile
size = 1
Bwin



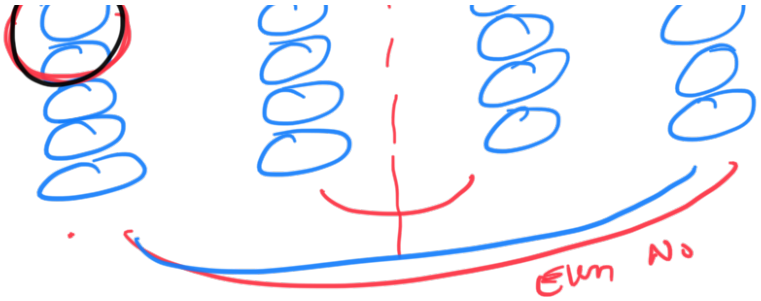
A will

gcd(n, 1) = n?
= 1

gcd(k, 1) = 1

Even No. of Piles Mirror

N = 4
K = 5



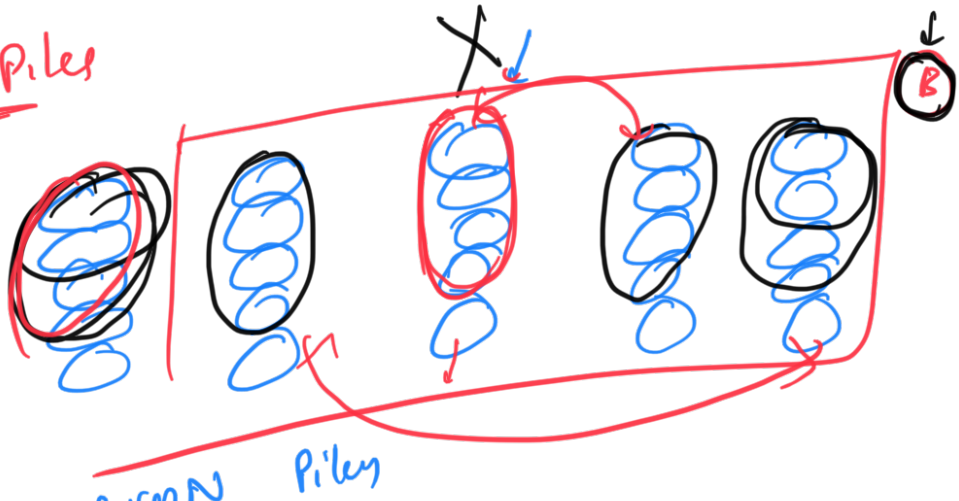
n = 5
y = 3

```
if(K==1)
    B
if N is even
    B
else
    A
```

Eventually, Player A will exhaust of moves.

'B' will always win for K=1

Odd Piles



T.C: O(1)

A will win

All piles have 2 coin ⇒ B will win

N = 6
K = 1

B will win

Question: Missing Integer.

Array of (n-1) unique integers where $1 \le a[i] \le n$. Find the missing Integer.

$N = 10^9$

[1, 8]

T.C: O(n)

N = 8

1 4 2 8 6 7 5

2)

Hashset = {1, 4, 2, 8, 6, 7, 3}

O(1)

Brute Force:

Sort the array →

T.C: O(nlogn)

S.C: O(1)

Approach 2: Hash set

[1-8]

set = {1, 4, 2, 8, 6, 7, 3}

```
for( i = 1; i <= n; i++) {
    if( i is not in set)
        return i.
```

T.C: O(n)

S.C: O(n) → (Hashset)

Approach 3:

sum(1, 2 .... N) = N(n+1)/2

sum(array)

n(n+1)/2 − Sum

1+2+3 .... N = n(n+1)/2 → O(1)

sum(array) → O(n)

sum1

sum2

T.C: O(n)

S.C: O(1)

long long = 10^{18}

N = 2×10^9

$10^9 \times 10^9 > 10^{18}$

Overflow

$N^2 = 4 \times 10^{18}$

Approach 4:

1 · 3 4 5

A ^ A = 0 ; ans = 0

0 ^ 1 ^ 4 ^ 2 ^ 8 ^ 6 ^ 7 ^ 5

ans = 1 ^ 4 ^ 2 ^ 6 ^ 8 ^ 7 ^ 5

A = { 1 ^ 4 ^ 2 ^ 8 ^ 6 ^ 7 ^ 5 }

N = { 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7 ^ 8 }

A ^ N = 5

0 ^ 0 = 0

(1 ^ 1)

0 ^ 0 ^ 0 ^ 0 ^ 0 ^ 5 = 5

(XOR) → variable

T.C : O(n)

S.C : O(1)

N = 2 x 10^9

N(N+1)/2 > 4 x 10^18

1 ^ 2 ^ 3 ^ 4 = 1 ^ 4 ^ 3 ^ 2

ans = 0;
ans = 1

```
ans = 0;
for(int i = 0; i < A.size(); i++) {
    ans = ans ^ a[i];
}
for(i = 1; i <= n; i++) {
    ans = ans ^ i;
}
return ans1 ^ ans2
return ans;
```

