TLE :- Time Limit exceeded.

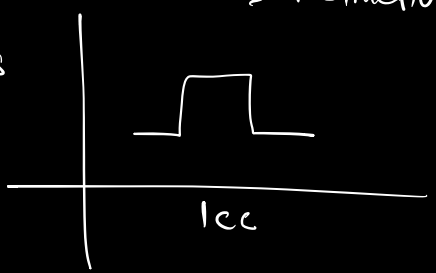
: Why TLE occurs ?

⇒ Code ⇒ Servers ⇒ $1\ GHz$ ⇒ 10^9 cc / sec

↳ instructions

↓

10^9 instructions/s



: In general, time limit in online service ⇒ 1 sec

time ⇒ 1 sec speed ⇒ $1\ GHz$

instructions allowed to compute ⇒ 10^9 (max).

eg ⇒

```
for(i=0; i<N; i++){
    if(i%2==0)
        print(even)
    else
        print(odd)
}
```

: iterations → N.

: instructions per iteration → 5

: total no. of instructions ⇒ 5N

eg ⇒ (1) 1 iteration ⇒ 10 instructions

⇒ max^m instructions allowed ⇒ 10^9

⇒ " iterations ⇒ $\frac{10^9}{10}$ ⇒ 10^8 iterations

(2) 1 iteration ⇒ 100 instructions

⇒ max^m instructions ⇒ 10^9

⇒ max^m iterations ⇒ $\frac{10^9}{100}$ ⇒ 10^7

In general, 1 iteration should have [10-100] instructions.

⇒ max^m instructions ⇒ 10^9

So, we can have $[10^7 - 10^8]$ iterations

⇒ every question, has a constraint :-

↓

defined range of input values.

⇒ Q Given array[N], calculate no. of eqm indexes.

Constraints :- $1 \leq N \leq 10^5$

$1 \leq arr[i] \leq 10^9$

Soln-1

a) Brute force soln with nested loops.

TC ⇒ $O(N^2)$

↓

at max iteration ⇒ $N = 10^5 \Rightarrow (N)^2 \Rightarrow 10^{10}$ (X)

b) ⇒ constraint ⇒ $1 \leq N \leq 10^3$.

at max iterations ⇒ N^2

⇒ $(10^3)^2 \Rightarrow 10^6$ (✓).

Soln-2

Solved using a loop & PF-sum

TC ⇒ $O(N)$

At max ⇒ N ⇒ 10^5 (✓)

iteration

Q1

constraints ⇒ $1 \leq N \leq 10^3$

$1 \leq arr[i] \leq 10^7$

Soln 1 ⇒ $O(N^3)$ ⇒ max^m iterations ⇒ $(10^3)^3$ ⇒ 10^9 (XX)

Soln 2 ⇒ $O(N^2)$ ⇒ max^m iteration ⇒ $(10^3)^2$ ⇒ 10^6 (✓)

↓

$O(N \log N)$

↓

$O(N)$

: TRICK

i) Read Question

ii) Read constraints

iii) Come up with a logic

→ Soln → Check TC & max^m iteration

↓

decide to code or not

: Max^m Subarray Sum :-

constraints ⇒ $1 \leq N \leq 10^6$ ⇒ $O(N^2)$ ⇒ $(10^6)^2 \Rightarrow 10^{12}$

|

(XXX)

Optimised $O(N) \Rightarrow 10^6$ (✓)

↓

Kadane's Algo

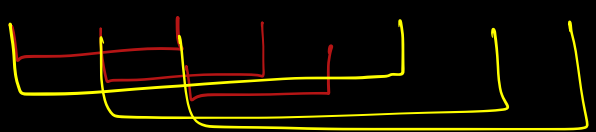
Advanced Batch

Q.1. Given arr[N], print start & end index of all subarrays of size 'k'.

eg ⇒ arr[12] ⇒ 3 4 2 -1 6 7 8 9 3 2 -1 4

0 1 2 3 4 5 6 7 8 9 10 11

k = 3.



