PHASE-2

```
for(int i=0; i<n; i++) {
    for(int j=0; j<n; j++) {
        ans = ans ^ (ar[i] + ar[j]);
    }
}
```

(I) $O(N^2)$

[a b c]

ans ^ = (a+a)

ans ^ = (a+b) x

ans ^ = (a+c) x

ans ^ = (b+a) x

ans ^ = (b+b)

ans ^ = (b+c) x

ans ^ = (c+a) x

ans ^ = (c+b) x

ans ^ = (c+c)

It can be optimized as

```
for(int i : ar)
{
    ans ^= (2*i);
}
```

$O(N)$

(II)

```
for(int i=0; i<n; i++){
    for(int j=0; j<n; j++){
        ans = ans + (ar[i] ^ ar[j]);
    }
}
```

$O(N^2)$

$a \wedge a = b \wedge b = c \wedge c = 0$

ans = $a \wedge b + a \wedge c + b \wedge a + \cdots$

ans = $2(a \wedge b) + 2(b \wedge c) + 2(c \wedge a)$

no. of pairs = $\frac{n(n-1)}{2}$

let XOR pairs be x_i

$x_1 + x_2 + x_3 + \cdots + x_k$

we can express each XOR pair as sum of 2 powers

let $x_1 + x_2 + x_3$

be $10 + 24 + 12$

$(8+2) + (16+8) + (8+4) \Rightarrow 3\times8 + 1\times4 + 1\times16 + 1\times2 \Rightarrow \sum c_i \times 2^i$

Hence ans = $2(\sum c_i \times 2^i)$

but XOR of 2 nos will be set only when i^{th} bit of one number is set and other is unset

In array let us say there are k numbers with i^{th} bit as set, Hence there will be n-k numbers having i^{th} bit unset

so $c_i = k(n-k)$;

```
for(int i=0; i<32; i++)
{  int k = 0;
   for (int j=0; j<n; j++)
   {
      if ((ar[j] >> i) & 1 == 1) k++;
   }
   int c = (n-k) * k;
   ~~ans += c x pow(2,i);~~   ans += c x (1<<i);
}
return (ans x 2 % M);
```

$O(32 \times N), 1$

III

ar_N : -12 14 -1 6 18 26

K : 34

check if there is a subset with sum K

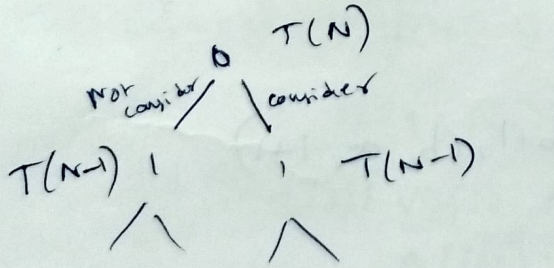
```
boolean ~~S~~subset (int[] a, int n, int K, int sum, ind idx)
{
   ~~if (sum == K) return true;~~
   ~~if2(sum~~  if(idx == N) return (sum == K);
   ~~return subset(int[] a, int n, int K, in~~
   return (subset(a, n, K, sum+a[idx], idx+1) || subset(a, n, K, sum, idx+1))
}
```

solutions

① $N \times 2^N$, 1

↳ Iterate all subsets and check sum

② $O(2^N)$



$T(N) = 2T(N-1) + 1$

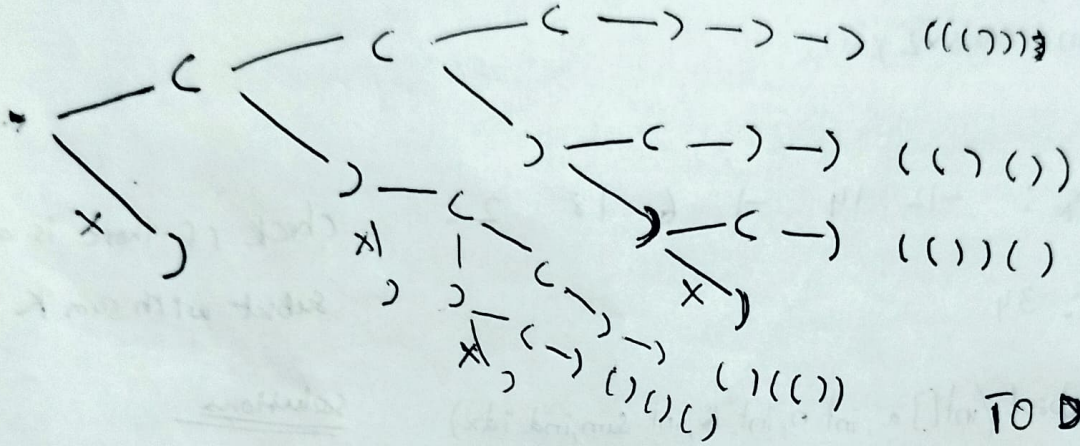
$\Rightarrow O(2^N)$

Parenthesis

N = 4
(( ))
( ) ( )