[1, 2, 3, 4 ... n]

element of array

[1 - N]

[1, 2, 3, 4, 5, 6, 7, 8]

$O^{1}x = x$

$1^{1}x = x \,?\, x$

Question:

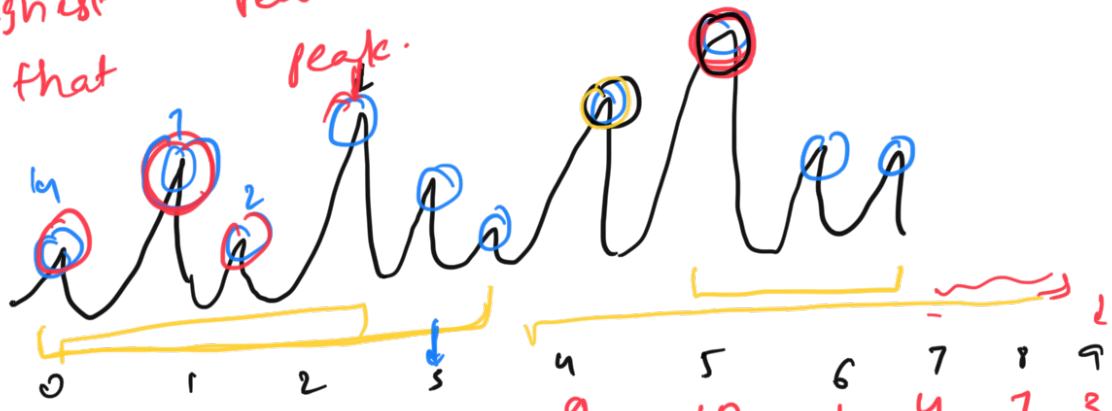
Range Sum Query

(L, R)

	0	1	2	3	4
A =	1	3	4	5	3
pre =	1	4	8	13	16

① [L, R] : pre[R] − pre[L−1]

Question: Array of heights of n peaks of mountains. For every peak, find the highest peak on both the sides of that peak.



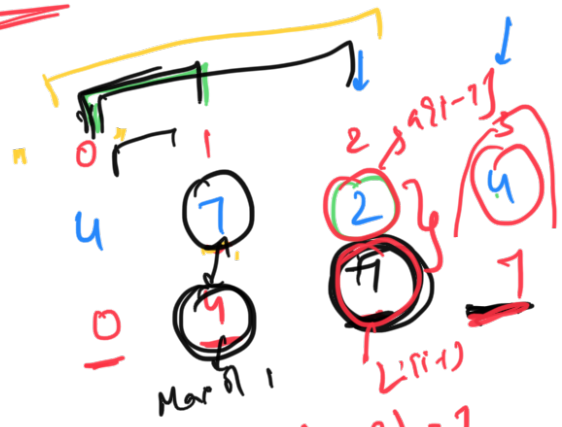
A =	4	7	2	4	9	10	1	4	7	3
L =	0	4	7	7	7	9	10	10	10	10
R =	10	10	10	10	10	7	7	7	3	0

Brute Force

O(n) for 1 Peaks

T.C : $O(n^2)$

Efficient Approach :



0 ... i-1

(i)

4	5	6	7	8	9
9	10	1	4	7	3

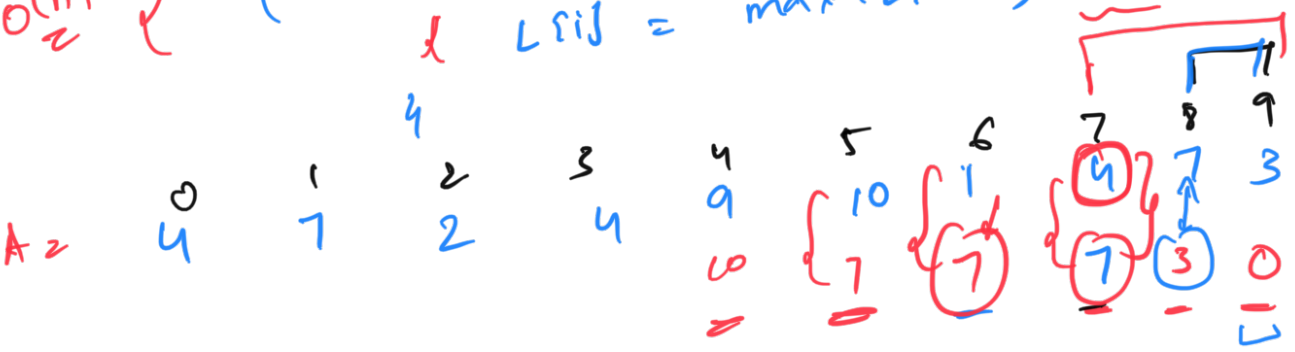
L : Prefix Max

Left[i] = max(Left[i-1], a[i-1])

Left(i) : PrefixMax (i) = The maximum number in the first 'i' numbers of the array

```
L[0] = 0;
for (i=1; i<n; i++) {
    L[i] = max(L[i-1], a[i-1])
}
```

$O(n)$



R :

Right[i] = max(Right[i+1], a[i+1])

max(7,4)

Prefix Max

```
R[n-1] = 0;
for (i = n-2; i >= 0; i--)
    max( R[i+1], a[i+1] );
```

O(n) → R[i] = max ...