O/P

[0 2]
[1 3]
[2 4]
[3 5]
[4 6]
[5 7]
[6 8]
[7 9]

k = 6

[0 5]
[1 6]
[2 7]
[3 8]
[4 9]
[5 10]
[6 11]

[s e] ⇒ e - s + 1.

[0 e] ⇒ k

⇒ e ⇒ k - 1

k = e - 0 + 1

⇒ e = k - 1

[8 10]
[9 11]

Generalise

subarray of size k.

[0 k-1].

[1 k]

[2 k+1]

[3 k+2]

[4 k+3]

.
.
.

[N-k N-1] → last subarray.

[S N-1] ⇒ k

⇒ (N-1) - S + 1 = k

⇒ S = N-1+1-k

⇒ S = N-k

pseudo

```
S = 0, e = k-1
while( e <= (N-1) {
    print (S, e)
    S++
    e++
}
```

```
S = 0, e = k-1
while(S <= (N-k)){
    print(S, e)
    S++
    e++
}
```

TC $\Rightarrow O(N-K)$

SC $\Rightarrow O(1)$

$\Rightarrow$ How many subarrays are there with len = k ?

s $\Rightarrow$ $[0, N-K]$

no. of subarrays $\Rightarrow$ $N-K-0+1$

$\Rightarrow$ $\boxed{N-K+1}$

Q 2. Given arr[N], find max subarray sum of len = k.

arr[10] $\Rightarrow$	-3	4	-2	5	3	-2	8	2	-1	4
	0	1	2	3	4	5	6	7	8	9

K = 5.

s	e	→ sum
0	4	7
1	5	8
2	6	12
3	7	16
4	8	10
5	9	11

Op = 16

Soln 1. for every subarray of size k, find it sum and maintain overall.

pseudo

```
msum = INT_MIN
s = 0
e = k-1
while ( e <= (N-1) ) {
    sum = 0
    for( i=s; i<=e; i++) {
        sum = sum + arr[i]
    }
    msum = max(msum, sum)
    s++;
    e++;
}
```

TC => O((N-k+1) * k)

=> O((N-k) * k)

SC => O(1)

for what value of k we can get worst possible TC.

k = 1 TC => O((N-1)+1)

=> O(N)

$k = N \Rightarrow TC \Rightarrow O((N-N+1) * N)$

$\Rightarrow O(N)$

$k = N/2 \quad TC \Rightarrow O\left(\left(N - N/2\right) * \left(N/2\right)\right)$

$\Rightarrow O\left(\frac{N^2}{4}\right)$

$\Rightarrow O(N^2)$

Worst case :- $k = N/2$

$TC \Rightarrow O(N^2)$

Soln-2 Optimised Soln using → pf sum

Subarray $\Rightarrow [s \; e] \Rightarrow sum(s\,e)$.

pseudo

1) create pfsum Array

2) $s=0$, $e=k-1$, $wsum = INT\_MIN$.

while ($e < N$) {

sum = [pfsum queries]

msum = max(msum, sum)

}

TC ⇒ O(N + N-K+1)

⇒ O(2N-K)

⇒ O(N-K)

worst case K = 1

TC ⇒ O(N + N-1)

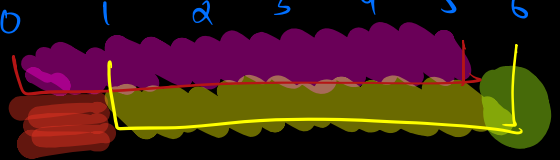
⇒ O(2N)

⇒ O(N)

SC ⇒ O(N)

eg arr[10] ⇒ -3 4 -2 5 3 -2 8 2 1 4

0 1 2 3 4 5 6 7 8 9



K = 6

[0 5] ⇒ 5

pink - red + green = yellow

[1 6] ⇒ 5 - arr[0] + arr[6] = 5 - (-3) + 8 = 16.

