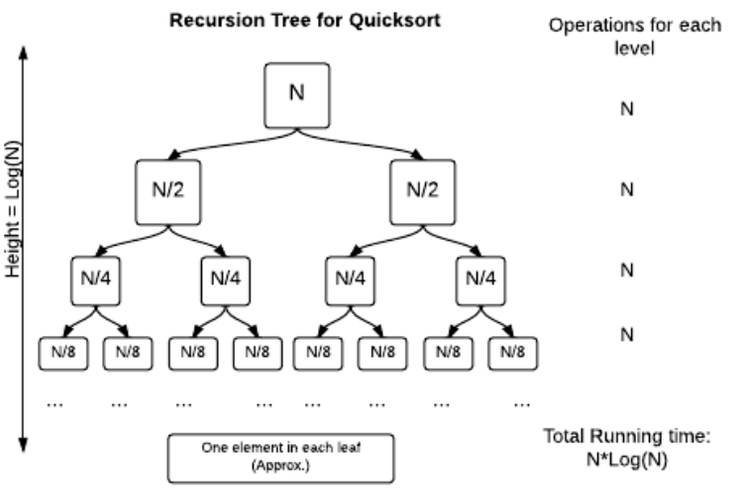
Running Time of Quicksort

Problem Statement

The running time of Quicksort will depend on how balanced the partitions are. If you are unlucky and select the greatest or the smallest element as the pivot, then each partition will separate only one element at a time, so the running time will be similar to InsertionSort.

However, Quicksort will usually pick a pivot that is mid-range, and it will partition the array into two parts. Let's assume Partition is lucky and it always picks the median element as the pivot. What will be the running time in such a case?

Running Time of Recursive Methods
Quicksort is a recursive method, so we will have to use a technique to calculate the total running time of all the method calls. We can use a version of the "Recursion Tree Method" to estimate the running time for a given array of N elements.



Each time *Partition* is called on a sub-array, each element in the sub-array needs to be compared with the pivot element. Since all the sub-arrays are passed to *partition*, there will be *N* total operations for each level of the tree.
How many levels will it take for the Quicksort to finish? Since we assume it always picks the middle element, the array will be split into 2 equal halves each time. So it will take *log(N)* splits until we get single elements in the sub-arrays. Since there are *log(N)* levels and each one involves *N* operations, the total running time for Quicksort will be *N* * *log(N)*.

In real sorting, Quicksort won't always pick the exact middle element. But as long as its regularly picking elements near the median value, it will have a running time better than Insertionsort. To make sure that Quicksort works well on most inputs, the real-world implementations do not pick the same index as pivot each time. They use some other technique, e.g., picking a random element. There are other techniques also that can be used to improve Quicksort. The Java Arrays class uses a modified version of Quicksort to sort primitives.

Notice that $O(N * log(N))$ of Quicksort is much much faster than the $O(N^2)$ of Insertion Sort. For example, for

an array of 1 million elements, $N^2 = 10^{12}$, while $N * log(N)$ is approx. 20 million, a much more manageable number.

Challenge
In practice, how much faster is Quicksort (in-place) than Insertion Sort? Compare the running time of the two algorithms by counting how many swaps or shifts each one takes to sort an array, and output the difference. You can modify your previous sorting code to keep track of the swaps. The number of swaps required by Quicksort to sort any given input have to be calculated. Keep in mind that the *last* element of a block is chosen as the pivot, and that the array is sorted in-place as demonstrated in the explanation below.

Any time a number is smaller than the partition, it should be "swapped", even if it doesn't actually move to a different location. Also ensure that you count the swap when the pivot is moved into place. The count for Insertion Sort should be the same as the previous challenge, where you just count the number of "shifts".

Note

Please use Lomuto Partition for this challenge.

Input Format
There will be two lines of input:

- *n* - the size of the array
- *ar* - *n* numbers that makes up the array

Output Format
Output one integer *D*, where *D = (insertion sort shifts) - (quicksort swaps)*

Constraints
$1 <= n <= 1000$
$-1000 <= x <= 1000$, $x \in ar$

Sample Input

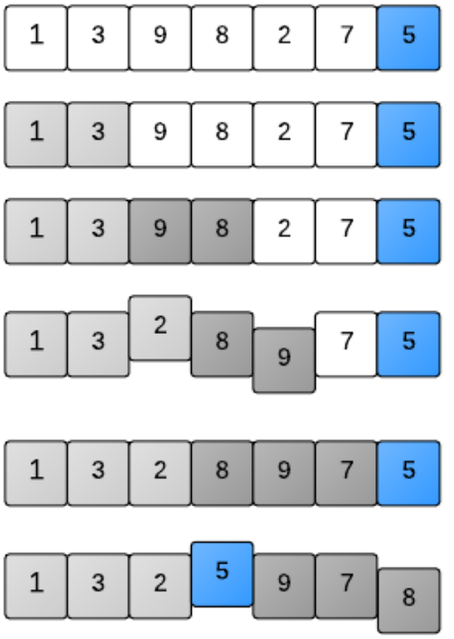
```
7
1 3 9 8 2 7 5
```

Sample Output

```
1
```

Explanation
Insertion Sort will take 9 "shifts" to sort the array. Quicksort will take 8 "swaps" to sort it, as shown in the diagram below. 9-8 = 1, the output.

Quicksort Swaps



	# of Swaps:
The first two elements are "swapped" with themself.	2
The next two element are greater than the pivot so they remain where they are.	0
The 2 is smaller than the pivot, so it is swapped.	1
The 7 is larger, so it remains where it is.	
Finally, the pivot is swapped into the correct location.	1

Quicksort continues by sorting each sub-array. This involves 2 swaps each - First to 'swap' the smaller element, and then to swap in the pivot.

Total Number of swaps for Quicksort: 4+2+2 = **8**