Divide and Conquer Sample Problem

1. Significant Inversions

Let A be an array. We say a pair of indices (i, j) is a **significant inversion** if $i < j$ and $A[i] > 2A[j]$.
Recall that a (regular) inversion is a pair with $i < j$ but $A[i] > A[j]$. For a significant inversion, it's not enough to be out-of-order, the pair also has to be very different in size. For example, in the array $[3, 1, 5, 4]$, the elements $3, 1$ (at indicies $0, 1$) form a significant inversion, as $3 > 2 \cdot 1$, but elements $5, 4$ (at indicies $2, 3$) do not form a significant inversion. Even though $5 > 4$ (i.e., they form a 'standard' inversion), it's not the case that $5 > 2 \cdot 4$, so they are not a significant inversion.

Design an algorithm to count the number of significant inversions in an array.

(a) Give pseudocode for your algorithm. The algorithm must use divide and conquer as the design strategy. Your algorithm should run in $\mathcal{O}\left(n \log^2 n\right)$ time.

Solution:

Intuition: Only need to sort the right subarray; do binary search to find location of $A[i]/2$, forms inversion with rest of array.

```
significantInversions(A):
    if A.length < 2:
        return A, 0
    else:
        mid = A.length / 2
        left, a = significantInversions(list[:mid])
        right, b = significantInversions(list[mid:])
        result, c = mergeAndCount(left, right)
        return result, (a + b + c)

mergeAndCount(left, right):
    result = []
    count = 0
    i,j = 0,0

    while(i < len(left) and j < len(right)):
        if(left[i] > 2*right[j]):
            count += len(left)-i
            j += 1
        else:
            i += 1

    i,j = 0,0
    while(i < len(left) and j < len(right)):
        if(left[i] < right[j]):
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    while(left[i:] or right[j:]):
        if(left[i:]):
            result.append(left[i])
            i += 1
```

```
            if(right[j:]):
                result.append(right[j])
                j += 1
        return result, count
```

(b) Briefly justify why your algorithm works. Specifically, you should explain why your 'combine' step counts all the 'crossing' significant inversions. **Solution:**

In the crossing case, we find all elements in the left sub array that are greater than twice the corresponding element in the right sub array. Since the right subarray is sorted, we can see exactly which elements are more than twice the value of the current element in the left subarray—it is all those to the right of where twice the element would belong in the sorted order.

(c) Give a recurrence that describes the running time. Also give a tight big-$\mathcal{O}$. You may need the version of Master Theorem on Wikipedia.
Solution:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n \log n) & \text{otherwise} \end{cases}$$

$\mathcal{O}\left(n \log^2 n\right)$