WEEK 1, LECTURE 1 ON 18 AUGUST 2021 CS1.301.M21 ALGORITHM ANALYSIS AND DESIGN

COMPUTATION PROBLEMS

Main Questions:

- What is a *computational problem*?
- How to prove correctness? Is it efficient?
- How to prove optimality?
- Can we do better?

Regarding problems: The way we define a solution makes problems different according to constraints.

Sorting appears to be a computational problem. For it to be so we need a computational solution.

A mythological sorting algorithm would have steps like: Sit under a banyan tree, meditate, ask god for output etc. This is not a computational solution to problems.

Posing problems follows these questions:

- When is it a computational problem?
- Solutions should have a set of computational constraints that define a problem.

We could give a solution and a problem can be whatever problem this solution solves. This is a catch 22 but is valid although we avoid this method of posing problems.

Pertinent question: Is there a way to pose problems that are not dependent on the solution of that problem?

-Eg. Is input equal to 2? Is the input the smallest positive even number? These are all different ways of posing the same problem. Do you want to count these as different problems? This depends on the context. But when counting problems we don't count these as different problems.

Problems as membership queries in *languages*

Supposing digitalized input (0s and 1s) and supposing a noisy environments since there are really no noiseless environments.

If the output has n bits then problems can be posed as n decision problems.

Therefore you can pose problems Problems as membership queries in languages. So if X is the input and when looking for an output we can ask whether X belongs to all the set of all inputs that give the output 1.

So the problem is a subset of all inputs that give a true answer (1). And we say that **the subset of strings is a language**.

`{0,1}*` is the set of all finite length binary strings. So every problem is a language which is a subset of `{0,1}*`.

Eg. is n (an integer) a prime number. It is the same as asking whether n belongs to the set of all primes i.e. a membership query in a language.

So a problem is a membership query in a language.

Now we can pose a problem without solving it. And also removes ambiguity when posing problems like this since it is just a membership query in the set `{2}`.

What is meant by a "solution"?

There are, in fact, problems that do not have algorithms that solve them. And also solutions that exist but cannot be executed without infinite resources.

Axioms of Computation

- *It takes non-zero time to retrieve data from far off locations*: Information travels at a finite speed as of today's technology. Basically, Random access memory is possible but only in small sizes i.e. Machines are not Omnipresent.
- *Only finite information can be stored/retrieved from finite volume*: infinite information cannot remember everything this is only possible only in a noiseless environment. i.e. Machines are not omniscient.
- *A finite length code only exerts finite amount of control*: Independent of the programming language used, a finite number of instructions can only exert control over a finite number of things i.e. Machines are not Omnipotent

How do we compare solutions?

We can use an asymptotic analysis but why is that a valid way of evaluating and algorithm? Since input size and execution conditions can vary.

"Betterness" depends on what resources are valuable to us like space and time. But time is not reusable, so it is the most valuable.

How to learn algorithms (or anything)?

- *No way communication*: Self discovering knowledge by thinking about what you know.
- *One way communication*: Reading, watching etc to gain knowledge.
- *Two way communication*: Talking, conversing and listening to come up with new knowledge.