```java
public int MaxSum(int[] arr, int n) {
    // Write your code here
    int maxTrain = 0; // -inf
    int maxSumSubarray = Integer.MIN_VALUE; // -inf

    int minTrain = 0; // +inf
    int minSumSubarray = Integer.MAX_VALUE; // +inf

    int total = 0;

    for(int i = 0;i<n;i++){
        total+=arr[i];

        //Calc maxSumSubarray
        maxTrain = Math.max(arr[i],arr[i] + maxTrain);
        maxSumSubarray = Math.max(maxSumSubarray,maxTrain);

        //Calc minSumSubarray
        minTrain = Math.min(arr[i],arr[i]+minTrain);
        minSumSubarray = Math.min(minSumSubarray,minTrain);
    }

    //if all are -ve
    if(total == minSumSubarray) return maxSumSubarray;

    return Math.max(maxSumSubarray , total - minSumSubarray);
                    14                  19
}
```

10 -2 -3 5 4

MaxSS = 14
MinSS = -5
+S = 19

-10 4 16 -10

10 20 30 40

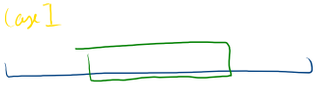
-10 -20 -30 -40

train = -100
MSS = -100
MSI = -10

-10, -100 + 100 ⇒ 0

(Max Sum Subarray, Total Sum − Min Sum Subarray)

Case 1



⇒ Max Sum Subarray

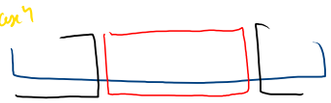
Case 2

⇒ Max Sum Subarray

Case 3

⇒ Max Sum Subarray.

Case 4



⇒ Total Sum − Min Sum Subarray

.

There is a car with k empty seats. The vehicle only drives east (i.e., it cannot turn around and drive west).

You are given the integer capacity and an array trips where trips[i] = [numPassengers, from, to] indicates that the ith trip has numPassengers passengers and the locations to pick them up and drop them off are from and to respectively. The locations are given as the number of kilometers due east from the car's initial location.

Return true if it is possible to pick up and drop off all passengers for all the given trips, or false otherwise.

Input Format

The first line of input contains an integer n which represents the size of the array.

The next n lines contains 3 integers representing number of passengers, starting point and ending point respectively.

The next line contains an integer k which represents the capacity of the car.

Output Format

Print true or false whether it is possible to pick up and drop off all the passengers for all the given trips.

Sum of Maximum Sum Subarray

5 -2 -6 7 3 -9 7 -3 4 10 5 -4 2 -10 -16 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

0 5 3 -3 9 10 1 8 5 9 19 20 22 12 -4 4

MSS = 19

int maxi = -∞

for(i=0; i<n; i++){

startsNew = arr[i]

BrewTrain = train + arr[i]

train = Max(startsNew, BrewTrain)

maxF = Max(maxi, train);

}

Sum of Maximum Sum Subarray

5 -2 -6 7 3 -9 7 -3 4 10 5 -4 2 -10 -16 4

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

5 3 -3 7 10 1 8 5 9 19

Maxi = 19 train = 19

asp = 3 csp = 3

aep = 9 cep = 9

-ve

New New

New New

There is a car with k empty seats. The vehicle only drives east (i.e., it cannot turn around and drive west).

You are given the integer capacity and an array trips where trips[i] = [numPassengers, from, to] indicates that the ith trip has numPassengers passengers and the locations to pick them up and drop them off are from and to respectively. The locations are given as the number of kilometers due east from the car's initial location.

Return true if it is possible to pick up and drop off all passengers for all the given trips, or false otherwise.

Input Format

The first line of input contains an integer n which represents the size of the array.

The next n lines contains 3 integers representing number of passengers, starting point and ending point respectively.

The next line contains an integer k which represents the capacity of the car.

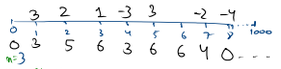
Output Format

Print true or false whether it is possible to pick up and drop off all the passengers for all the given trips.

```java
public boolean carPooling(int[][] trips, int capacity) {
    //your code
    int[] station = new int[1001];
    for(int[] trip: trips){
        int numPass = trip[0];
        int from = trip[1];
        int to = trip[2];

        station[from]+=numPass;
        station[to] -= numPass;
    }
    for(int i = 1;i<1001;i++){
        station[i] += station[i-1];
    }
    for(int  i = 0;i<1001;i++){
        if(station[i] > capacity) return false;
    }
    return true;
}
```

K = 6



n = 3

Num. Si ei

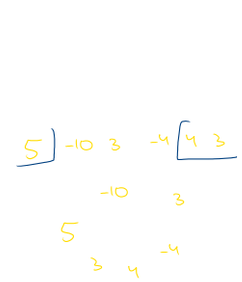
3 1 4

2 2 7

4 5 8

1 3 5

1000



MSS = 9

total -11

MS -11

-10 5 4 -10

-11 -(-11) ⇒ 0