Gems from the world of data structures and algorithms

Bit-Sum Prime

Difficulty level: *moderate*

Every student, who has learned programming, must have written a program to determine whether a given positive integer is a prime number. Basically in order to determine whether a positive integer n is prime, we search for any number in the range $[2, n-1]$ which can divide n. Some of you would have designed a slighly better implementation where you search for any divisor of n from the range $[2, \lfloor\sqrt{n}\rfloor]$. Does either of these two implementations correspond to an efficient algorithm ? Ponder over this question deeply.

Let us try to design an efficient algorithm for a problem which looks as simple and innocent as the problem discussed above. First, we give a definition.

Definition 1. *A positive integer is said to be bit-sum prime if the sum of the bits in its binary representation is a prime number. For example 6(110), 14(1110) are bit-sum prime numbers, whereas 29(11101) is* **not** *a bit-sum prime number.*

Design and implement an algorithm which receives a 64-bit integer n and outputs the count of all the bit-sum prime numbers less than n. Test it for a really large number, for example, execute your algorithm for 123456789123456789.

Hint: Your algorithm/program is **NOT** supposed to enumerate all the bit-sum prime numbers. Instead, it has to just report the count of all the bit-sum prime number. Hence the output will be just a single number. Notice that you will have to use clever programming skills also in this problem.

every art is beautiful and so is the art of algorithm design ...