Intuition

We have **two things**, **array** of intergers and **number** x.
We can do **one** operation **each** time select rightmost or leftmost item from the array and **subtract** it from x.
The goal is to make x equal to zero.

Look interesting 🤯
Let's **simplify** our problem a little ?
We only need to know **sum** of numbers from right and left that equal to x.
But how we get this number ? 🧐
Let's see this example:

nums = (3, 4, 7, 1, 3, 8, 2, 4), x = 9

we can see here that the answer of minimum elements from left and right (operations) is 3 which are (3, 2, 4)
There is also something interesting. 🤩
We can see that there is a subarray that we **didn't touch** which is (4, 7, 1, 3, 8)

Let's make a **relation** between them 🚀

sum(3, 4, 7, 1, 3, 8, 2, 4) = sum(4, 7, 1, 3, 8) + sum(3, 2, 4)

sum(3, 4, 7, 1, 3, 8, 2, 4) = sum(4, 7, 1, 3, 8) + x

sum(3, 4, 7, 1, 3, 8, 2, 4) - x = sum(4, 7, 1, 3, 8)

23 = sum(4, 7, 1, 3, 8)

We can see something here. 😀
That the sum subarray that I talked about before is the sum of the whole array - x

Ok we made a **relation** between them but **why** I walked through all of this ?

The reason is that we can **utilize** an efficient technique that is called **Two Pointers**. 🚀 🚀

Thats it, instead of finding the **minimum** number of operations
from leftmost and rightmost elements. We can find the **continous**
subarray that **anyother** element in the array is the **answer** to our **minimum** operations.

And this is the solution for our today problem I hope that you understood it 🚀 🚀

Approach

1. Calculate the total sum of elements.
2. Compute the target value as the difference between the total sum and the provided target x.
3. Check if the target value is negative; if so, return -1 as the target sum is not achievable.
4. Check if the target value is zero; if so, **return** the **size** of nums since we need to subtract **all** of the elements from x.
5. Initialize pointers leftIndex and rightIndex to track a sliding window.
6. Within the loop, check if currentSum exceeds the target value. If it does, increment leftIndex and update currentSum.
7. Whenever currentSum equals the target value, calculate the **minimum** number of operations required and update minOperations.
8. **Return** the **minimum** number of operations.

Complexity

- **Time complexity:**O(N)
  In this method we have two pointers, each of them can iterate over the array at most once. So the complexity is 2 * N which is O(N).
- **Space complexity:**O(1)
  We are storing couple of variables and not storing arrays or other data structure so the complexity is O(1).