: Why TLE occurs ?

↓

Time Limit exceeded.

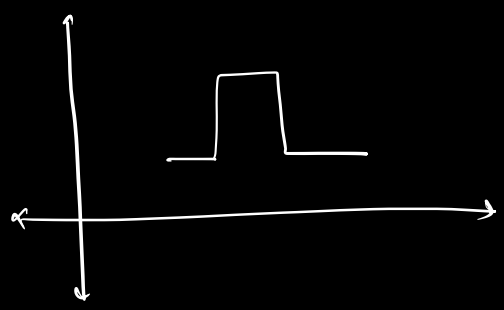
⇒ Code executes in a server

↓

assume ⇒ 1 GHz

⇒ 10^9 cc/s.

: In general, time limit in an online service is 1 sec



So, time ⇒ 1s

speed ⇒ 10^9 cc/s

or, 10^9 instructions/sec

1 cc = 1 instruction

→ max^m no. of instruction ⇒ 10^9.

eg ⇒

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

: iterations → N

: instruction → 5 per iteration

: total instruction → 5 * N = 5N

Ex ⇒ assume 1 iteration ⇒ 10 instructions.

⇒ N iterations ⇒ 10N instructions.

max^m instructions ⇒ 10^9.

max^m iterations

⇒ $10N = 10^9$

⇒ $N = \frac{10^9}{10} = 10^8$ [iterations].

assumption-2

1 iteration ⇒ 100 instructions

N iterations ⇒ 100N instructions

max^m instruction ⇒ 10^9.

max^m iterations ⇒ $100N = 10^9$

⇒ $N = 10^7$

[max^m iterations].

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

In general, max^m iterations, we should have

$[10^7 - 10^8]$

Q. Given N array elements, calculate eqm index.

Constraints ⇒ $1 \le N \le 10^5$

$1 \le arr[i] \le 10^9$.

Soln 1 Brute force Soln.

two nested loops.

TC ⇒ $O(N^2)$.

maxm iterations ⇒ $(10^5)^2 \Rightarrow 10^{10}$ - [xxx].

Soln 2 PFsum solution

TC ⇒ $O(N)$

maxm iteration ⇒ $(10^5) \Rightarrow 10^5$ [ttc]

Q 2. constraints :

$1 \le N \le 10^3$ ← size of array

$1 \le arr[i] \le 10^7$ ← size of each element of array

Soln 1 ⇒ $O(N^3)$ → maxm iter ⇒ $(10^3)^3 \Rightarrow 10^9$ [xxx]

Soln 2 ⇒ $O(N^2)$ → maxm iter ⇒ $(10^3)^2 \Rightarrow 10^6$ [ttc]

$O(N \log N)$

$O(N)$.

⇒ Flow.

1) Read the question
2) Read the constraints
3) Come up with logic
   ↓
   worst case / maximum iterations
   ↓
   decide to code or not

⇒ maxm subarray sum :-

Constraints ⇒ $1 \leq N \leq 10^6$

BOM ⇒ TC ⇒ $O(N^2)$
↓
maxm iter ⇒ $(10^6)^2 \Rightarrow 10^{12}$ [xxx].

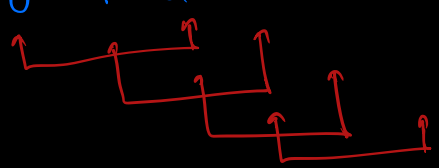
$O(N)$ → maxm iter ⇒ $(10^6) \rightarrow 10^6$ [yes].

How → Kadane's Algo

Advanced Batch.

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

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
8	10
9	11

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 - 0 + 1 = k

⇒ e = k - 1

Generalise

length = N.

size = k.

1st subarray

[0 k-1]

[1 k]

[2 k+1]

[3 k+2]

[4 k+3]

(

[N-k N-1]

[s e] ⇒ k

⇒ [s N-1] ⇒ k

⇒ (N-1) - s + 1 = k

⇒ N - s = k

⇒ s = N-k

pseudo

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

TC ⇒ O(N-k)
SC ⇒ O(1)

⇒ How many subarrays are possible of size 'k', in array of length N

s ⇒ [0 N-k]

⇒ N-k-0+1

⇒ N-k+1 → no. of subarrays

Q2. Given an arr[N], find max subarray sum of len k.

arr[10] ⇒	-3	4	-2	5	3	-2	8	2	-1	4
	0	1	2	3	4	5	6	7	8	9

k = 5

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

O/P = 16

Soln - 1

For every subarray of size k, find sum & maintain maxm sum.

