CS512 LECTURE NOTES - LECTURE 1

1 Algorithms

An **Algorithm** is:

- sequence of steps
- is not ambiguous
- process to solve a problem
- takes an input
- produces an output
- Terminates in a finite amount of time

Examples:

1. **Sorting Algorithm**
   Input: finite array of numbers
   Output: sorted finite array of numbers
2. **Shortest path**
   Input: Graph (V,E), v1,v2
   Output: v1=x1,x2,x3,x4,...,xn=v2 (shortest path)
3. **Prime**
   Input: $n \in \mathbb{N}$
   Output: True or False (prime or not prime)

Problems for which the output is either True or False are called decision problems. (Recall DFA, PDA, Turing Machines from CS205, they either accept or reject).

Why "Computer" algorithms?

Algorithms have existed since ancient times. Remember Euclid's GCD algorithm, generation of all primes less than a fixed given integer n, etc.

However, due to the speed with which a computer can execute the algorithms, most algorithms are intended to be used by a computer.

Analysis of Algorithms

There are three aspect that are important to analyze:

- Correctness: Formal proof that the algorithm halts and produces the correct output given an input.
- Time complexity:
  - Measures the time it takes to solve a given problem.
  - Is used to compare one algorithm to another one.
  - Is used to determine if we have the best possible algorithm for a problem (optimal) or if there is room for improvement.
- Space complexity: Measures the amount of memory used by an algorithm.

1.1 Measuring execution time

- Not useful to measure the actual time (in seconds) since we might be using different computers with different speeds.
- The basic idea is to count operations
- The running time is:
  1. Proportional to the number of operations
  2. A function of the length of the input

NOTICE: The actual running time is a function of the input, not only of the input length. In order to simplify our analysis we use three possible cases:

- **Best case:** Among all possible input instances of the problem (P_n) with length n, we compute the minimum execution time:

  $$B(n) = min\{time(I)|I \in P_n\}$$

- **Average case:** Among all possible input instances of the problem (P_n) with length n, we compute the average execution time:

  $$A(n) = \sum_{i \in P_n} \{probability(I)time(I)\}$$

- **Worst case:** Among all possible input instances of the problem (P_n) with length n, we compute the maximum execution time:

  $$W(n) = max\{time(I)|I \in P_n\}$$

1.2 Big Oh and Big Omega

The functions that we are interested in satisfy:

1. Defined over the set of natrual numbers
2. Are positive non-decreasing

Given a function $g : \mathbb{N} \to \mathbb{R}^+$ we can classify other functions according to the following definitions:

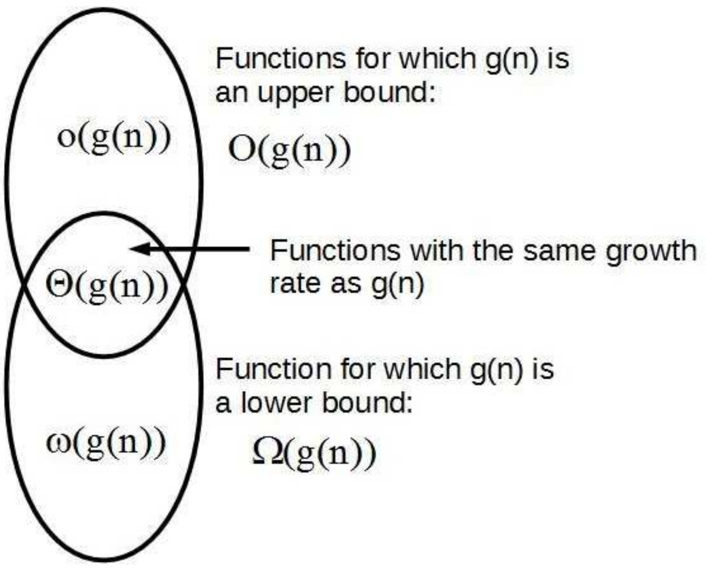
- Functions $f(n)$ for which $g(n)$ is an upper bound (i.e. functions that grow no faster than $g(n)$)

$$O(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^+ | f(n) \leq cg(n) \text{ for some } c > 0\}$$

- Functions $f(n)$ for which $g(n)$ is a lower bound (i.e. functions that grow at least as fast as $g(n)$)

$$\Omega(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^+ | f(n) \geq cg(n) \text{ for some } c > 0\}$$

1.3 Little o and Little ω


Functions for which g(n) is an upper bound:
O(g(n))
o(g(n))
Θ(g(n))
Functions with the same growth rate as g(n)
Function for which g(n) is a lower bound:
Ω(g(n))
ω(g(n))

Notice that

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

So we can define

$$\Theta(g(n)) = \{f : \mathbb{N} \to \mathbb{R}^+ | c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for some } c_1 > 0,\ c_2 > 0\}$$

where $f(n)$ is "sandwiched" between $c_1 g(n)$ and $c_2 g(n)$

In order to show that we can use these sets to classify functions according to their rate of growth, we will show that the relation defined by Θ is an equivalence relation, i.e. it is reflexive, symmetric and transitive.

1.4 Θ is an equivalence relation

Reflexive:
Clearly $f(n)$ satisfies $c_1 f(n) \leq f(n) \leq c_2 f(n)$, therefore $f(n) \in \Theta(f(n))$.

Symmetric:
Let $f(n) \in \Theta(g(n)) \Rightarrow c_1 g(n) \leq f(n) \leq c_2 g(n)$.

From the first inequality we have that:

$$g(n) \leq \frac{1}{c_1} f(n)$$

And from the second inequality we have that:

$$\frac{1}{c_2} f(n) \leq g(n)$$

Let $c'_1 = \frac{1}{c_2}$, and $c'_2 = \frac{1}{c_1}$, we have that:

$$c'_1 f(n) \leq g(n) \leq c'_2 f(n)$$

Transitive:

Assume that $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n))$. Using the definition of Θ we have that:

$$\begin{array}{ccccc} c_1 g(n) & \leq & f(n) & \leq & c_2 g(n) \\ c_3 h(n) & \leq & g(n) & \leq & c_4 h(n) \end{array}$$

From the first inequality we have:

$$g(n) \leq \frac{1}{c_1} f(n)$$

Combining it with the third inequality we have:

$$c_3 h(n) \leq g(n) \leq \frac{1}{c_1} f(n) \qquad \textbf{(5)}$$

From the second inequality we have:

$$\frac{1}{c_2} f(n) \leq g(n)$$

Combining it with the fourth inequality we have:

$$\frac{1}{c_2} f(n) \leq g(n) \leq c_4 h(n) \qquad \textbf{(6)}$$

If we let $c_1' = c_1 c_3$, $c_2' = c_2 c_4$, from **(5)** and **(6)** we get:

$$c_1' h(n) \leq f(n) \leq c_2' h(n)$$

Therefore, $f(n) \in \Theta(h(n))$

So we have that we can use $\Theta(g(n)$ to classify functions according to their asymptotic growth rate into equivalence classes.

1.5 Using limits

Since we are interested in the asymptotic behavior of the functions, it is sometimes easier to use limits to determine their growth rate. It is easy to see that:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} c & \Rightarrow & f(n) \in \Theta(g(n)) \\ 0 & \Rightarrow & f(n) \in o(g(n)) \\ \infty & \Rightarrow & f(n) \in \omega(g(n)) \end{cases}$$