# Asymptotic Analysis

- Runtime Analysis and Big Oh - $O(\cdot)$
- Complexity Classes and Curse of Exponential Time
- $\Omega(\cdot)$, $\Theta(\cdot)$, $o(\cdot)$, $\omega(\cdot)$ - Relational properties

Imdad ullah Khan

# Comparing Algorithms

Given two algorithms for the same problem which one is "better"?

Which one has smaller runtime?

We can compare integers $37 < 49$ ?

We can compare real numbers $37.05 < 22.49$ ?

We can compare signed real numbers $-37.5 < -22.4$ ?

Can we compare arrays? $\boxed{36 \mid 9 \mid 22} < \boxed{8 \mid 35 \mid 24}$ ?

Can we compare functions? $n^2 + 2n \;<\; 3n^2$

# Function: List Representation

Let $X$ and $Y$ be two sets. A function $f$ maps **each** element of $X$ to **exactly one** element of $Y$

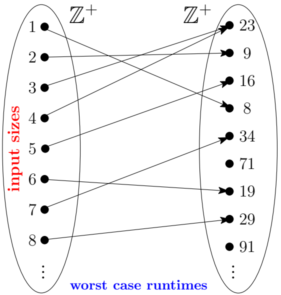
$f : X \mapsto Y$

- $X$ is the domain of $f$
- $Y$ is the codomain of $f$

$f(x) = y$

- $y$ is the image of $x$
- $x$ is the pre-image of $y$

Worstcase runtime of algorithms characterized as functions of input sizes




**input sizes**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ⋯ |
|---|---|---|---|---|---|---|---|---|
| 8 | 9 | 23 | 23 | 16 | 19 | 34 | 29 | ⋯ |

**worst case runtimes**

# Asymptotic Notation

- We use asymptotic analysis of functions to analyze algorithms running time
- Characterize running time for all inputs instances of a certain size (so worst-case) with just one runtime function
- Small inputs are not much of a problem, we want to learn behavior of an algorithm on large inputs

# Asymptotic Notation

Our foremost goals in analysis of algorithms are to

**Goal 1:** Determine the runtime of an algorithm on inputs of **large size**

**Goal 2:** Determine how the runtime **grows with increasing inputs**

$\triangleright$ How the runtime changes when input size is doubled/tripled?

# Asymptotic Analysis: Big Oh

**Definition (Big Oh)**

A function $g(n) \in O\big(f(n)\big)$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that

$$\forall n \geq n_0 \qquad g(n) \leq c \cdot f(n)$$

- $O(f(n))$ is a set of functions
  - ▷ We abuse the notation and say $g(n) = O(f(n))$ for $g(n) \in O(f(n))$
- A notion of $a \leq b$ for functions as for real numbers
- $f(n)$ is an asymptotic upper bound on $g(n)$

**Provides the right framework for both our goals**

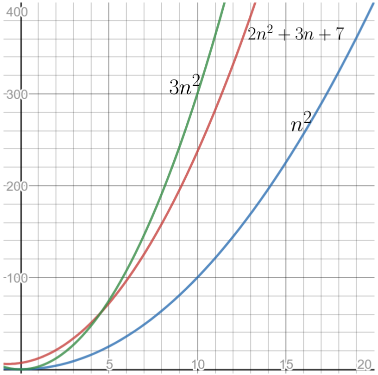
# Big Oh

A function $g(n) \in O\big(f(n)\big)$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that

$$\forall n \geq n_0 \qquad g(n) \leq c \cdot f(n)$$

$$2n^2 + 3n + 7 \;=\; O(n^2)$$

▷ $c = 3$ and $n_0 = 5$



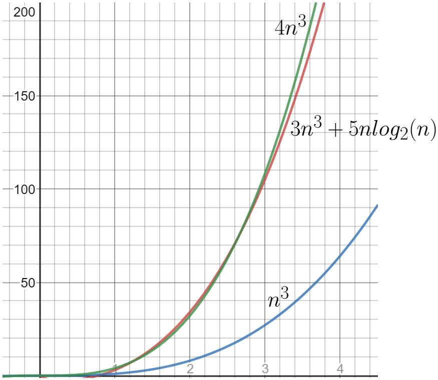
# Big Oh

A function $g(n) \in O\big(f(n)\big)$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that

$$\forall n \geq n_0 \qquad g(n) \leq c \cdot f(n)$$

$3n^3 + 5n \log n = O(n^3)$

▷ $c = 4$ and $n_0 = 3$



# Big Oh: Common Rules

The following two rules help simplify finding asymptotic upper bounds

- Lower order terms are ignored
  - $n^a$ dominates $n^b$ if $a > b$
- Multiplicative constants are omitted
  - e.g. $7n^4 + 3n^3 + 10 \ = \ O(n^4)$
  - e.g. $3n^3 + 5n \log n \ = \ O(n^3)$