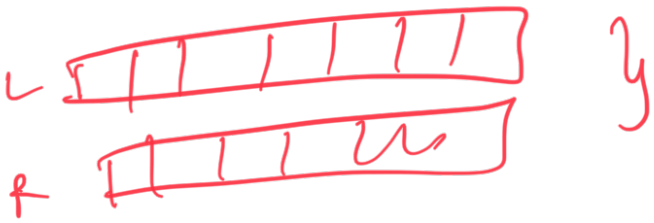
Stacks

T.C: O(n)
S.C: O(n)

L[i] = [0 .... i-1]

L[i] = [0 .... i] ← prefix max

L[i] = max [0 .... i-1]



Question: Array of size N.

Pick (B) elements either from left or right end of the array.

Find the max possible sum

N = 5

B = 3

0	1	2	3	4
5	-2	3	1	2

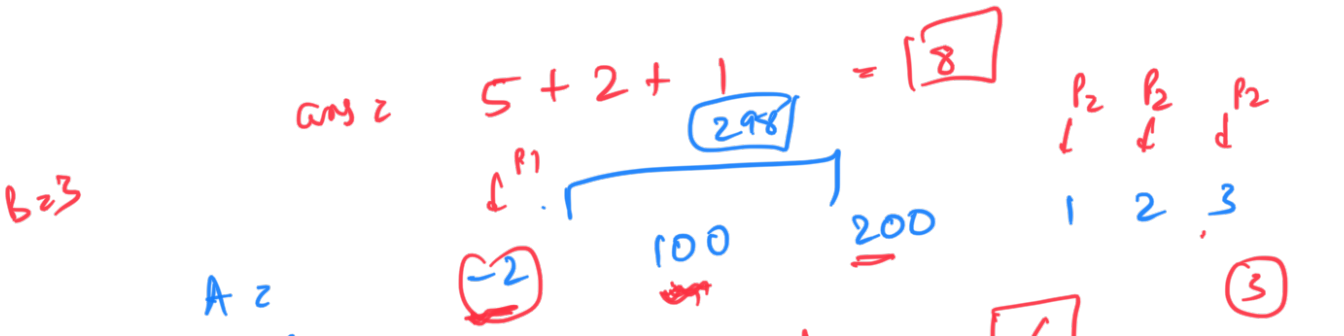
Step 1: 5 + 2 + 1 = 8

Suggested Approach:

