# Design and Analysis of Algorithms

Prepared by Dr. Muhammad Imran

# Three Cases of Analysis

- **Best Case:** constraints on the input, other than size, resulting in the fastest possible running time.
  - Searching an element in an Array?
- **Worst Case:** constraints on the input, other than size, resulting in the slowest possible running time.
  - Searching an element in an Array?
  - Searching an element in a sorted Array, using binary search?
- **Average Case:** average running time over every possible type of input (usually involve probabilities of different types of input)
  - Searching an element in an Array?

# Best-case, average-case, worst-case

- Worst case: – maximum over inputs of size *n*
- Best case: – minimum over inputs of size *n*
- Average case: – "average" over inputs of size *n*

  – NOT the average of worst and best case

  – Under some assumption about the probability distribution of all possible inputs of size *n,* calculate the weighted sum of expected C(n) (numbers of basic operation repetitions) over all inputs of size n.

# Example: Sequential search

- *Problem:* Given a list of *n* elements and a search key *K*, find an element equal to *K*, if any.
- *Algorithm:* Scan the list and compare its successive elements with *K* until either a matching element is found (*successful search*) or the list is exhausted (*unsuccessful search*)
  - Worst case ?
  - Best case ?
  - Average case ?

# Asymptotic growth rate

- A way of comparing functions that ignore constant factors and small input sizes

- $O(g(n))$: class of functions $f(n)$ that grow ***no faster*** than $g(n)$

- $\Theta(g(n))$: class of functions $f(n)$ that grow ***at same rate*** as $g(n)$

- $\Omega(g(n))$: class of functions $f(n)$ that grow ***at least as fast*** as $g(n)$

