DIVIDE AND CONQUER

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.

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

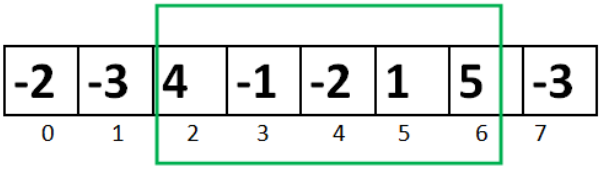
In essence,

- 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: Maximum Subsequence Sum (revisited)



4 + (-1) + (-2) + 1 + 5 = 7

Maximum Contiguous Array Sum is 7

,

INPUT

A sequence of n integers, n ≤ 100000.

OUTPUT

The sum of the maximum subsequence (the contiguous subsequence within the given sequence of numbers which has the largest sum).

Supplemental: The test-case files, archived as maxsum.zip is the same as that used in week 1

Given that maxSubSum(i,k) will give the solution for the sequence , $a_i, a_{i+1}, \ \dots \ , a_k$.

1. Split the subsequence, $a_i, a_{i+1}, \ \dots \ , a_k$ into two subsequences. In terms of correctness, the split point can be anywhere as long as it splits the sequence into two. However, the sequence should be split into halves for maximum efficiency (shortest running time). Why?

2. Write the code to perform the split as indicated in step 1, the index of the split point is all what is needed. Then add code to solve each half with a recursion.

3. Given that the answers for the front subsequence is m_a and the answer for the trailing subsequence is m_b, is it adequate to assume that the answer for the sequence is max(m_a, m_b) ?

4. Following step 3 above, is there any other subsequence that may give the larger sum than the one in step 3? What would be the essential condition of such subsequence i.e. what would be required for its beginning and ending indices?

5. The subsequence in step 4 would be one that spans across the split point. Its sum will be sum of the two maximum subsequences extending from the split point (one toward front and one toward tail).

6. Write a function to calculate the sum of the "crossing" subsequence defined in step 5.

7. Assume that the return value of the function is step 6 is m_c, what would be the sum of maximum subsequence of the original subsequence $(a_i, a_{i+1}, \ \dots \ , a_k)$?

8. Given that n is the size of a subsequence s, the running time of the function in step 6 on subsequence s would be $O($ ____ $)$

9. The total running of the maxSubSum(i, k) would be

   $T(n) = 2T(n/2) +$ the running time in step 8

   Can you resolve this recurrence equation for its upperbound? In order words, $T(n) = O(\ ?\ )$

10. Test it on the provided test cases.

11. Is this divide-and-conquer algorithm faster or slower than Kadane's 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.