CS 202

Design and Analysis of Algorithms

Assignment 1

[Question 1] Power Modulo

The RSA algorithm has been demonstrated in class, where its encryption and decryption both require the power modulo operation. Given three positive integers *m*, *k* and *n*, the power modulo operation calculates m^k modulus *n*, denoted as

$$r = m^k \bmod n \text{ , where } r \in \mathbb{Z} \text{ and } 0 \le r < n.$$

Please write a program to implement the power modulo operation. Test inputs begin with a number indicating the number of lines below. For each row below, there are three numbers: *m*, *k* and *n*. For each line, please output the result $r = m^k \bmod n$ on a single line.

Skeleton code is provided in the file **A1Q1.py**. You need to modify the function `power_modulo`. *m*, *k*, *n* are all integers between 1 and 2^{31}-1.

A testcase starts with the number of lines below, and each line contains three values *m*, *k* and *n* respectively. There will be 6 sets of test cases with 1 mark each, 2 additional marks are awarded if your code is implemented using recursion given that your code passes all 6 sets of test cases. The remaining 2 marks are awarded with the correct justification of the time complexity of your algorithm, which you should specify in the comment box when submitting on eLearn.

Sample Input

```
3
1 6 7
2 10 10
202 10000019 1000000007
```

Sample Output

```
1
4
809179673
```

[Question 2] Listing combinations at the positions powers of 10

Given a pair of integers *n* and *m*, Tom and Jerry each wrote a recursion to list all possible *m*-combinations from *n* integers {*0*, *1*, *2*, …, *n-1*} in lexicographical order, and they want to check if they generated the same list. However, the two lists are too long to compare, and they decided to compare only the 1st, the 10th, the 100th … lines, up to position the highest power of 10 in the list. As the number of combinations to be listed is significantly reduced, Tom wishes to have a more efficient algorithm to do this task, and seeks your help. Please implement an efficient algorithm for this task.

Skeleton code is provided in the file **A1Q2.py**, together with two recursion algorithms for listing all *m*-combinations in lexicographical order. You may start by investigating the two recursion algorithms, choose one to start coding your own functions, and feel free to modify the functions and parameters in the skeleton code. *m* and *n* are integers and $1 \le m \le n \le 200$.

A testcase starts with the number of lines below, and each line contains two values *n* and *m*. There will be 6 sets of test cases with 1 mark each, 2 additional marks are awarded if your code is implemented using recursion given that your code passes all 6 sets of test cases. The remaining 2 marks are awarded with the correct justification of the time complexity of your algorithm, which you should specify in the comment box when submitting on eLearn.

Sample Input

```
6
10 1
5 2
6 3
10 7
16 16
20 10
```

Sample Output

```
0
9
0, 1
3, 4
0, 1, 2
0, 4, 5
0, 1, 2, 3, 4, 5, 6
0, 1, 2, 3, 4, 8, 9
1, 2, 4, 5, 6, 7, 8
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
0, 1, 2, 3, 4, 5, 6, 7, 8, 18
0, 1, 2, 3, 4, 5, 6, 8, 12, 19
0, 1, 2, 3, 4, 5, 15, 17, 18, 19
```

0, 1, 2, 4, 5, 13, 15, 16, 17, 18
1, 2, 3, 5, 9, 10, 12, 14, 16, 17

[Question 3] Asymptotic Analysis

Using any of the techniques for solving recurrence in class, provide an asymptotic bound for *T(n)* for each of the following recurrences:

i. $T(n) = T\left(\frac{n}{3}\right) + 2 \cdot T\left(\frac{2n}{3}\right) + \log n$

ii. $T(n) = 3T\left(\frac{3n}{4}\right) + \log n$

Please write your answer in a word document A1Q3.docx, and keep your answer within 1 page for this question.