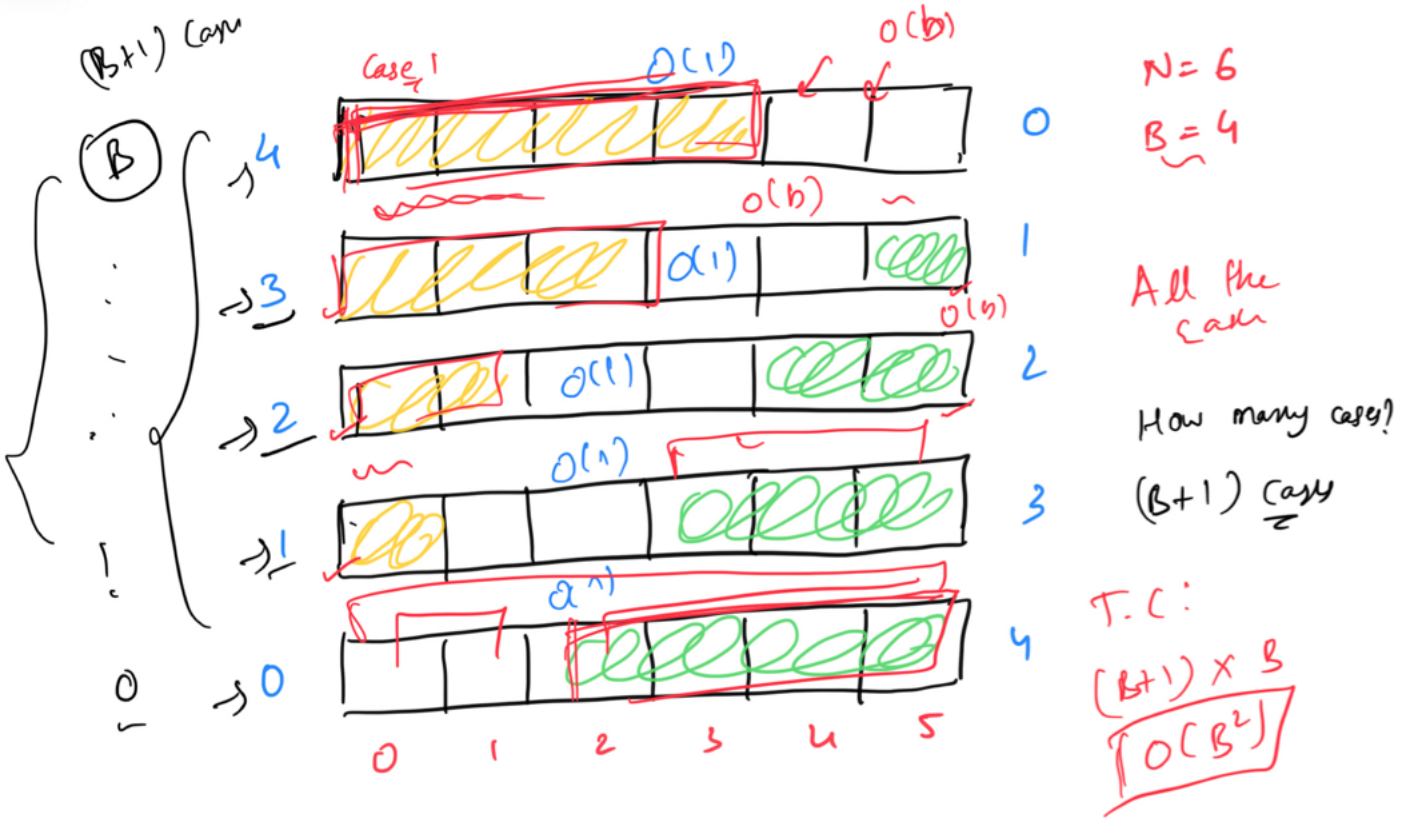
-2 1 2 3 5

ans = 5+3+2 = 10

Suggested Approach2: 2 pointers — (Greedy)



Solution :


(B+1) Cases
B
4
3
2
1
0
Case 1
O(1)
O(b)
O(1)
O(b)
O(1)
O(b)
O(1)
O(n)
0
1
2
3
4
5
N = 6
B = 4
All the cases
How many cases?
(B+1) Cases
T.C:
(B+1) x B
O(B²)

Brute Force : $O(B^2)$

Efficient :

Use Prefix Sum

T.C : $O(B*1) + O(n)$

S.C : $O(n)$

T.C : $O(n+B)$

1) Generate Prefix sum Array : $O(n)$

2) Considering all (B+1) cases $*$ $O(1)$

T.C : $O(n) + O(B)$

$\Rightarrow O(n+B) \Rightarrow O(n)$

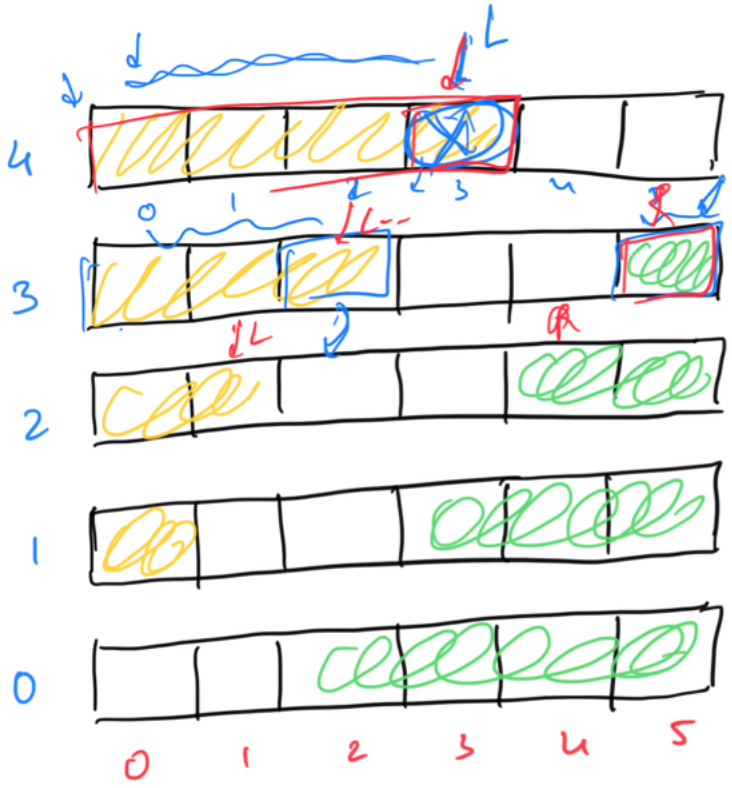
T.C : $O(n+B)$

S.C : $O(n)$

(pseudocode)

Approach 2 :

O(B)



N = 6

B = 4

0 leftSum = —
L = B-1

1 rightSum = 0
R = N-1

2 ans = 0

3 {leftSum -= A[L]
rightSum += A[R];
update ans:
L--;
R--;

T.C: O(B) + O(B)

T.C: O(B)
S.C: O(1)

Jh Doubts session

Optimisation Problem
- Greedy
- Explore all possibilities (Recursion/DP) Paradigm

Question: Beggars Outside Temple.