N = 6
((( ))) ①
( ) (( )) ⑤
( ) (( )) ④
(( ) ( )) ③
(( ) ( )) ②

Print all valid parenthesis in lexicographical order



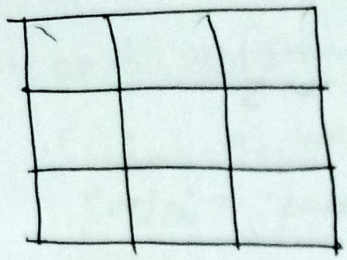
TO DO

Find valid no of parenthesis for N

Catalan number

$$\frac{2n!}{(n+1)!}$$

```
void parenthesis (N, ob, cb, char[] ar, int i)
{
    if (i==N){  for(char i: a)
                    { print(a+" "); }
    }
    if (ob < N/2)
        ar[i] = '('
        parenthesis (N, ob+1, cb, ar, i+1)
    if (cb < ob)
        ar[i] = ')' ;
        parenthesis (N, ob, cb+1, a, i+1)
}
```



Magic Square

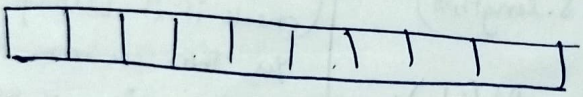
1 - 9
No repetitions
sum of rows, cols, diag equal

Q: given a matrix find min cost to convert it to a magic square.

As, brute force

- we will generate all 3x3 matrices having 1-9
- check for magic squares possible
- compare with given matrix and update cost

3x3 matrix is converted as array of size 9

a: 

to generate all possible 3x3 matrices we find permutations of 1-9

```
Void permutations (int[] ar, bool[] visited, int idx)   O(9!)
{  if(idx==9){ if (magicSq(ar)) { int min = math.min(min, cost(ar, og))}}
   for(int i=1; i<=9; i++){
      if (!vis[i]){ vis[i] = True;
                    a[idx] = i;
                    permutations(ar, vis, idx+1);
                    vis[i] = false;
                  }
   }
```

LAB: XOR of sum of pairs
Sum of XOR of pairs
Balanced parentheses
smart square.

6/11/24

Str: s m a r t i n t e r v i e w s

L: ["s", "sm", "sma", "art", "interviews", "inter", "views", "smart", "view"]

(I) Check possibility of partition

(II) No. of ways to partition

(III) Min no. of partitions.

(I)

```
/* boolean partition (String s, list<> L, int idx)
   {  string c = "";
      while (idx < s.length())
      {
          c += s.charAt(idx);
          if (l.contains(c)) if (partition (S, L, idx+1)) return True;
          idx++;
      }
      return false;
   } */
```

[Every time we add char and check if it belongs to list if Yes Then we move to next partions, else we keep on adding]

$O(2^N)$

```
bool partition (string S, list<> L, int idx)
{  if (idx == n) return True;
   string c = "";
   while ( idx < s.length)
         c+=s.charAt(idx);
         if(l.contains(c)){if (partition (S, L, idx+1)) return True;
      } idx++;
   return false;
}
```

// Number of partitions

```
int partition(String s, List L, int idx)
{
    if(idx == n) return 1;

    int ans = 0;
    String c = "";
    while(idx < n)
    {
        c += s.charAt(idx);

        if(l.contains(c))
        {
            ~~if for~~
            ans += partition(s, L, idx);
        }
        idx++;
    }
    return ans;
}
```

// Min no of partitions

```
int partition(String s, List l, int idx, int parts)
```

(2) arr_N: _ ___ ___ ___ _ _______

B_M: 2 5 3

$K = 2$

No. of ways of placing the blocks in a such that there is atleast a gap of k b/w two blocks.