# Big Oh: Common Rules

A function $g(n) \in O\big(f(n)\big)$ if there exists constants $c > 0$ and $n_0 \geq 0$ such that

$$\forall n \geq n_0 \qquad g(n) \leq c \cdot f(n)$$

$$\begin{aligned} f(n) &= pn^2 + qn + r \\ &\leq |p|n^2 + |q|n^2 + |r|n^2 \\ &= \big(|p| + |q| + |r|\big)n^2 \end{aligned}$$

This is true for all $n \geq 1$, hence with $c = \big(|p| + |q| + |r|\big)$ we get that $f(n) = O(n^2)$

# Big Oh: Justification to ignore lower order terms

Let the runtime of algorithm $\mathcal{A}$ be $\quad T(n) := n^2 + 10n$

$$T(n) = O(n^2)$$

Consider an input size of $10^9$, then

$$n^2 + 100n \ = \ 10^{18} + 10^{11} \qquad \text{and} \qquad n^2 \ = \ 10^{18}$$

$$\text{fractional error} \ = \ \frac{10^{11}}{10^{18}} \ = \ 10^{-7}$$

**Goal 1:** Determine the runtime of an algorithm on inputs of **large size**

For $n = 10^9$, $T(n) = n^2 + 10n$ is only 0.00001% more than $n^2$

# Big Oh: Justification to ignore Coefficients

Coefficients do not really affect growth of functions

**Goal 2:** Determine how the runtime **grows with increasing inputs**

| Linear $f(n) = 5n$ | Quadratic $f(n) = 7n^2$ | Cubic $f(n) = 2n^3$ |
|---|---|---|
| $n : 5n$ | $n : 7n^2$ | $n : 2n^3$ |
| $2n : 2(5n)$ | $2n : 4(7n^2)$ | $2n : 8(2n^3)$ |
| $3n : 3(5n)$ | $3n : 9(7n^2)$ | $3n : 27(2n^3)$ |
| $4n : 4(5n)$ | $4n : 16(7n^2)$ | $4n : 64(2n^3)$ |

Since we are concerned with scalability of algorithm, the same growth factor is observed if we consider $f(n) = n$, $f(n) = n^2$, and $f(n) = n^3$

# Non tightness of Big Oh

Note: $O(\cdot)$ expresses only an upper bound on growth rate of a function

$\triangleright$ It does not necessarily give the exact growth rate of the function

For example, $f(n) = 3n^2 + 4n + 5 \ = \ O(n^2)$ it is also $O(n^3)$

Indeed, $f(n) \ \leq \ 12n^2$ and also $f(n) \ \leq \ 12n^3$

# Big Oh: Finding the right constants

Let $g(n) \; = \; 7n + 4 \quad$ and $\quad f(n) \; = \; n$

$g(n) = O(f(n))$

$\triangleright$ take $c = 8$ and $n_0 = 4$, $\quad 7n + 4 \leq 8(n)$ whenever $n_0 \geq 4$,

1. To get $c$: We want $7n + 4 \leq cn$

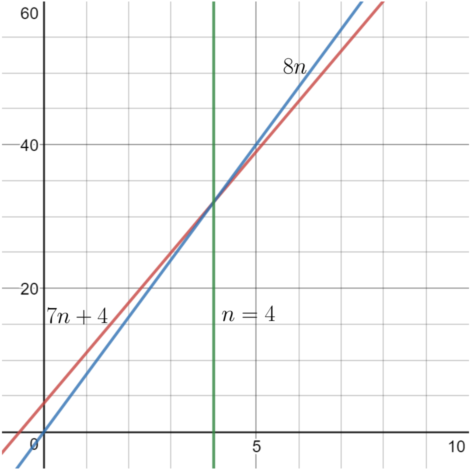
   Solving the inequality for $c$, we get $c \geq \frac{7n}{n} + \frac{4}{n}$

   For $n \geq 4$, $8 \geq 7 + \frac{4}{n}$

   $\triangleright$ Observe that $\lim_{n \to \infty} \frac{7n+4}{n} \to 7$, but this $(c = 7)$ would require $n_0$ to be approaching $\infty$, so we take $c = 8$

2. To get $n_0$: We want $7n + 4 \leq 8n$, this is true whenever $n \geq 4$

# Big Oh: Finding the right constants



# Big Oh: Finding the right constants

Let $f(n) \;=\; 6n + 24$ and $h(n) = n^2$,

$$f(n) \;=\; O(h(n))$$

As $\lim_{n\to\infty} \dfrac{6n+24}{n^2} \to 0$, so any $c > 0$ will work.