N beggars
W worshippers
[x money [L, R]

Find money with all the beggars

N = 6

		0	1	2	3	4	5
		0	0	0	0	0	0
W	5 [2,4]	0	0	5	5	5	0
	10 [1,3]	0	10	15	15	5	0
	3 [4,5]	0	10	15	15	8	3
	2 [0,2]	2	12	17	15	8	3

[L, R, x]

Brute Force:
- Take array of size 'N' with all of

W worshippers [L, R, x]

```
for (int i=L; i<=R; i++)        O(N)
    a[i] += x;
```

T.C : $O(W \times N)$

$= O(W \cdot N)$

S.C : $O(n)$

$N = 10^5$

$W = 10^5$

TLE

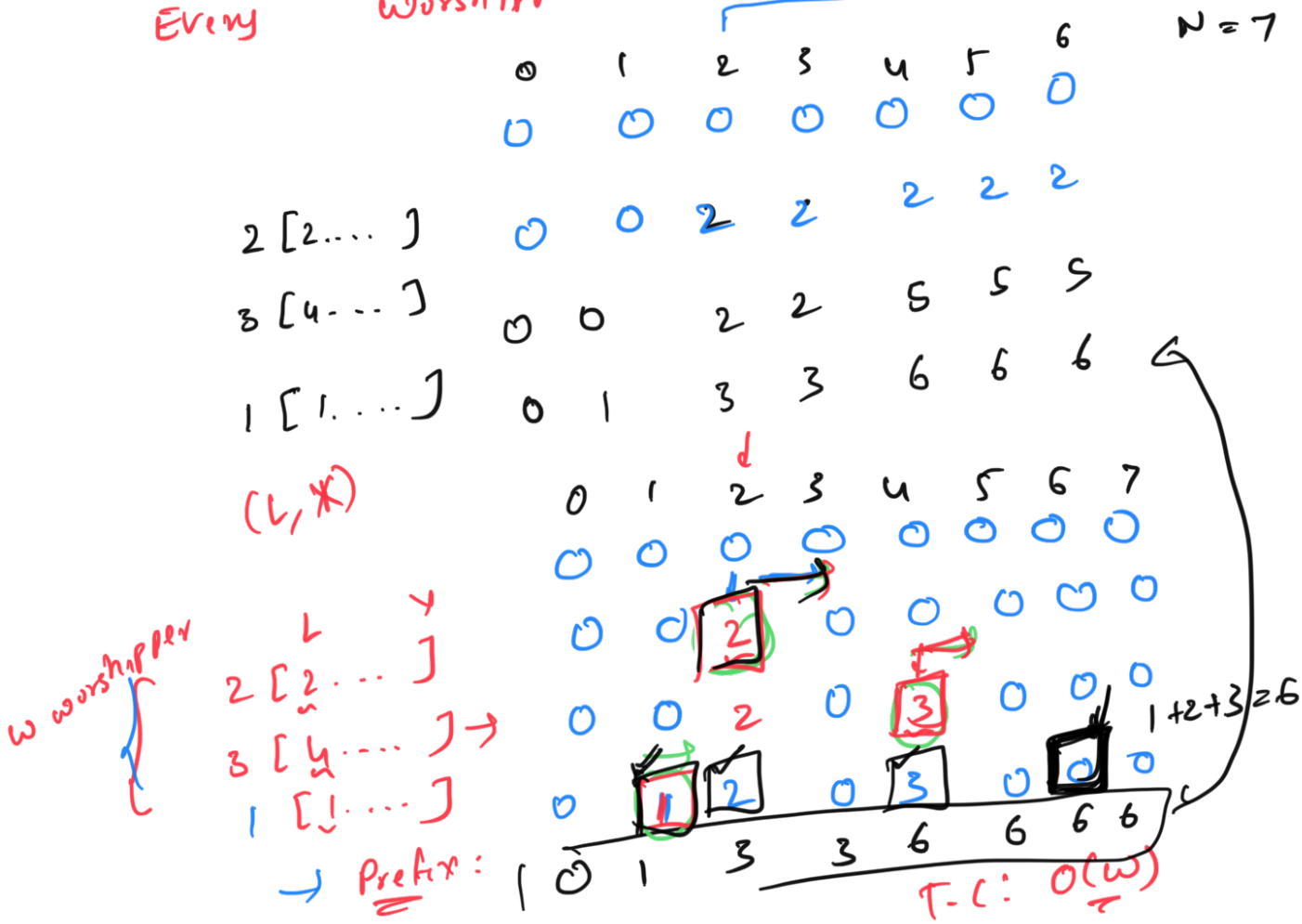
Efficient Approach:

Reduce the Problem

[L, R]

[L, N-1]

Every worshipper gives from [L, N-1]