N tasks, arr[i] = time taken to cmplt i^{th} task

K - workers

arr_N : 10 1 2 3 5 4 1 2 9 5 2 2 8 3 2

K = 4

```
Void fmax(int[] a, int N, int K, int idx, int cans){
if(K==1){
        sum = 0;
        for( i → idx to N)
        {
            sum += a[i];
        }
        cans = max(ans, sum);
}
sum = 0
for(i → idx to N) {
        sum += a[i];
        fmax(arr, N, K-1, i+1, max(cans, sum))
}
}
```

// Binary search technique

```
l = max(a), h = sum(a);
while (l ≤ h){
    mid = (l+h)/2;
    if (check(a, N, K, mid)) {
        ans = mid;
        high = mid-1;
    }
    else{
        l = mid+1;
    }
}
```

$O(\log_2^{(h-l+1)} * N)$

```
bool check(int[] a, int N, int K, int m) {
{
    int w = 0; curr = 0;
    for(int i=0; i<N ; i++){
        curr += arr[i];
        if(curr > m){
            w++;
            curr = arr[i];
        }
    return(w ≤ k);
}
```

// Aggressive cows

arr[N]: 2 3 10 14 17 20 24 29 33 39 51 54 70 71

K=5

cows should be placed such that the distance blw two cows shld be maximum.

```
l = 1, h = a[N-1] - a[0], ans = l;

while (l <= h)
{
   m = (l+h)/2
   if( check(a, N, K, m))
   { ans = mid;
     l = mid+1;
   }
   else
     h = mid-1;
}

bool check (int a[], int N, int K, int m)
{
     int c = 1; curr = arr[0];
     for(i=0; i<N; i++){
        if (arr[i] - curr) >= mid){
              c++;
             curr = arr[i];
        }
     }
     return ( c >= K)
```

- Given 2 sorted arrays A_N and B_N. Find the median of combined A and B

A_N : -2 1 3 5 11

B_N : -6 -1 2 4 7 10 13 21

Notes: len(A), len(B) => odd, No duplicates in A & B

Solutions

1. N+M, N+M

Two pointer, new array

2. $(N+M)/2$, 1

Two pointer till mid, no new array

12/11/24

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
A:	a	b	p	x	y	t	p	x	t	p	x	a	b	x	t	a	b	c	x	y	t	b	x	y

	0	1	2	3	4	5	6	7	8	9
B:	a	p	x	y	t	b	x	y	x	t

Q

i	j	k	l
2	5	1	4
7	8	8	9
4	8	3	7
~~12~~	~~7~~	~~13~~	~~23~~
18	23	2	7

$A[i:j] == B[k:l]$?

Good hash

$(\sum s[i] * p^{(i+1)}) \% M$

To avoid collisions we use two hash function taking primes P1, P2

solution

① Two pointers

Q * min(N,M), I

② Use string hash

Precomputations

power arrays

```
// P1 = power array of P1
// P2 = power array of P2
P1[0] = P
for (i=1; i<=N; i++)
    P1[i] = P1[i-1]*P;

P2[0] = q
for (i=1; i<=N; i++)
    P2[i] = P2[i-1]*q;
```

//Prehash arrays

```
PhA1[0] = A[0] * P1[0]; PhA2[0] = A[0] * P2[0];
for(i=1; i<=N; i++){
    phA1[i] = (phA1[i-1] + (A[i] * P1[i+1]) %M) %M
    phA2[i] = (phA2[i-1] + (A[i] * P2[i+1]) %M) %M
}
```

```
phB1[0] = B[0] * P1[0];
phB2[0] = B[0] * P2[0];
for(i=1; i<N; i++)
{
    phB1[i] = phB1[i-1] + (B[i] * P1[i+1])
    phB2[i] = phB2[i-1] + (B[i] * P2[i+1])
}
query = q
for (int t=0; t<q; t++)
{
    Read i, j, k, l;
if(i>0)
    hashA1 = phA1[j] - phA1[i-1]
else hashA2 = phA2[j] - phA2[i-1]
    hashA1 = phA1[j]
    hashA2 = phA2[j]
 if (k>0)
     hashB1 = phB1[l] - phB1[k-1]
     hashB2 = phB2[l] - phB2[k-1]
  else
     hashB1 = phB1[l]
     hashB2 = phB2[l]
 diff = abs(k-i)
 if (i<k)
     hashA1 = hashA1 * P1[diff]
     hashA2 = hashA2 * P2[diff]
 else
     hashB1 = hashB1 * P1[diff]
     hashB2 = hashB2 * P2[diff]
return ((hashA1 == hashB1) && (hashA2 == hashB2))
```

(IV)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A:	a	b	x	y	a	p	q	x	t	b	p	q	x	y	a	b	x	t	z	p	x

	0	1	2	3	4	5
B_M:	b	p	q	x	y	a

(is B a substring of A?)

