Approach / Rough Work

Assignment

(Introduction to Problem Solving - I)

Q1. Maximum Subarray Easy </> Solved

Using hints except Complete Solution is Penalty free now — Use Hint

Problem Description

You are given an integer array **C** of size **A**. Now you need to find a subarray (contiguous elements) so that the sum of contiguous elements is maximum.
But the sum must not exceed **B**.

Problem Constraints

$1 <= A <= 10^3$
$1 <= B <= 10^9$
$1 <= C[i] <= 10^6$

Input Format

The first argument is the integer A.
The second argument is the integer B.
The third argument is the integer array C.

Output Format

Return a single integer which denotes the maximum sum.

Example Input

Input 1:

```
A = 5
B = 12
C = [2, 1, 3, 4, 5]
```

Input 2:

```
A = 3
B = 1
C = [2, 2, 2]
```

Example Output

Output 1:

```
12
```

Output 2:

```
0
```

Explanation 1:

```
We can select {3,4,5} which sums up to 12
which is the maximum possible sum.
```

Explanation 2:

```
All elements are greater than B, which means
we cannot select any subarray.
Hence, the answer is 0.
```

$\Rightarrow$ Size of input array A = 10^3 (maximum)

$\Rightarrow$ A^2 iterations max allowed

2	1	3	4	5
0	1	2	3	4

If we iterate through all possible Subarray indexes ...

```
for (int i = 0; i < A; i++) {
    int sum = 0;
    for (int j = i; j < A; j++) {
        //print (i,j)
        // sum += C[j]
```

→ this inner 'j' loop will give all the possible index-range for all sub-arrays

→ we can have a prefix-sum array configured previously, which can calculate the sum in range [i,j], and keep on checking where the sum is the maximum

OR

→ we can keep on carry forwarding the sum

Observing Behaviour of sum

i = 0

sum = 0;

j = 0

sum = C[0] → sum of subarray [0]

j = 1

sum = C[0] + C[1]

j = 2

sum = C[0] + C[1] + C[2] → sum of subarray [0,1,2]

↓

sum of subarray [0, 2]

⋮

j = N-1

sum = C[0] + C[1] + C[2] + ... + C[N-2] + C[N-1]

sum of subarray [0, 1, .., N-1]

Approach

→ After each iteration of 'j' check whether current sum value is maximum or not till yet, at the same time being less than or equal to B

→ If we approach a sum value which is equal to B, we can stop the iteration and return

Edge Case

→ If every element of C[] is greater than B, there is no such subarray with maximum subarray sum ≤ B

→ straight away return 0

Skeleton

```
// finding min
min = C[0]
for(int i = 1; i < A; i++) {
    if(C[i] < min)  min = C[i];
}

// edge case
if(min > B) {
    return 0;
} else if(min == B) {  // optional
    return B;
}

for(int i = 0; i < A; i++) {
    int sum = 0;
    for(int j = i; j < A; j++) {
        sum += C[j];
        max = Max(sum, B);
    }
    → optional: if(max == B) return B;
}

return max;
```

T.C. → $O(N^2)$

S.C. → $O(1)$

Why Carry forward over Prefix Sum?

→ Both have T.C. $O(N^2)$, however if we had used prefix-sum where modification of array is not allowed, we would have to initialise an array, thereby increasing Space Complexity to $O(N)$