**All-Go-Rhythm**

So what is an algorithm? How do you go about writing algorithms? And if given more than one

algorithm how do you find which one is the best? Several books have been written on

algorithms, which tell you about lots and lots of algorithms, and how you'd implement them,

then why would you want to refer to this section?

An **algorithm** is a step-by-step procedure for calculations. In this section we'd be regularly

analysing typical problems, and the ways we generally approach them. Following that would be

one of the best algorithms suited for that problem. We hope this section inculcates in you the

ability to think over about the best algorithm given a problem.

**Today's problem: Generate all prime numbers between 1 to n.**

**Our usual algorithm:**

1. take x=2

2. do

3. take p = 2

4. do

5. check whether x is divisible by p.

6. If yes then break out of the loop.

7. Else increase p.

8. Check if p=x/2.

9. If yes break out of the loop.

10. Else go to 4

11. check if p=x/2+1

12. if yes then number is prime

13. increase x

14. check if x=n

15. if yes, break out of the loop.

Although there is no problem with this algorithm, this loop checks each number with all

numbers less than half of it, and goes on for each and every number from 1 to n. This is

obviously not what we need. It would be best if we follow an elimination method to remove all

numbers which are not prime within a given range.

{Set of prime nos.} = {all numbers} – {set of numbers which are not prime}

**Sieve of Eratosthenes**

The sieve of Eratosthenes, one of a number of prime number sieves, is a simple, ancient

algorithm for finding all prime numbers up to any given limit. It does so by iteratively marking

as composite the multiples of each prime, starting with the multiples of 2. Let us see the

algorithm for doing the same task.

To find all the prime numbers less than or equal to a given integer *n* by Eratosthenes' method:

1. Create a list of consecutive integers from 2 to *n*: (2, 3, 4, ..., *n*).

2. Initially, let *p* equal 2, the first prime number.

3. Starting from *p*, count up in increments of *p* and mark each of these numbers greater

than *p* itself in the list. These numbers will be 2*p*, 3*p*, 4*p*, etc.; note that some of them

may have already been marked.

4. Find the first number greater than *p* in the list that is not marked. If there was no such

number, stop. Otherwise, let *p* now equal this number (which is the next prime), and

repeat from step 3.

When the algorithm terminates, all the numbers in the list that are not marked are

prime.

Take out a sheet and implement this algorithm for numbers 2 to 50. Does this work faster? Yes?

Well good. This algorithm has a complexity of O(n log log n) which is very small when

compared to the earlier algorithm which we implemented. (Try calculating its complexity).

Now let us go through this algorithm, and see one of its shortcomings. If you notice, the

number 12 will be marked for 2,3,4 and 6. Since it is sufficient if we mark it once, we are

obviously wasting time here. **As a refinement, it is sufficient to mark the numbers in**

**step 3 starting from *p*2, as all the smaller multiples of *p* will have already been**

**marked at that point. This means that the algorithm is allowed to terminate in step**

**4 when *p*2 is greater than *n*.**

We are through, and now most of us know about a new algorithm, which will run at a faster

speed. But this is not where you stop. What you must do next is google it up, find what more

modifications other scientists have done, try developing a code for this, and send it to us. The

most efficient ones get their name featured in this column next month! So hurry!

**Where you mail:** team@techahoy.in

**What you mail:** Your code in any language, along with your name, branch, year and college/

name, class and school.

See you soon!

Anindya Dutta,

Head of Software Development,

Tech Ahoy