Solutions

① Brute force
$(N-M)*M$, 1

② Rabin Karp
N, 1

③ KMP (TO-DO)

~~precompute pows array~~

```
hashA = 0
hashB = 0
pow = p
for(i: 0 → M-1) {
    hashA = hashA + (A[i] * pow)
    hashB = hashB + (B[i] * pow)
    pow = pow * p
if (hashA == hashB) return True;
rpow = p
for (i: M → N-1)
{
    hashA = hashA + (A[i] * pow);
    hashA = hashA - (A[i-M] * rpow);
    hashB = hashB * p ;
    pow = pow * p;
    rpow = rpow * p;
    if( hashA == hashB) return true;
}
return false;
}
```

↗ Rolling hash.

// TO make it better we can use double hashing

Str: x p t x y q a b b a q a b b a q y x y z b a

longest palindromic Substring

solutions

① (log N) * N

↓ Binary search

↓ string is palindrome or not

```
bool isPrime(int N){
    for(int i=0; i<=√N ; i++) {
        if(n % i == 0)
        {
            if(i != n/i)
                count+=2;
            else
                count+=1;
        }
        if(count > 2) return false;
    }
    return true;
}
```

complexity

① $\sqrt{N}$, 1

```
void allPrime(int N){
    for(i → 2:N)
    {
        if(isPrime(i))
            print(i);
    }
```

complexity (TO - DO)

```
void allPrime(int N){
    bool[] p = new bool[N+1]
    b[0] = false;
    b[1] = false;
    for(i = 2 → √N)
    {
        ~~if(prime[i]~~
        if(P[i] == T){
            for(int j = 2*i; j<=N; j=j+i){
                P[j] = false;
            }
        }
    }
    for(2 → N)
    {
        if(P[i])
            print(i)
    }
```

1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35
36 37 38 39 40 41 42
43 44 45 46 47 48 49

Sieve of erastosthenes

$\frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \frac{N}{P} \cdots$

$N + N\left[\frac{1}{2} + \frac{1}{3} + \frac{1}{5} \cdots + \frac{1}{P}\right]$ (TO-∞)

14/11/24

A_N : -3 1 8 10 20 6 30 16

K : 4 Subset sum K

-3 1 8	10 20 6
{} = 0	{} = 0
{-3} = -3	{10} = 10
{1} = 1	{20} = 20
{8} = 8	{6} = 6
{-3,1} = -2	{10,20} = 30
{-3,8} = 5	{10,6} = 16
{1,8} = 9	{20,6} = 26
{-3,1,8} = 6	{10,20,6} = 36

Solutions

① $N \times 2^N$, 1
(generate all subsets)

② 2^N, 1
(Recursion)

③ $2^{N/2} \times 1 + 2^{N/2} \times 1$, $2^{N/2}$

Adding first half sums to a hash set

Let generated sum be 'a', search for k-a in hash set.

(Meet in the middle)
(MITM)

$(a+b+c+d) = K$

$a+b = k-(c+d)$ (with MITM)

Pseudo

$a[i] \wedge b[j] = c[k] \wedge d[l]$.

a_d : 0 1 6 7 9 16 18 34 36 40

rotated to d times

d=4

18 34 36 40 0 1 6 7 9 16

Solutions

① LS → N, 1

② $\log_2^N + \log_2^N$, 1

BS 0 to d-1 BS d to N

③ Check range and search

⇒ $\log N_2$, 1

! if d is unknown !

To find d we binary search

```
int findd(int[] a, int n){
 l=0, h=n-1, ans=-1;
while(l<=h)
 {
   m=(l+h)/2;
   if(a[0] < a[mid])
      l = m+1;
   else{
      h = m-1;
      ans=m;
   }
 }
return ans;
}
```

Solution

① $N + \log N_2$, 1

LS to find d — binary search on req side

② $\log N + \log N$, 1

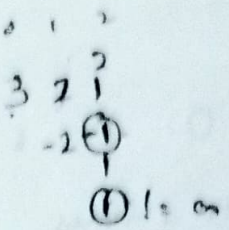
bs to find d — bs to find element

③ $\log N_2$

a single binary search

Single bs

```
l=0, h=n-1
while(l<=h)
{ mid = (l+h)/2
  if(a[m] == k) return m;
  if(mid ∈ left) //(ar[0] < ar[mid])
     if (k ∈ (l0, mid))
        h = m-1;
     else
        l = m+1;
  else
     if (k ∈ (mid, hi))
        l = m+1;
     else h = m-1;
}
```



Q: N : -5 10 4 1 6 2 -1 8 12 16 1 8

④ Inplace solution.