A[L] += x

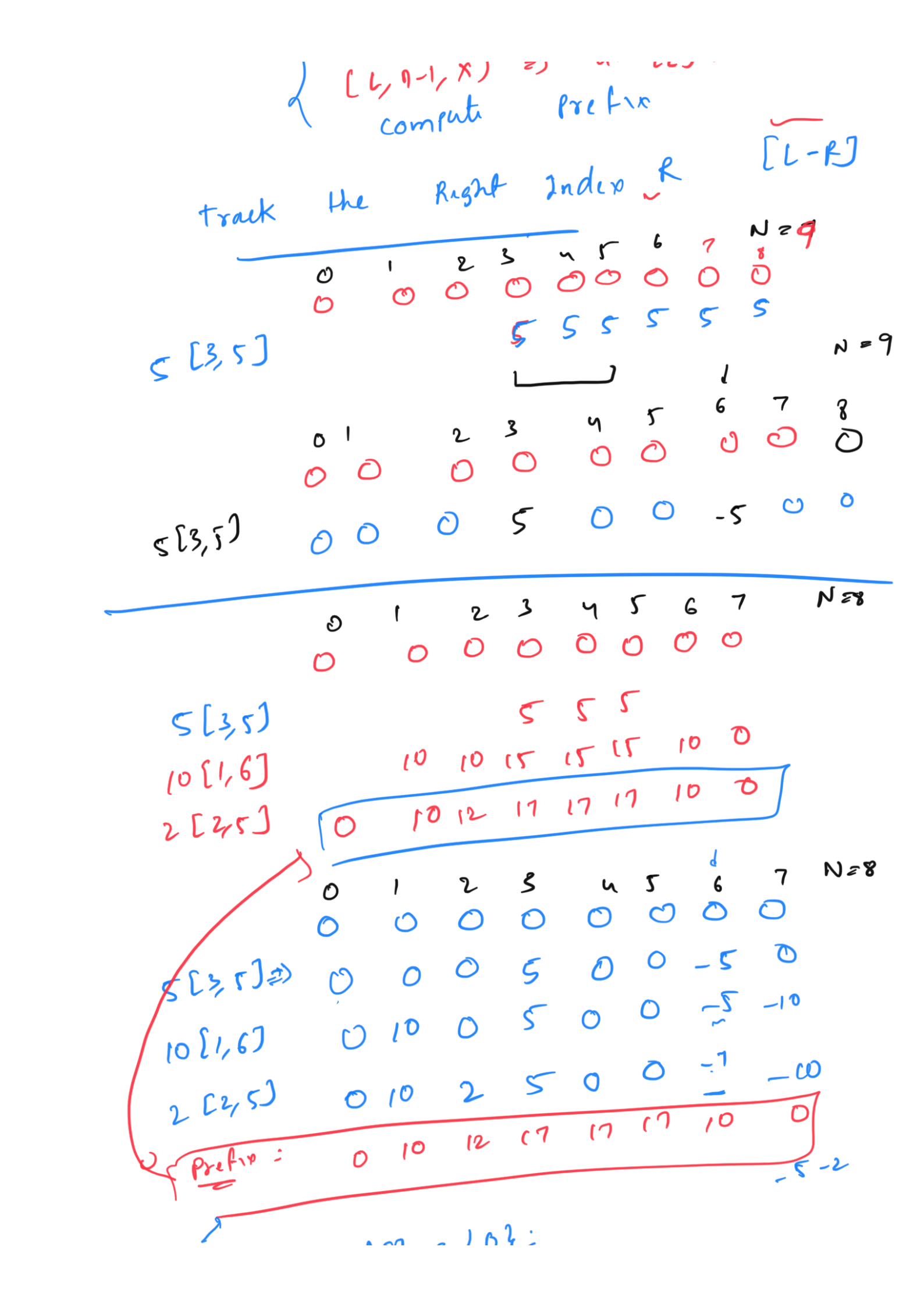
Prefix Sum

T.C: $O(W) + O(n)$

T.C: $O(W+n)$

A[] = {0};

arr[L] += x;


[L, n-1, x)
compute Prefix
[L-R]
Track the Right Index R
N = 9
5 [3, 5]
N = 8
5 [3, 5]
10 [1, 6]
2 [2, 5]
Prefix :

A[] = {0}

(L, R, x)

A[L] += x;
A[R+1] -= x; }

Prefix Sum of A.