for $c = 1$, we want

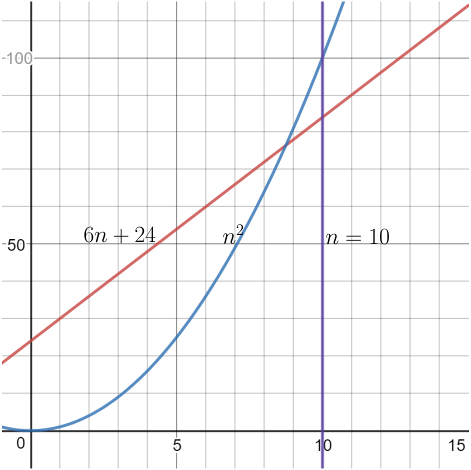
$6n + 24 \leq 1 \cdot n^2$

which is true whenever $n \geq 10$

So we choose $c = 1$ and $n_0 = 10$

More examples in lecture notes and problem set

# Big Oh: Finding the right constants

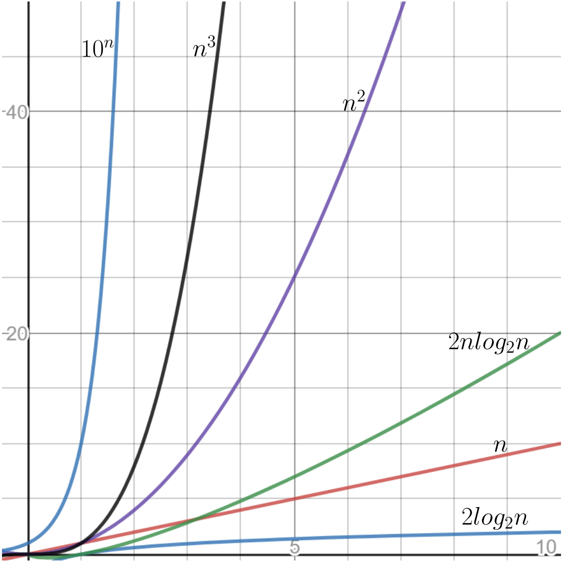


# Asymptotic-Complexity Classes

| **Class Name** | **Class Symbol** | **Example** |
|---|---|---|
| Constant | $O(1)$ | Comparison of two integers |
| Logarithmic | $O(log(n))$ | Binary Search, Exponentiation |
| Linear | $O(n)$ | Linear Search |
| Log-Linear | $On(log(n))$ | Merge Sort |
| Quadratic | $O(n^2)$ | Integer multiplications |
| Cubic | $O(n^3)$ | Matrix multiplication |
| Polynomial | $O(n^a)$, $a \in \mathbb{R}$ | |
| Exponential | $O(a^n)$, $a \in \mathbb{R}$ | Print all subsets |
| Factorial | $O(n!)$ | Print all permutations |

$$n! \gg 2^n \gg n^3 \gg n^2 \gg nlogn \gg n \gg logn \gg 1$$

# Growth Rates of Functions



# Big Oh: Why does it make sense?

Runtimes of algorithms of different runtime for input size $n$ (on 1GHz PC). Assume that each operation takes 1 $n$s

| **n** | $O(\log n)$ | $O(n)$ | $O(n \log n)$ | $O(n^2)$ | $O(2^n)$ | $O(n!)$ |
|---|---|---|---|---|---|---|
| 10 | $0.003\mu s$ | $0.01\mu s$ | $0.033\mu s$ | $0.1\mu s$ | $1\mu s$ | $3.63ms$ |
| 20 | $0.004\mu s$ | $0.02\mu s$ | $0.086\mu s$ | $0.4\mu s$ | $1ms$ | $77.1\ yrs$ |
| 30 | $0.005\mu s$ | $0.03\mu s$ | $0.147\mu s$ | $0.9\mu s$ | $1sec$ | $8 \cdot 10^{15}\ yrs$ |
| 40 | $0.005\mu s$ | $0.04\mu s$ | $0.213\mu s$ | $1.6\mu s$ | $18.3min$ | very long |
| 50 | $0.006\mu s$ | $0.05\mu s$ | $0.282\mu s$ | $2.5\mu s$ | $13\ days$ | very long |
| 100 | $0.007\mu s$ | $0.10\mu s$ | $0.644\mu s$ | $10\mu s$ | $4 \cdot 10^{13}\ yrs$ | very long |
| $10^3$ | $0.010\mu s$ | $1.00\mu s$ | $9.966\mu s$ | $1ms$ | very long | very long |
| $10^4$ | $0.013\mu s$ | $10\mu s$ | $130\mu s$ | $100ms$ | very long | very long |
| $10^5$ | $0.017\mu s$ | $0.10ms$ | $1.67ms$ | $10sec$ | very long | very long |
| $10^6$ | $0.020\mu s$ | $1ms$ | $19.93ms$ | $16.7min$ | very long | very long |
| $10^7$ | $0.023\mu s$ | $0.01sec$ | $0.23sec$ | $1.16\ days$ | very long | very long |
| $10^8$ | $0.027\mu s$ | $0.10sec$ | $2.66sec$ | $115.7\ days$ | very long | very long |
| $10^9$ | $0.030\mu s$ | $1sec$ | $29.90sec$ | $31.7\ yrs$ | very long | very long |