[2 7] => 16 - arr[1] + arr[7] = 16 - 4 + 2 = 14.

[3 8] => 14 - arr[2] + arr[8] = 14 - (-2) + 1 = 17.

[4 9] => 17 - arr[3] + arr[9] => 17 - 5 + 4 = 16.

: Sliding Window Technique

generalize

N, k

1st subs [0 k-1] => iterate from [0 k-1] => sum.

2nd subs [1 k] => sum = sum - arr[0] + arr[k].

3rd subs [2 k+1] => sum = sum - arr[1] + arr[k+1]

4th subs [3 k+2] => sum = sum - arr[2] + arr[k+2].

:

get max sum.

Pseudo.

```
msum = INT_MIN.
sum = 0.
for(i=0; i<=(k-1); i++) {
        sum = sum + arr[i]
}
msum = max(msum, sum);
s=1
e=k
while(e<N) {
 sum = sum - arr[s-1] + arr[e]
 msum = max(msum, sum)
 s++
 e++
}
```

total iterate $\Rightarrow$ $k + (N-k) \Rightarrow N$

$TC \Rightarrow O(N)$

$SC \Rightarrow O(1)$

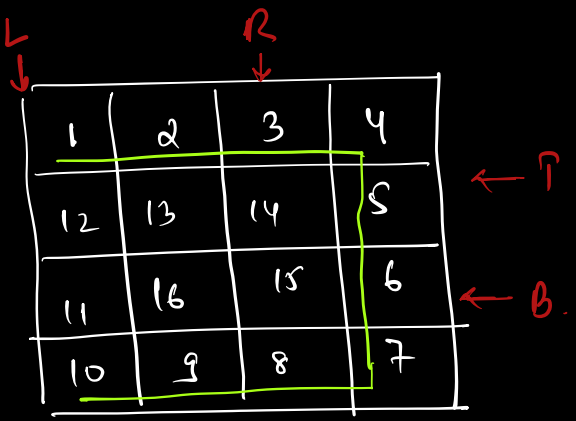
Q4. Spiral Order Matrix

Given a no. N, Create a 2D matrix of size N x N, containing nos. from 1 - N^2, in spiral order.

N = 4.

1 to 4^2

⇒ 1 to 16



D = 0, 1, 2, 3.

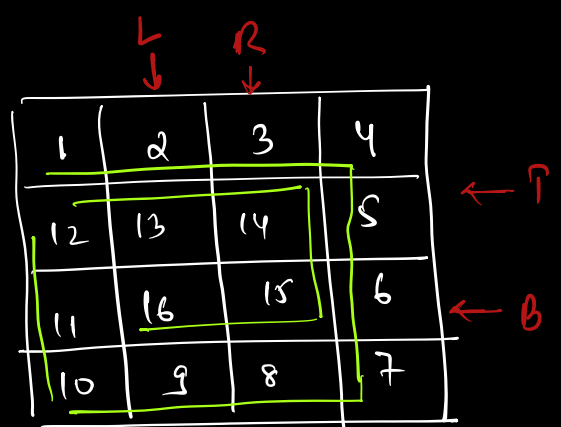
T = 0 L = 0 R = N-1 B = N-1.

k = 1

```
while ( k <= N^2) {
    if (D == 0){
        for(i = L; i <= R; i++) {
            arr[T][i] = k++
        }
        T++
        D = 1
```

```
if( D==1)
    for(i= T; i<= B ;i++) {
        arr[T][R] = k++ .
    }
    R--;
    D= 2
if( D==2) {
    for(i=R; i>=L; i--) {
        arr[B][i] = k++
    } B--;
    D= 3

if( D==3) {
for(i=B; i>=T; i--){
arr[i][L] = k++
}
  L++
  D==0,
}
```



TC = O(N²)

SC = O(N²)