Data Structure

Sheet2

__init__() Function:

- Always executed when creating an object of the class.
- It automatically initializes object attributes when an object is created (like a constructor).

Self Parameter:

- Is a reference to the current instance of the class.
- It allows us to access the attributes and methods of the object.

```
class Dog:
    species = "Canine"  # Class attribute

    def __init__(self, name, age):
        self.name = name  # Instance attribute
        self.age = age  # Instance attribute
```

Notes:

- Specifying a data type to a variable is done using **Type Annotations**.
- It indicates the expected type of a variable od function argument/return value.
  - They don't enforce the type at runtime
- **Syntax:**
  - For Variables: `variable_name: type`
  - For function args/return: `parameter_name: type  ->return_type`
- __str__() method in Python allows us to define a custom string representation of an object.

CreditCard class: Lecture Example:

```
class CreditCard:
  """A consumer credit card."""

  def __init__(self, customer, bank, acnt, limit):
    """Create a new credit card instance.

    The initial balance is zero.

    customer  the name of the customer (e.g., 'John Bowman')
    bank      the name of the bank (e.g., 'California Savings')
    acnt      the acount identifier (e.g., '5391 0375 9387 5309')
    limit     credit limit (measured in dollars)
    """
    self._customer = customer
    self._bank = bank
    self._account = acnt
    self._limit = limit
    self._balance = 0
```

```python
def get_customer(self):
  """Return name of the customer."""
  return self._customer

def get_bank(self):
  """Return the bank's name."""
  return self._bank

def get_account(self):
  """Return the card identifying number (typically stored as a string)."""
  return self._account

def get_limit(self):
  """Return current credit limit."""
  return self._limit

def get_balance(self):
  """Return current balance."""
  return self._balance
```

```
def charge(self, price):
  """Charge given price to the card, assuming sufficient credit limit.

  Return True if charge was processed; False if charge was denied.
  """
  if price + self._balance > self._limit:      # if charge would exceed limit,
    return False                               # cannot accept charge
  else:
    self._balance += price
    return True

def make_payment(self, amount):
  """Process customer payment that reduces balance."""
  self._balance −= amount
```

Complexity Analysis:

- Complexity:
  - It describes how the runtime or space requirement of an algorithm grow as the input size increases.
  - The length of an input determines how many operations the algorithm will do.
- E.g.: Array Search:

10	20	18	5	99	100	13

Best Case: Ω

Average Case: Θ

Worst Case: O

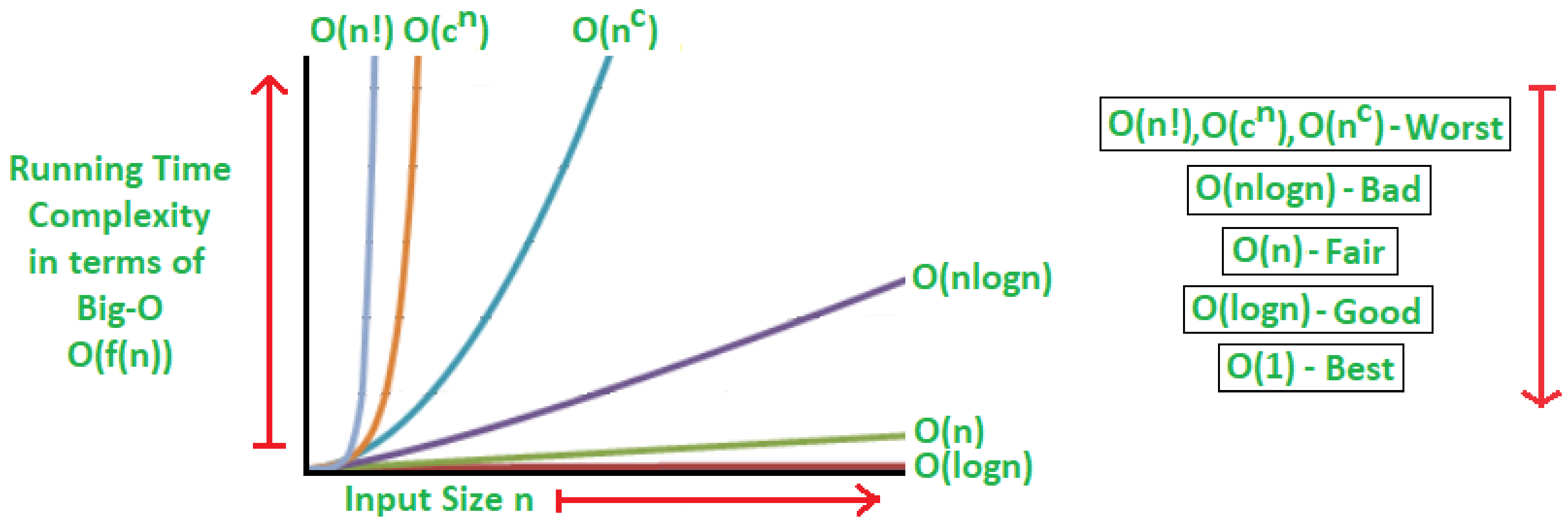
- These notations are used to measure the time complexity and are called **Asymptotic Notations**.