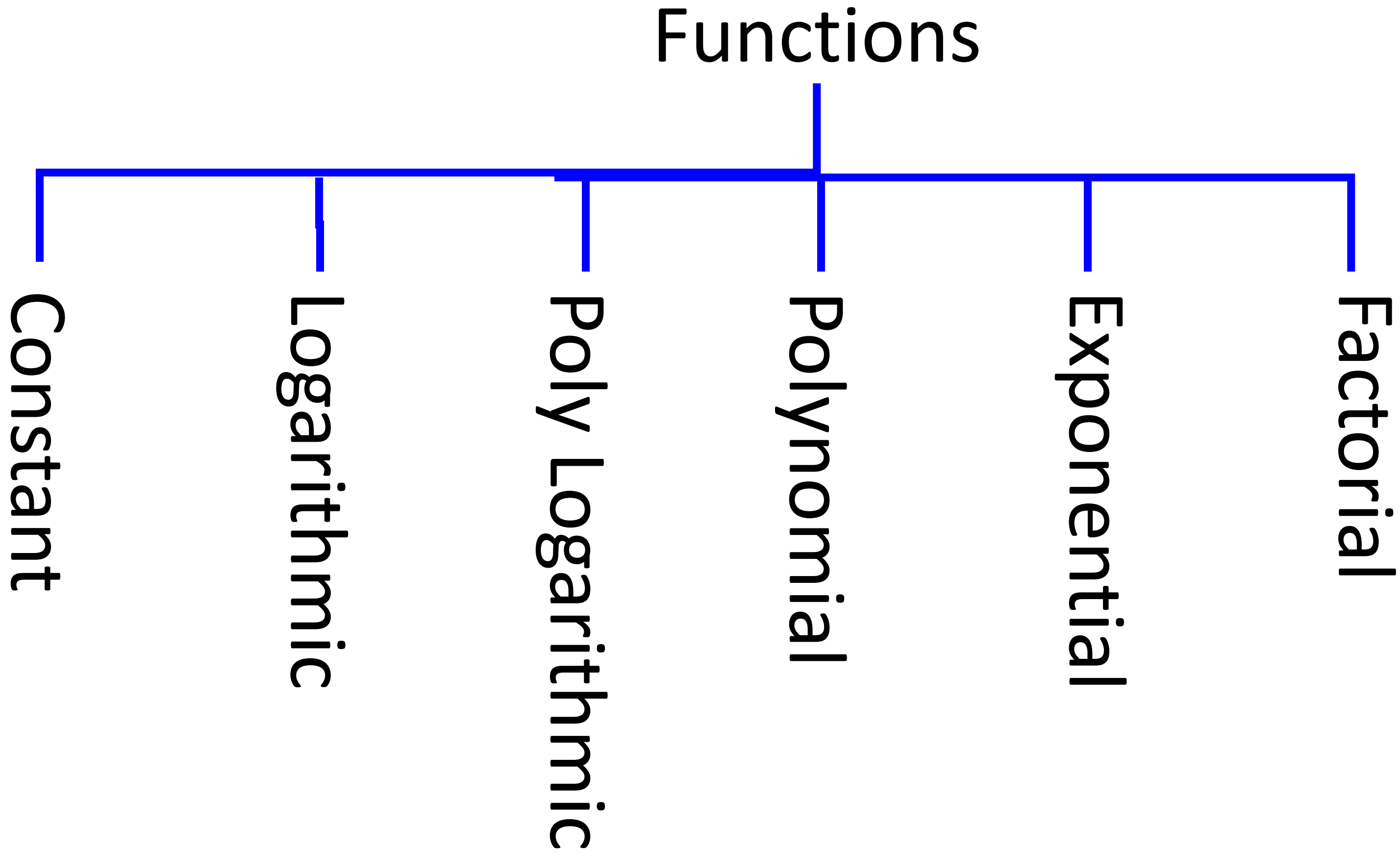
# Complexity Table

| $n$ | $\log_2 n$ | $n$ | $n \log_2 n$ | $n^2$ | $n^3$ | $2^n$ | $n!$ |
|---|---|---|---|---|---|---|---|
| $10$ | $3.3$ | $10^1$ | $3.3 \cdot 10^1$ | $10^2$ | $10^3$ | $10^3$ | $3.6 \cdot 10^6$ |
| $10^2$ | $6.6$ | $10^2$ | $6.6 \cdot 10^2$ | $10^4$ | $10^6$ | $1.3 \cdot 10^{30}$ | $9.3 \cdot 10^{157}$ |
| $10^3$ | $10$ | $10^3$ | $1.0 \cdot 10^4$ | $10^6$ | $10^9$ | | |
| $10^4$ | $13$ | $10^4$ | $1.3 \cdot 10^5$ | $10^8$ | $10^{12}$ | | |
| $10^5$ | $17$ | $10^5$ | $1.7 \cdot 10^6$ | $10^{10}$ | $10^{15}$ | | |
| $10^6$ | $20$ | $10^6$ | $2.0 \cdot 10^7$ | $10^{12}$ | $10^{18}$ | | |

**Table 2.1** Values (some approximate) of several functions important for analysis of algorithms

# Classifying Functions?

- Giving an idea of how fast a function grows without going into too much detail.



