DIVIDE AND CONQUER

There are two problems from which you can learn the pattern of divide and conquer algorithm.

Problem A is easy

Problem B is hard (very few students successfully did it)

If you can do both, GREAT!

But for the class session, decide early whether you will go for A, B, or A+B

Obviously, a strong student is capable of finishing both.

PROBLEM A: Exponentiation

This problem is very simple. Given two positive integers a and x, compute a^x.

$$2 \leq a \leq 13$$

$$0 \leq x \leq 10^{18}$$

However, using ** operator is not allowed here. (we are coding in Python 3)

To keep size of answer convenient to read, print the output as being modulo by 2147483647.

1) Write a straightforward program that computes a^x by using for loop that repeats x times.

2) The running time of program in step 1 is O(_____).

3) Utilize the following fact, rewrite a fast version of exponentiating program.

$$a^x = \begin{cases} \sqrt{a^x} \times \sqrt{a^x} & ;x \text{ is even} \\ a \times \sqrt{a^x} \times \sqrt{a^x} & ;x \text{ is odd} \end{cases}$$

In essence,

- Divide the large problem (with respect to x) into smaller subproblems
- Conquer the subproblems
- Combine the results

NOTE Dividing can automatically result in a floating-point number. Be aware to keep x integer.

4) The running time of program in step 3 is O(_____).

PROBLEM B: Counting Inversion

Amozan is a online-sale company. In the database of Amozan, products are ranked by popularity.

Amozan tries to develop a system for recommending new product to each user. The strategy is to find a few other customers that have similar patterns of buying products, based on the popularity ranking, then, decide what to recommend based on the "next" product choice of those other customers. For example, user A has successively bought products whose ranks are as follows.

4 10 2 8

Suppose that user B is chosen to be the one with similar pattern of ranks. Let us take a look at the ranks of successive products bought by user B, as follows.

3 10 1 7 5

Following the trend of user B, Amozan will recommend product of ranks that are less than 8 but more than 4, for example, to user A.

How Amozan actually picks the product to recommend is to be developed later. The current requirement is to determine the similarity between two patterns. For example, in the case above, how may the similarity between 3 10 1 7 and 4 10 2 8 be determined?

Amozan concludes that the similarity is determined by the "number of inverted pairs of rank". A pair of rank A_i and A_j, where $i < j$ means that product i is bought before product j, is called to be inverted if and only if $A_i > A_j$.

Therefore, 3 10 1 7 contains 3 inverted pairs; (3,1), (10,1), and (10,7). And 4 10 2 8 contains 3 inverted pairs; (4,2), (10,2), and (10,8). Both patterns have the same number of inverted pairs, which is 3, and so are considered as being "very similar".

PROGRESS ACCORDING TO THE FOLLOWING PRESCRIBED STEPS. THIS IS A LEARNING CLASS, NOT A PROBLEM SOLVING CHALLENGE!

YOU ARE EXPECTED TO LEARN FROM ANSWERING EVERY QUESTION.

Given a sequence of ranks of the products that are successively bought, determine the number of inverted pairs.

1) A brute-force solution can be obtained by comparing every pair of indices (don't use *enumerate* because it's faster than nested loop only in Python, but not in fast language, like C++).
   This will result in O(___) algorithm.

   At this point, you should be able to quickly write a Python program for the brute-force solution. As a practice, spend no longer than 15 minutes to get the brute-force version work. The test cases can be downloaded from the class materials for this week.

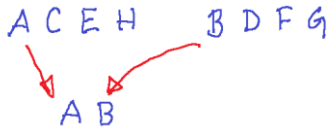
2) Review merge-sort algorithm. Then, write a merge sort program by utilizing the "merge" function given in the classroom materials for this week.

It is essential that you spend no longer than 10 minutes to complete writing a merge sort program, given the "merge" function, including testing that the program actually works perfectly.

Unless you completely understand how merging works, there is no point to move on the next step. So, study merge sort until you are certain that you understand clearly how it works.

3) How merge sort may be adapted to count the number of inversions?
(NO PROGRAMMING IN THIS QUESTION, SO DON'T JUMP INTO WRITING ONE PREMATURELY)

Consider the following merging situation. Both halves are already sorted, and are being merged.



a. Before each half was sorted, what letters were unquestionably positioned in front of B?

b. From a, at least how many inversions with B definitely existed in the original list ?

c. In the merging, B will be picked before how many letters in the first half ?

d. Given that "total inversions" is the global variable that will be the eventual answer, how many inversions can be added to it when B is picked into the merged list.

e. When C is picked, however, is there any unpicked letter in both halves that will cause inversion with C ?

f. So, does picking from the first half expose an inversion that can be added to the total amount of inversions?

g. After C is placed into merged list, D will be picked. Using the same concept as when picking B adds inversions to total amount, how many inversions with D can be added to the "total inversions"?

h. Generalize on how "total inversions" are collected to come up with the technique for counting total amount of inversions.

4) Create a global variable invCnt for counting the total number of inverted pairs. Then, modify the merge algorithm to add *a line* that collects the number of inverted pairs.

5) The resulted algorithm runs in O(_____)

Mergesort is a DIVIDE-AND-CONQUER algorithm.
Subproblems in Depth-First-Search are explored “options”.
In Divide-and-Conquer, the subproblems are explicitly specified.
Therefore, each subproblem is unique, always getting smaller, and will not be repeated.