Count of Range Sum

Problem Description

Given an integer array ***nums***, return the number of range sums that lie in ***[lower, upper]*** inclusive.

- Range sum ***S(i, j)*** is defined as the sum of the elements in ***nums*** between indices ***i*** and ***j*** ($i \leqslant j$), inclusive.

Example

Input: ***nums*** = [-2, 5, -1], ***lower*** = -2, ***upper*** = 2,

Output: 3

Explanation: S(0, 0) = -2, S(2, 2) = -1, S(0, 2) = -2 + 5 - 1 = 2

Note

Requirement on Complexity

A naïve algorithm of $O(n^2)$ is trivial. You **have to** do better than that.

- Just searching over all ***S(i, j)*** will not pass the tests!
- We will check your code

Try to write an algorithm of $O(n \log n)$

Due

Dec. 4 15:59 CST (Dec. 3 23:59 PST)

Hint

Prefix Sums

To calculate any ***S(i, j)*** in $O(1)$ time

$$P[i] = \sum_{k=0}^{i} nums[k]$$

$$S(i,j) = P[j] - P[i-1]$$

$O(n)$ time for generating array P

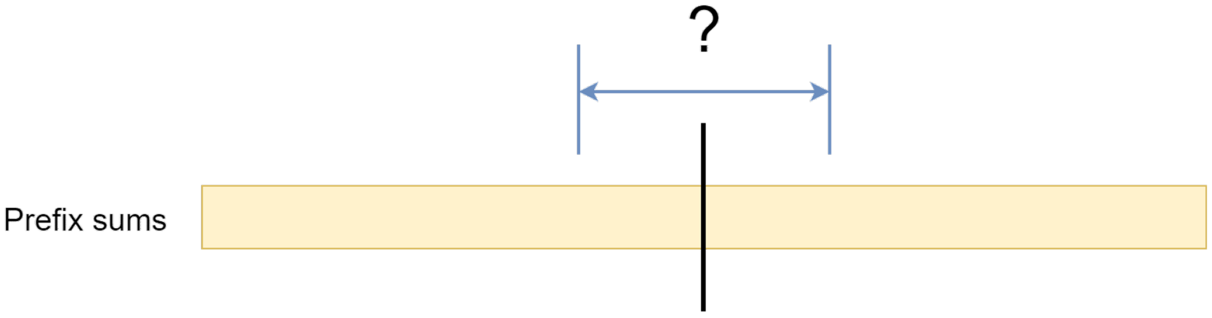
For the first 5 testing cases, `lower` and `upper` are in range $[-2147483648, 2147483647]$.

… while the last 5 testing cases are not. Remember to use **long long** type if you write C/C++

Hint

Divide and Conquer

How about borrowing the idea of merge sort?



Divide:

- 2 halves

Conquer:

- # of ranges within the left half that meet the condition
- # of ranges within the right half that meet the condition

Combine:

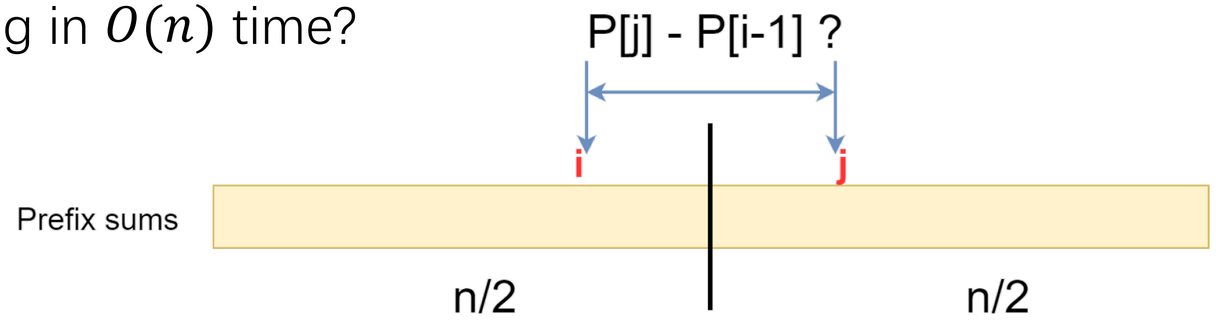
- And the ranges that cross the middle line?

Hint

Combine

We want to achieve $T(n) = O(n \log n)$. $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

How to do the combining in $O(n)$ time?



Check every pair of (i, j): unfortunately it is $\left(\frac{n}{2}\right) * \left(\frac{n}{2}\right) = O(n^2)$ ⋯

⋯ if we make no assumption on the array P that you have at this stage.

How about borrowing the idea of **merge sort**?