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

TC ⇒ $O((N-k+1) * k)$.

worst possible TC in terms of N.

k = 1

TC ⇒ $O((N-1+1) * 1)$

⇒ $O(N)$

$k = N$

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

$\Rightarrow O(N)$.

$k = N/2$

$TC \Rightarrow O((N - N/2 + 1) * (N/2))$

$\Rightarrow O((N/2 + 1) * N/2)$

$\Rightarrow O(\frac{N^2}{4} + N/2)$

$\Rightarrow O(N^2)$

worst $TC \Rightarrow O(N^2)$, when $k = N/2$

$SC \Rightarrow O(1)$

Optimisation ①

Sum of subarray $\Rightarrow [s-e]$.

```
msum = INT_MIN.
s=0  e=k-1
while( e<N){
    sum = { find sum
            using pfsum queries }
    s++
    e++
}   msum = max(sum, msum)
```

TC ⇒ $O(N + N-K)$

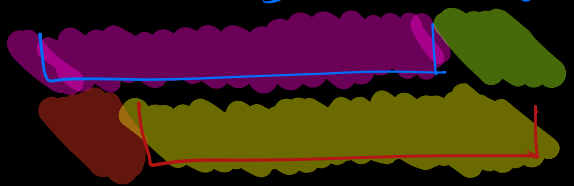
⇒ $O(2N-K)$ ⇒ final ⇒ $O(N)$

SC ⇒ $O(N)$.

Optimised 2

arr[10] ⇒ -3 4 -2 5 3 -2 8 2 1 4

0 1 2 3 4 5 6 7 8 9



K = 6.

sum

[0 5] → 5

[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 → Ans

[4 9] → 17 − arr[3] + arr[9] ⇒ 17 − 5 + 4 = 16.

Sliding Window Technique

generalise

N, k

1st subarr [0 k-1] → iterate from [0 k-1] & get sum.

2nd subarray [1 k] → sum = sum − arr[0] + arr[k].

3rd subarr [2 k+1] → sum = sum − arr[1] + arr[k+1].

⋮

till end → maintain max sum

pseudo

```
mSum = INT_MAX
sum = 0.

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

```
sum = sum - arr[s-1] + arr[e]
msum = max(msum, sum)
s++
e++
}
```

(K + N - K)

TC => O(N).

SC => O(1)

Q 3. Spiral Order Matrix :-

Give a no. N, create a matrix (N×N) containing all nos. from 1 to N^2 in spiral form.

N = 4

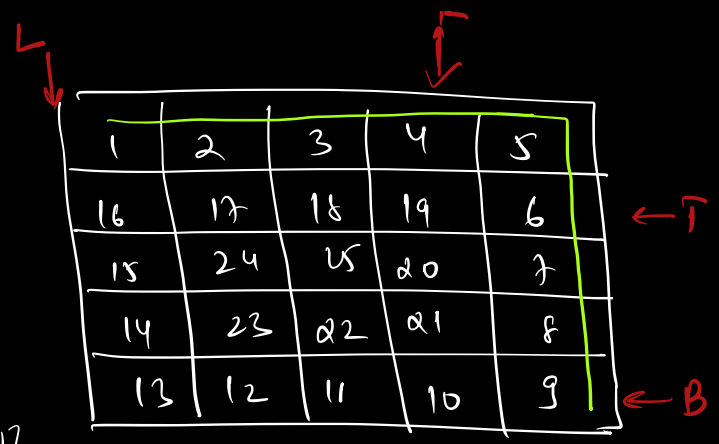
1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

N = 5

D = 0, 1, 2, 3

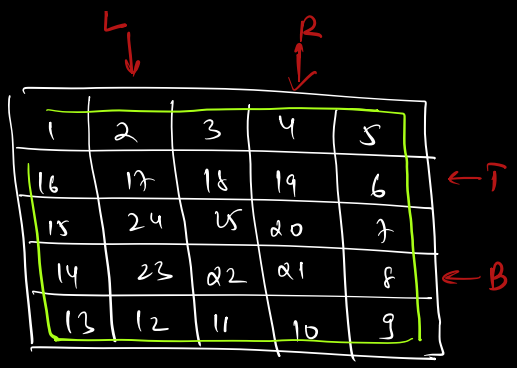
L = 0 T = 0

R = N-1 B = N-1




```
int[][] arr = new int[N][N]
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[i][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)