First mark the invalid elements as '∞'

valid range [1, N]

if number 'n' is present make the number at 'n-1' idx as negative, if it is negative do not change.

After this, the first element which positive, then ans is idx+1,

0	1	2	3	4	5	6	7	8	9	10	11
-5	10	4	1	6	2	-1	8	12	16	1	8
∞	10	4	1	6	2	∞	8	12	∞	1	8
-∞	-10	+4	-1	6	-2	∞	-8	12	-∞	1	-8

first +ve at idx→2 so ans = 3

First missing positive integer

Solutions

① N×N, 1

(for all possibilities, search in the array)

② NlogN + N, 1
↓ sort ↓ LS

③ N + N, N
↓ populate hashset ↓ search

in place soln for freq

0	1	2	3
1	2	3	3

101 102 103
203

1 * 2 → divide with 100 → frequencies of idx+1

Given a binary array, find longest subarr with equal 0's & 1's

solutions

① $N^2 \times N$, 1 — Brute force

② N, N — PS + Hash map

A_N:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
	1	1	1	0	0	1	0	1	1	1	0	1	0	0
	1	1	1	-1	-1	1	-1	1	1	1	-1	1	-1	-1
PS:	1	2	3	2	1	2	1	2	3	4	3	4	3	2

Store a hashmap with first occurrence index

if u find an element which is already in hashmap

Then update ans with diff.

```
int equal01(int[] a, int n)
{
  int[] ps = int[n];
  int ans = 0;
  for(int i=0; i<n; i++)
  {
    if(a[i]==0) a[i]=-1;
  }
  ps[0]=a[0];
  for(int i=1; i<n; i++)
  {
    ps[i] = ps[i-1]+a[i];
  }
  Map<Int, Int> h = new HashMap<>();
  for(int i=0; i<n; i++)
  {
    if(h.containsKey(ps[i])){
      ans = Math.max(i - h.get(ps[i]), ans);
    }
    else{
      h.put(ps[i], i);
    }
  }
  return ans;
}
```

```
c=0, ans=0;
Map<int,int> hm = new HashMap<>();
for(int i=0; i<N; i++){
  ele = ar[i];
  c += (ele==-1) ? 1 : -1;
  if(c in hm)
    ans = max(ans, (i-hm[c]))
  else hm[c] = i;
}
```

LAB

1. Quad of XOR
2. First missing +ve integer
3. Equal 0's & 1's
4. Subsequence sum
5. Search in sorted rotated array [LC]
6. longest subarray having equeven no. of vowell count [LC]

19/11/24

A: 5 1 4 8 3 9 12

```
long ans=0;
for(int i=0;i<30;i++)
{ int count=0, c=1;
  for(int j : a)
  {
    if((j>>i)&1 == 1)
    {
      count += (c * (N-i));
      c=1;
    }
    else{
      c++;
    }
  }
  ans += (count*(1<<i));
}
```

Find the sum of OR of all subarrays

Solutions

① N^3, 1
Brute Force

② N^2, 1
Carry forward

③ $30*N$, 1

Let req ans be $O_1 + O_2 + O_3 \cdots$

Let us say $5 + 7 + 24 \cdots$

$4+1+4+2+1+16+8+\cdots$

$C_0 \times 2^0 + C_1 \times 2^1 + C_2 \times 2^2 + \cdots$

To find C_B

C_B is set only when the OR of a given subarray is set. for a given subarray's OR to be set the subarray must have atleast one element with B^{th} bit set. Hence we iterate over the array if B^{th} bit is set we add $(N-i)$, if unset we increase a 'c' variable until we find a number with B^{th} bit as set. Then we add $c*(N-i)$ to count the subarrays which start with unset numbers.

