Design and Analysis of Algorithms

2018A7PS0193P

January 22, 2021

1 Fundamentals

Definition 1.1. *An algorithm is a well defined computational procedure. It takes an input, does some computation and terminates with output*

To check the correctness of an algorithm, we must check the following characteristics:

- Initialization: The algorithm is correct at the beginning
- Maintenance : The algorithm remains correct as it runs
- Termination : The algorithm terminates in finite time, correctly

For this entire course, we must always prove these characteristics when defining any algorithm.

Algorithms are generally defined by a complexity - the time taken to complete the computation on a given input size. There are three ways we could consider this - best case, worst case, or average case.

Complexity is discussed a lot in DSA, so I'm not going to rewrite it here. A quick roundup is:

- $O(g(n)) = \{f(n) : \text{there exists } c, n_0 : 0 \leq f(n) \leq c \cdot g(n) \forall n \geq n_0\}$
- $\Omega(g(n)) = \{f(n) : \text{there exists } c, n_0 : 0 \leq c \cdot g(n) \leq f(n) \forall n \geq n_0\}$
- $\Theta(g(n)) = \{f(n) : \text{there exists } c, n_0 : 0 \leq c_2 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \forall n \geq n_0\}$

To find complexities in the case of recurrences, we use the **master method**. Let the recurrence be given by:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Here, $a, b \geq 1$. Let ϵ be a constant. Then:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ then $T(n) = \Theta(n^{\log_b a})$
2. If $f(n) = O(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$
3. If $f(n) = O(n^{\log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$ provided if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Here, we redo DSA despite it being a prerequisite of the course. This recap has lasted 3 lectures (so far). You should probably just read CLRS, this is a waste. The topics covered are:

- Quicksort (and it's average case analysis)
- The $\Omega(n \log n)$ lower bound of comparison sorting
- Non-comparison sorting like counting sort, radix sort, etc.
- Average case analysis of bucket sort