Asymptotic Bounds and Runtime Analysis

This document contains summary of some of the items discussed as part of the COMS 3110 Lectures. This document **does not replace the lecture materials**. This document may contain some topics that are not covered as part of the lecture; you will not be tested on those parts, they are made available to you for gaining further knowledge on topics/concepts that are related to class-lecture.

1 Topics Covered

- Runtime analysis
- Asymptotic bounds (Big-O)

2 Runtime Analysis

To analyze the runtime of an algorithm in a way that is independent of any specific programming language or execution environment, we use a hypothetical machine/computation model. A widely adopted choice is the random access machine model (RAM model). The RAM model is both simple and sufficiently accurate for theoretical analysis. It is based on the idea of breaking down computations into atomic steps (i.e., fundamental operations such as simple variable assignments, condition checks, boolean evaluations, etc.) We make the following assumptions in this model.

1. Each atomic step can be performed in "constant" time.
2. All atomic steps take the "same" amount of constant time.

Under these assumptions, the runtime of an algorithm is expressed as a function

$$T : \mathbb{N} \to \mathbb{N},$$

where the input is the size of the problem (e.g., the number of elements in the input array). This function $T(n)$ captures the rate at which the time taken to complete the algorithm grows as the input size n increases.

Several types of runtime analysis are commonly used.

1. Worst-case runtime: The maximum time the algorithm can take on any input of a given size. This provides a guarantee that the algorithm will never take longer than this bound, regardless of the input.
2. Average-case runtime: The average time taken over all possible inputs of a given size, assuming some probability distribution over the inputs. This provides insight into the typical performance of the algorithm.
3. Expected runtime: The average time over all possible random choices made by the algorithm itself. This is relevant for randomized algorithms, where performance depends on the algorithm's internal randomness rather than just the input.
4. Amortized runtime: The average time per operation over a sequence of operations. This is useful when some individual operations are expensive, but their cost can be spread out so that the overall average remains low.

3 Asymptotic Bounds

Definition 1 (Dominance of function)**.** *A function $f : \mathbb{N} \to \mathbb{N}$ is dominated by a function $g : \mathbb{N} \to \mathbb{N}$ iff*

$$\exists c > 0, \exists n_0 > 0, \forall n \geq n_0 . f(n) \leq c.g(n).$$

Dually: A function $f : \mathbb{N} \to \mathbb{N}$ dominates a function $g : \mathbb{N} \to \mathbb{N}$ iff

$$\exists c > 0, \exists n_0 > 0, \forall n \geq n_0 . f(n) \geq c.g(n).$$

c: scaling factor.

1. $f(n) \in O(g(n))$ iff f is dominated by g. The valuation of the function f is no worse than the valuation of scaled ($c\times$) valuation of function g for all $n \geq n_0$. Intuition: function f grows at most as fast as function g.

   The following concepts of function-domination relations is not covered in class.

2. $f(n) \in \Omega(g(n))$ iff f dominates g. Intuition: function f grows at least as fast as function g.
3. $f(n) \in \Theta(g(n))$ iff f dominates and is dominated by g. Intuition: function f and g grows at the same rate.
4. $f(n) \in o(g(n))$ iff f is dominated by g for any scaling factor c. Intuition: f grows slower than g.
5. $f(n) \in \omega(g(n))$ iff f dominates g for any scaling factor c. Intuition: f grows faster than g.

Some noteworthy conclusions: $f(n) \in o(g(n)) \Rightarrow f(n) \in O(g(n))$, and $f(n) \in \omega(g(n)) \Rightarrow f(n) \in \Omega(g(n))$. $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$ iff $f(n) \in \Theta(g(n))$.

Relationship with Limits. The limiting values for the functions can be used to infer the dominance and, in turn, their relationship in terms of $O, \Omega, \Theta, o, \omega$. For instance, the $\lim_{n\to\infty} f(n)/g(n)$ captures the ratio of the functions as n tends to infinity.

1. if the above limiting value is not infinity, then $f(n)$ grows at most as fast as $g(n)$, i.e., $f(n) \in O(g(n))$.
2. if the above limiting value is not 0, then then $f(n)$ grows at least as fast as $g(n)$, i.e., $f(n) \in \Omega(g(n))$.
3. if the above limiting value is neither 0 nor ∞, then $f(n) \in \Theta(g(n))$.
4. Think about the limiting values that would imply the relationship between f and g is either o or ω.

The following lemma formalizes item 1 above.

Lemma 1. *Suppose there exists a constant $L \geq 0$ such that*

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = L.$$

Then, $f(n) \in O(g(n))$.

Proof. By definition of limit at infinity, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = L$ implies that for all $\epsilon > 0$, there exists $N > 0$ such that

$$\left| \frac{f(n)}{g(n)} - L \right| < \epsilon,$$

for all $n \geq N$.

Pick $n_0 = N$ and $c > L$. Let $\epsilon = c - L > 0$. Then, we can conclude that for all $n \geq n_0$,

$$\begin{aligned} \left|\frac{f(n)}{g(n)} - L\right| &< c - L \\ \frac{f(n)}{g(n)} - L &< c - L \\ f(n) &< cg(n) \end{aligned}$$

Thus, by definition, $f \in O(g(n))$. □

One of the rules that becomes very useful when dealing with limits of ratio of functions is the rule due to L'Hospital (French Mathematician). It states that if $\lim_{n\to\infty} f(n) = \infty$ and $\lim_{n\to\infty} g(n) = \infty$, then $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} \frac{\frac{d}{dn}f(n)}{\frac{d}{dn}g(n)}$. Combining this result with Lemma 1, we obtain the following lemma.

Lemma 2. *Suppose* $\lim_{n\to\infty} f(n) = \infty$, $\lim_{n\to\infty} g(n) = \infty$, *and*

$$\lim_{n\to\infty} \frac{\frac{d}{dn}f(n)}{\frac{d}{dn}g(n)} = 0.$$

Then, $f(n) \in O(g(n))$.

4 Exercise

1.
```
for (i=1 to n)
    for (j=i+1 to n)
      <do some atomic operations>
```

2.
```
for (i=1 to n)
    for (j=1 to i^2)
      <do some atomic operations>
```

3.
```
for (i=1; i<n; i=i*2)
    <do some atomic operations>
```

4.
```
for (i=n; i>1; i=i/2)
    <do some atomic operations>
```

5.
```
for (i=1 to n)
    for (j=1; j<i; j=j*2)
      <do some atomic operations>
```