CS207 Design and Analysis of Algorithms

Sajith Gopalan

Indian Institute of Technology Guwahati

sajith@iitg.ac.in

January 10, 2022

Aymptotics

- Time and space complexity of algorithms are represented using the following asymptotic notations
- O, o, Ω, ω, Θ
- these correspond respectively to asymptotic $\leq$, $<$, $\geq$, $>$, $=$

O-notation

- $O(g(n)) = \{f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0, f(n) \leq c.g(n)\}$
- To show the membership of $f(n)$ in $O(g(n))$ we write $f(n) = O(g(n))$
- in other words, $f(n) = O(g(n))$ iff while moving right along the numberline, sooner or later we come to a number n_0 such that $f(n)$ will be dominated by a scaled version of $g(n)$ for all n furtheron

o-notation

- $o(g(n)) = \{f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0, f(n) < c.g(n)\}$
- To show the membership of $f(n)$ in $o(g(n))$ we write $f(n) = o(g(n))$
- in other words, $f(n) = o(g(n))$ iff while moving right along the numberline, sooner or later we come to a number n_0 such that $f(n)$ will be *strictly* dominated by a scaled version of $g(n)$ for all n furtheron

Ω-notation

- $\Omega(g(n)) = \{f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0, f(n) \geq c.g(n)\}$
- To show the membership of $f(n)$ in $\Omega(g(n))$ we write $f(n) = \Omega(g(n))$
- in other words, $f(n) = \Omega(g(n))$ iff while moving right along the numberline, sooner or later we come to a number n_0 such that $f(n)$ will dominate a scaled version of $g(n)$ for all n furtheron

ω-notation

- $\omega(g(n)) = \{f(n) \mid \exists c > 0, \exists n_0 > 0, \forall n \geq n_0, f(n) > c.g(n)\}$
- To show the membership of $f(n)$ in $\omega(g(n))$ we write $f(n) = \omega(g(n))$
- in other words, $f(n) = \omega(g(n))$ iff while moving right along the numberline, sooner or later we come to a number n_0 such that $f(n)$ will *strictly* dominate a scaled version of $g(n)$ for all n furtheron

Θ-notation

- $\Theta(g(n)) = \{f(n) \mid \exists c > 0, \exists d > 0, \exists n_0 > 0, \forall n \geq n_0, c.g(n) \leq f(n) \leq d.g(n)\}$
- To show the membership of $f(n)$ in $\Theta(g(n))$ we write $f(n) = \Theta(g(n))$
- in other words, $f(n) = \Theta(g(n))$ iff while moving right along the numberline, sooner or later we come to a number n_0 such that $f(n)$ will be sandwiched by two scaled versions of $g(n)$ for all n furtheron

Example

- $(\log n)^{\log \log n}$ vs $2^{\sqrt{\log n}}$
- Take logarithm on both sides: $(\log \log n)^2$ vs $\sqrt{\log n}$
- Take logarithm on both sides again: $2 \log \log \log n$ vs $\frac{1}{2} \log \log n$
- Substitute $x = \log \log \log n$: $2x$ vs $\frac{1}{2} 2^x$ (Linear vs Exponential)
- So $(\log n)^{\log \log n} = O(2^{\sqrt{\log n}})$

Time complexity

- Denote by $T_A(w)$ the time that algorithm A takes on input w
- This is the number of instructions that A executes on input w before coming to halt
- $T : \mathbb{N} \to \mathbb{N}$ is the worst case time complexity of A if $T(n) = \max_{w \text{ of length } n} T_A(w)$
- $T : \mathbb{N} \to \mathbb{N}$ is the average case time complexity of A if $T(n) = \text{ave}_{w \text{ of length } n} T_A(w)$
- the algorithm with the best worst-case complexity does not necessarily have the best average complexity and vice-versa

Space complexity

- Denote by $S_A(w)$ the space that algorithm A takes on input w
- This is the number of memory registers that A uses on input w before coming to halt
- $S : \mathbb{N} \to \mathbb{N}$ is the worst case space complexity of A if $S(n) = \max_{w \text{ of length } n} S_A(w)$
- $S : \mathbb{N} \to \mathbb{N}$ is the average case space complexity of A if $S(n) = \text{ave}_{w \text{ of length } n} S_A(w)$
- the algorithm with the best worst-case complexity does not necessarily have the best average complexity and vice-versa