T.C: Q x O(1) + O(N)

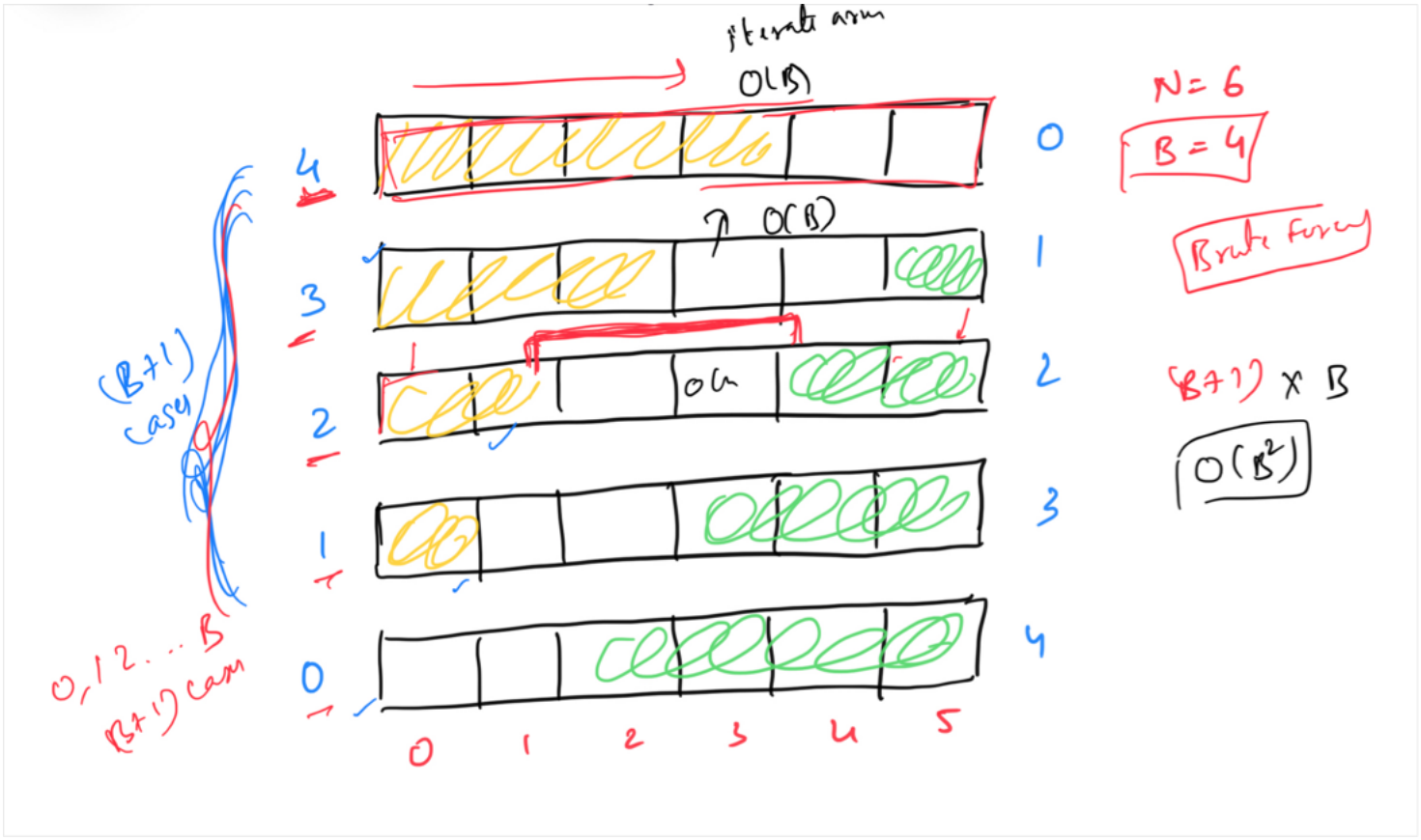
T.C: O(Q+N)

