Data Structures & Algorithms (COMP2113)
Lecture # 12
Algorithm Analysis | Part03
Asymptotic Analysis

Course Instructor

Dr. Aftab Akram

PhD CS

Assistant Professor

Department of Information Sciences, Division of Science & Technology

University of Education, Lahore

Asymptotic Analysis

- The study of an algorithm as the input size “gets big” or reaches a limit (in the calculus sense).
- Asymptotic analysis is a form of “back of the envelope” estimation for algorithm resource consumption.
- It provides a simplified model of the running time or other resource needs of an algorithm.
- This simplification usually helps you understand the behavior of your algorithms.
- Generally, problems with smaller input sizes consider constants. However, for larger problems the constants does not matter.

Upper Bounds

- It indicates the upper or highest growth rate that the algorithm can have.
- Big-Oh Notation used for this purpose.
- If the upper bound for an algorithm's growth rate (for, say, the worst case) is $f(n)$, then we would write that this algorithm is *"**in the set $O(f(n))$ in the worst case**"* (or just ***"in $O(f(n))$ in the worst case")***.
- For example, if n^2grows as fast as $T(n)$ (the running time of our algorithm) for the worst-case input, we would say the algorithm is "in $O(n^2)$in the worst case."

Upper Bounds

- $T(n)$ represents the true running time of the algorithm.
- $f(n)$ is some expression for the upper bound.
- For $T(n)$ a non-negative valued function, $T(n)$ is in set $O(f(n))$ if there exist two positive constants c and n_0 such that $T(n) \leq cf(n)$
- The definition says that for all inputs of the type in question (such as the worst case for all inputs of size n) that are large enough (i.e.,$n > n_0$), the algorithm always executes in less than $cf(n)$ steps for some constant c.
- Constant $n_0$is the smallest value of n for which the claim of an upper bound holds true. Usually, n_0 is 1 (but not always)

Example 3.4 Consider the sequential search algorithm for finding a specified value in an array of integers. If visiting and examining one value in the array requires c_s steps where c_s is a positive number, and if the value we search for has equal probability of appearing in any position in the array, then in the average case $\mathbf{T}(n) = c_s n/2$. For all values of $n > 1$, $c_s n/2 \le c_s n$. Therefore, by the definition, $\mathbf{T}(n)$ is in $\mathrm{O}(n)$ for $n_0 = 1$ and $c = c_s$.

Example 3.5 For a particular algorithm, $\mathbf{T}(n) = c_1 n^2 + c_2 n$ in the average case where c_1 and c_2 are positive numbers. Then, $c_1 n^2 + c_2 n \le c_1 n^2 + c_2 n^2 \le (c_1 + c_2)n^2$ for all $n > 1$. So, $\mathbf{T}(n) \le cn^2$ for $c = c_1 + c_2$, and $n_0 = 1$. Therefore, $\mathbf{T}(n)$ is in $\mathrm{O}(n^2)$ by the second definition.

Example 3.6 Assigning the value from the first position of an array to a variable takes constant time regardless of the size of the array. Thus, $\mathbf{T}(n) = c$ (for the best, worst, and average cases). We could say in this case that $\mathbf{T}(n)$ is in $\mathrm{O}(c)$. However, it is traditional to say that an algorithm whose running time has a constant upper bound is in $\mathrm{O}(1)$.

Upper Bounds

- Some algorithms have the same behavior no matter which input instance they receive, e.g., searching maximum in an array.
- But for many algorithms, it makes a big difference, e.g., searching an unsorted array for a particular value.
- So any statement about the upper bound of an algorithm must be in the context of some class of inputs of size n.
- We measure this upper bound nearly always on the best-case, average-case, or worst-case inputs.
- We always seek to define the running time of an algorithm with the tightest (lowest) possible upper bound.

Lower Bounds

- Big-Oh notation describes an upper bound.
- In other words, big-Oh notation states a claim about the greatest amount of some resource (usually time) that is required by an algorithm for some class of inputs of size n
- Similar Ω notation is used to describe the least amount of a resource that an algorithm needs for some class of input.
- The lower bound for an algorithm (or a problem, as explained later) is denoted by the symbol Ω, pronounced “big-Omega” or just “Omega.”

Lower Bounds

- The following definition Ω for is symmetric with the definition of big-Oh.
- For $T(n)$ a non-negatively valued function, $T(n)$ is in set $\Omega(g(n))$ if there exist two positive constants c and n_0 such that $T(n) \geq cg(n)$ for all $n \geq n_0$.

Example 3.7 Assume $\mathbf{T}(n) = c_1 n^2 + c_2 n$ for c_1 and $c_2 > 0$. Then,

$$c_1 n^2 + c_2 n \geq c_1 n^2$$

for all $n > 1$. So, $\mathbf{T}(n) \geq cn^2$ for $c = c_1$ and $n_0 = 1$. Therefore, $\mathbf{T}(n)$ is in $\Omega(n^2)$ by the definition.

Θ Notation

- The definitions for big-Oh and Ω give us ways to describe the upper bound for an algorithm
  - if we can find an equation for the maximum cost of a particular class of inputs of size n
- the lower bound for an algorithm
  - if we can find an equation for the minimum cost for a particular class of inputs of size n
- When the upper and lower bounds are the same within a constant factor, we indicate this by using Θ (Big-Theta) notation.
- An algorithm is said to be $\Theta(h(n))$ if it is in $O(h(n))$ and it is in $\Omega(h(n))$.
- Note that we drop the word "in" for Θ notation, because there is a strict equality for two equations with the same Θ.
- In other words, if $f(n)$ is $\Theta(g(n))$, then $g(n)$ is $\Theta(f(n))$.

Θ Notation

- The sequential search algorithm is both in $O(n)$ and in $\Omega(n)$ in the average case, we say it is $\Theta(n)$ in the average case.

Next Lecture

- In next lecture, we will discuss simplifying rules and determining running time of an algorithm.