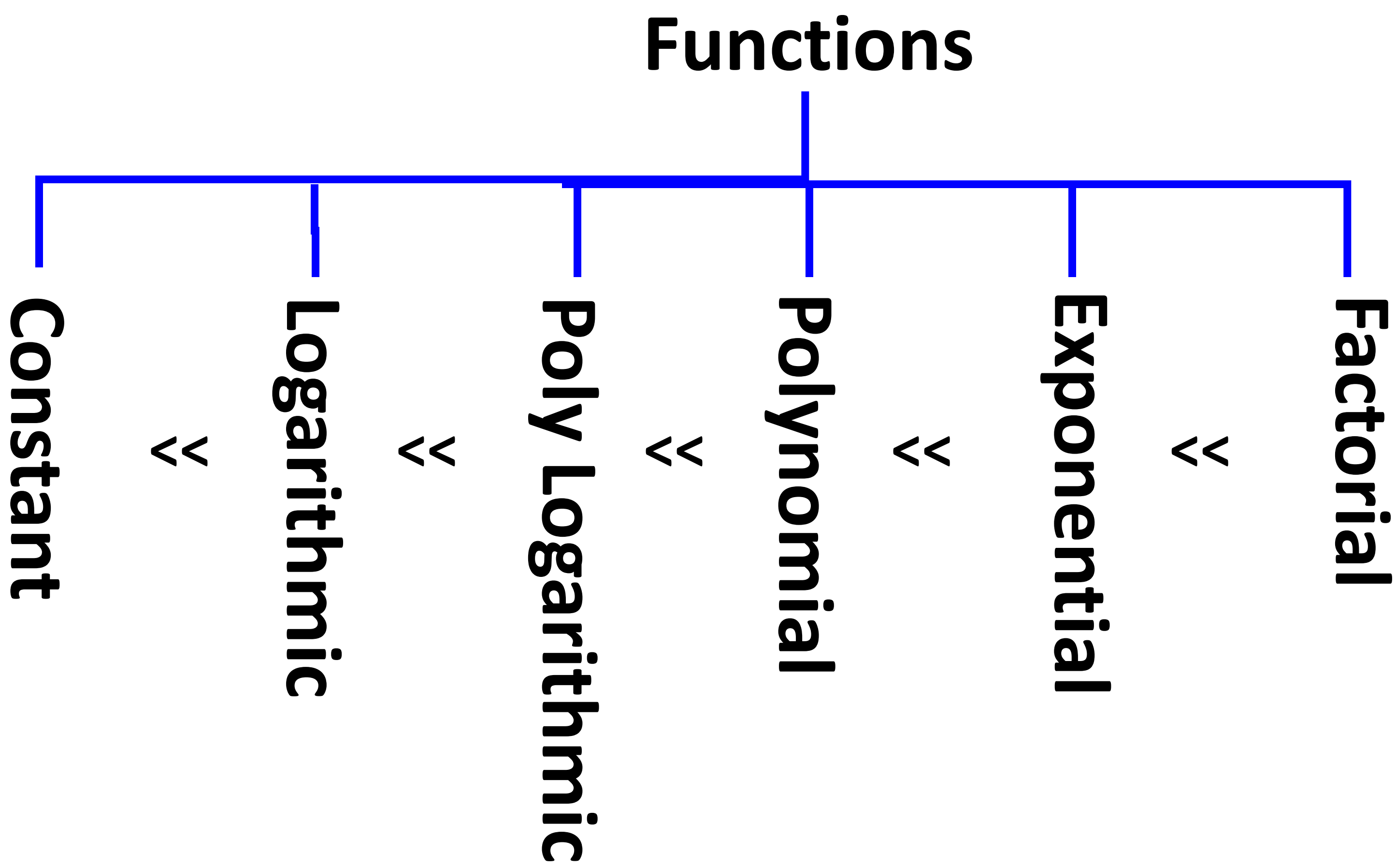

| | | | | | |
|---|---|---|---|---|---|
| **5** | **5 log n** | **$(\log n)^5$** | **$n^5$** | **$2^{5n}$** | **n!** |

# Classifying Functions

Polynomial

- Linear — **5n**
- Quadratic — **$5n^2$**
- Cubic — **$5n^3$**
- ? — **$5n^4$**

# Ordering Functions




$5 \ll \log n \ll (\log n)^2 \ll n^3 \ll 2^n \ll n!$

**For sufficiently large n**

# Which Functions are "Constant"?

The running time of the algorithm is a "Constant" if it does not depend <u>significantly</u> on the size of the input.

- 5
- 1,000
- 0.0001
- -5
- 0
- 8 + sin(n)

# Which Functions are Constant?

| | |
|---|---|
| Yes | • 5 |
| Yes | • 1,000 |
| Yes | • 0.0001 |
| Yes | • -5 |
| Yes | • 0 |
| No | • 8 + sin(n) |

# Which Functions are Quadratic?

- $n^2$
- $0.001\ n^2$
- $1000\ n^2$
- $5n^2 + 3n + 2\log n$

# Which Functions are Quadratic?

- $n^2$
- $0.001\ n^2$
- $1000\ n^2$
- $5n^2 + 3n + 2\log n$

Ignore low-order terms

Ignore multiplicative constants.

Ignore "small" values of n.

Write $\theta(n^2)$.

# Analyzing Algorithms

- Simplicity
  - Informal, easy to understand, easy to change etc.
- Time efficiency
  - As a function of its input size, how long does it take?
- Space efficiency
  - As a function of its input size, how much additional space does it use?
- Running time
  - Depends on the number of primitive operations (addition, multiplication, comparisons) used to solve the problem and on problem instance.

# Big-O Common Names

constant: $O(1)$

logarithmic: $O(\log n)$

linear: $O(n)$

log-linear: $O(n \log n)$

superlinear: $O(n^{1+c})$ (c is a constant, where $0 < c < 1$)

quadratic: $O(n^2)$

polynomial: $O(n^k)$ (k is a constant)

exponential: $O(c^n)$ (c is a constant > 1)

# Asymptotic Complexity

- Running time of an algorithm as a function of input size *n* **for large *n***.
- Expressed using only the **highest-order term** in the expression for the exact running time.
  - Instead of exact running time, say $\Theta(n^2)$.
- Describes behavior of function within certain limit.
  - Written using ***Asymptotic Notation.***

# Types of Analysis

- **Worst case**
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are
- **Best case**
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest

$$\boxed{\textit{Lower Bound} \leq \textit{Running Time} \leq \textit{Upper Bound}}$$

- **Average case**
  - Provides a prediction about the running time
  - Assumes that the input is random

# How do we compare algorithms?

- We need to define a number of objective measures

  (1) Compare execution times?

  ***Not good***: times are specific to a particular computer !!

  (2) Count the number of statements executed?

  ***Not good***: number of statements vary with the programming language as well as the style of the individual programmer.

- **Ideal Solution**
- Express running time as a function of the input size ***n*** (i.e., *f(n)*).
- Compare different functions corresponding to running times.
- Such an analysis is independent of machine time, programming style, etc.

# Rate of Growth

- Consider the example of buying *elephants* and *goldfish:*

  **Cost**: cost_of_elephants + cost_of_goldfish

  **Cost** ~ cost_of_elephants (approximation)

- The low order terms in a function are relatively insignificant for **large** *n*

$$n^4 + 100n^2 + 10n + 50 \quad \sim \quad n^4$$

*i.e.,* we say that $n^4 + 100n^2 + 10n + 50$ and $n^4$ have the same **rate of growth**

# Big-O Notation

- We say $f_A(n)=30n+8$ is *order n*, or O (n) It is, at most, roughly *proportional* to $n$
- $f_B(n)=n^2+1$ is *order* $n^2$, or $O(n^2)$. It is, at most, roughly proportional to $n^2$
- In general, any $O(n^2)$ function is faster-growing (in terms of computation) than any $O(n)$ function
- Therefore, $O(n^2)$ function is slower than $O(n)$ function

# More Examples

- $n^4 + 100n^2 + 10n + 50$ is $O(n^4)$
- $10n^3 + 2n^2$ is $O(n^3)$
- $n^3 - n^2$ is $O(n^3)$
- constants
  - 10 is $O(1)$
  - 127 is $O(1)$