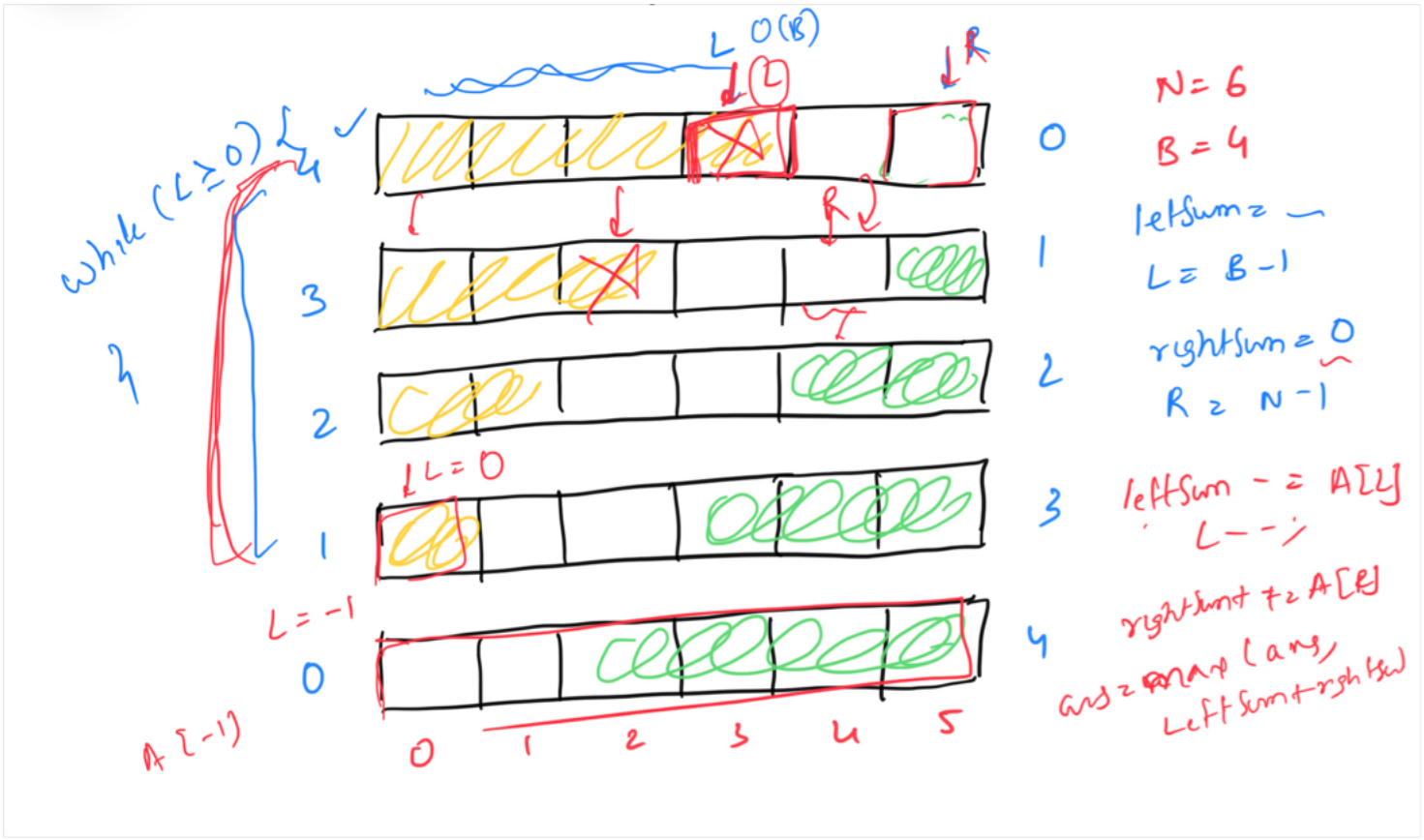
S.C: O(N)

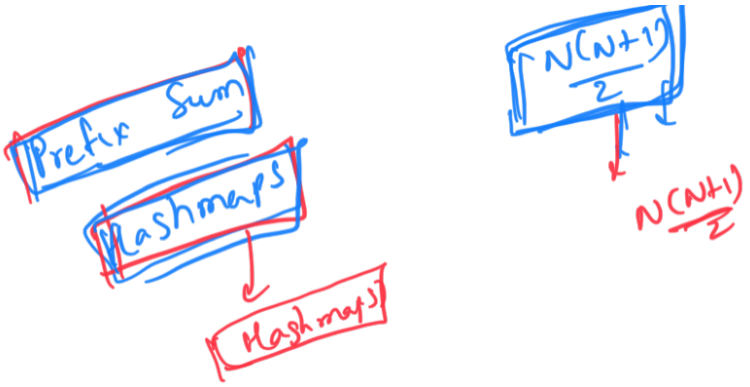
C

x, y

gcd(x, y) = 1







Find Missing Integer

```
ans = 0;
// Array of N-1 integers
for(int i = 0; i < N-1; i++)
      ans = ans ^ a[i];
// Iterate the first N natural numbers
for(int i = 1; i <= N; i++)
      ans = ans ^ i;
return ans;
```

Find peaks to the left and right

```
L[0] = 0;
for(int i = 1; i < n; i++){
      L[i] = max(L[i-1], arr[i-1];
}
R[n-1] = 0;
for(i = n-2; i>=0; i--){
      R[i]  = max(R[i+1], arr[i+1];
}
```

Find max sum using B elements: O(n) space

```
pre[0] = A[0];
for(int i = 1; i < n; i++)
      pre[i] = A[i] + pre[i-1];

int L = b-1, R = n-1;
ans = INT_MIN;
while(L >= 0){
      ans = max(ans, pre[L][left part] + (pre[n-1] - pre[R])[right
part];
      L--;
      R--;
}
// The last case where we take 0 elements from left and B elements
//from right
ans = max(ans, pre[n-1] - pre[n - B - 1]);
return ans;
```

Find max sum using B elements: O(1) space

```
// Sum of first B elements
LeftSum = sum[0… B-1]
L = B-1;
RightSum = 0;
R = n-1;
ans = INT_MIN;

while(L >= 0){
      ans = max(ans, LeftSum + RightSum);
      LeftSum -= A[L];
      L--;
      RightSum+= A[R];
      R--;
}
// Now L = -1. last case where we take 0 elements from left and B
//elements from right
ans = max(ans, LeftSum + RightSum);
return ans;
```

Find money with beggars

```
A[N+1] = {0};
for(int i = 0; i < W; i++){
      L,R,X = i'th worshipper's data
      A[L] += X;
      A[R+1] -= X;
}
// Prefix Sum
for(int i = 1; i < N; i++)
      A[i] += A[i-1];
```