	0	1	2	3	4	5	6	7	8	9	10	11	12
A_N:	5	-1	8	6	-1	4	2	1	2	9	3	11	6

```
void greatest(int a[], int B[], int n)
{
    list<Integer> S = new ArrayList<>();
    S.add(0);
    for(int i=0; i<n; i++)
    {
        if(a[i] <= S.get(S.size()-1))
            S.add(i);
        else{
            while(a[S.get(S.size()-1)] < a[i])
            {
            int j = S.remove(S.size()-1);
            }, B[j] = a[i];
        }
    }
```

populate array B such that
B[i] = first greater element on the right

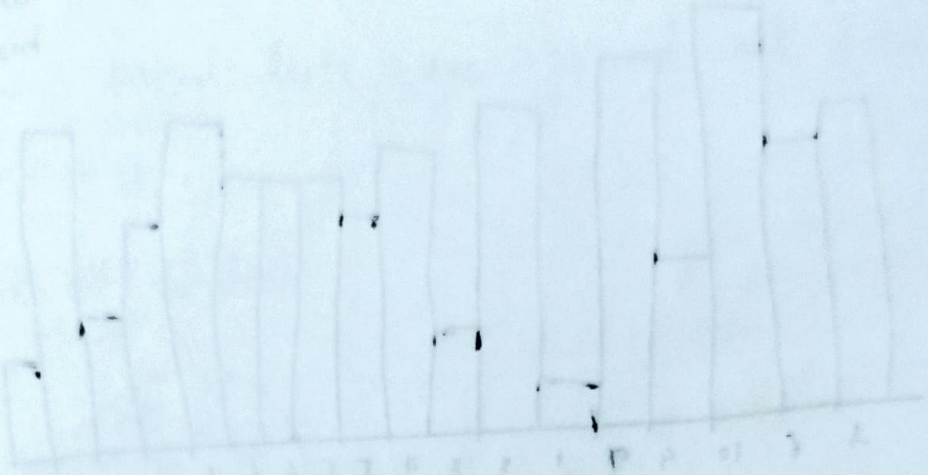
Solutions

① N^2, 1
Brute force

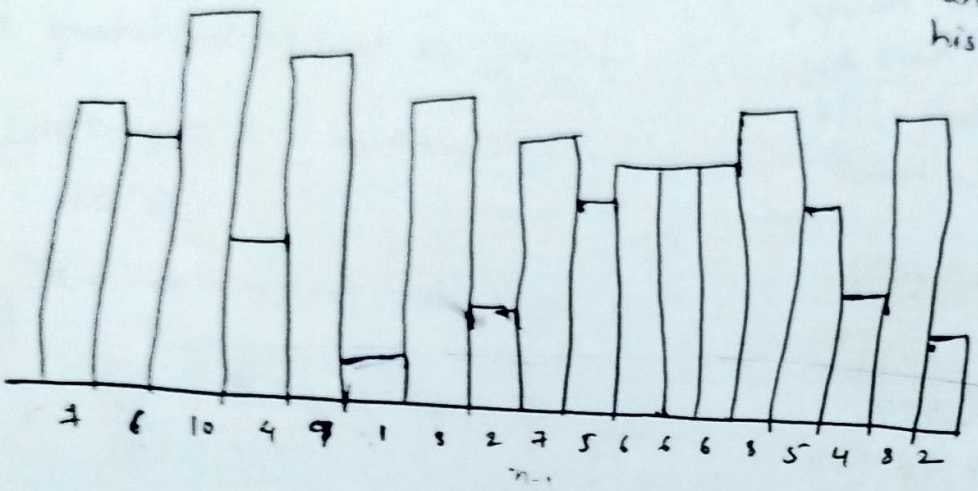
② N, N ↳ for stack
(front & back)
↓
(TO-DO)

```
void greatest(int a[], int b[], int n) {
    Stack<Integer> S = new stack<>();
    for(int i=0; ; i<n; i++){
        while(!S.empty && a[S.top] < a[i]) {
            int idx = S.top();
            S.pop()
            B[idx] = a[i]
        }
        S.push(i);
    while(!S.empty())
        idx = S.pop()
        B[ind] = -∞
```

// Back to front (TO DO)



④ Max rectangular area under histogram



lmin[], rmin[]

ans = max(ans, ((rmin[i] - lmin[i] - 1) * a[i]));

We find the left smallest and right smallest building for current building and find the area of rectangle. For that we precompute them using prev.

Solutions

① N^2, 1

Brute force : for every ele move on its right & left end multiply.

② N+N+N, 3N

A_N: 5 9 1 2 3 6 8 4 1 9 2

K : 4

print max of every subarray of size K

Double ended queue DEQUE (deck)

- Before adding every element, remove all the elements which are lesser than the current element and then append it

Solutions

① $(N-K+1)^*K$, 1

→ when we are moving to next window make sure that the left most elements are popped as well.