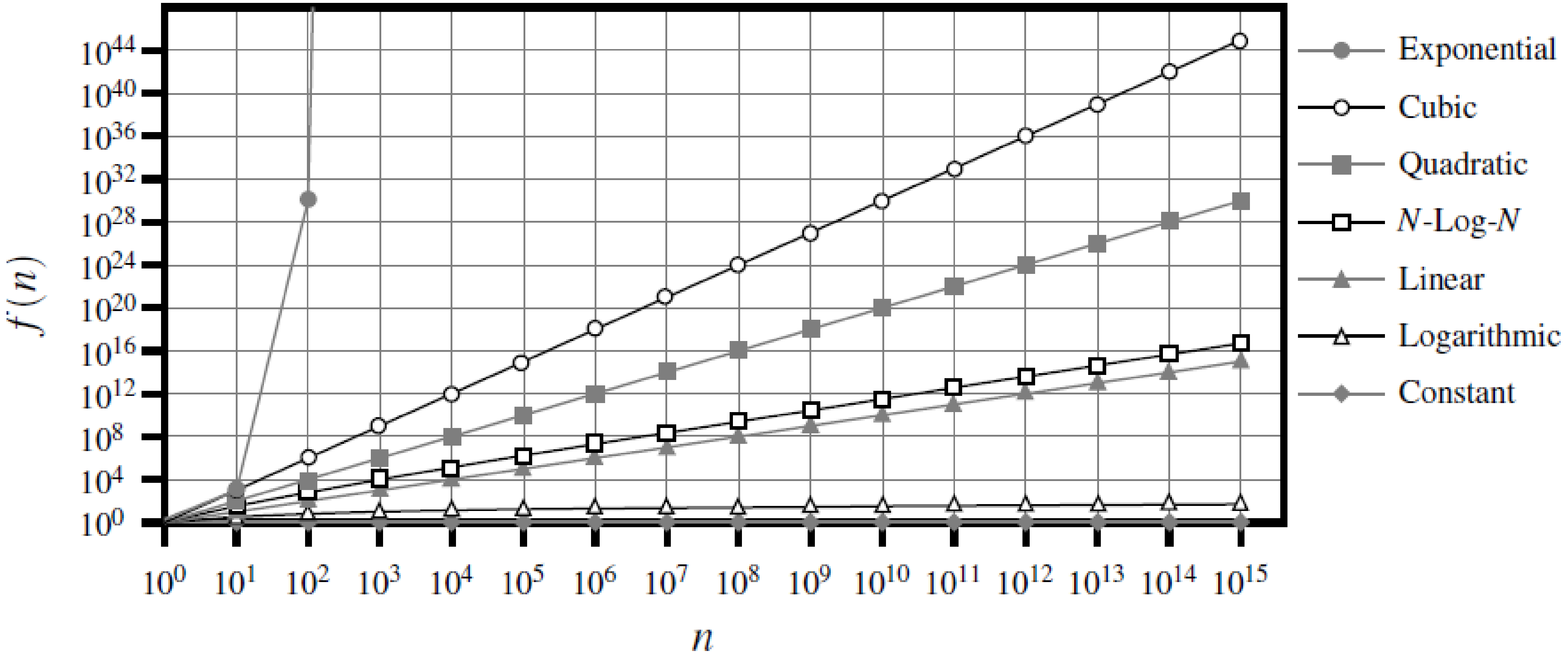
Big-O Notations:

- Big-O notation is a way to measure the time and space complexity of an algorithm. It describes the upper bound of the complexity in the worst-case scenario.
- Common Big-O Notations:
  - Linear Time Complexity: O(n)
    - Array Linear Search
  - Logarithmic Time Complexity: O(logn)
    - Binary Search
  - Quadratic Time Complexity: O(n^2)
    - Bubble-sort algorithm
  - Cubic Time Complexity: O(n^3)
    - Matrix multiplication algorithm

Rate of Growth:

- The faster the function grows, the more time it takes, the worst the function.





Exponential
Cubic
Quadratic
N-Log-N
Linear
Logarithmic
Constant
f(n)
n
10^0
10^4
10^8
10^{12}
10^{16}
10^{20}
10^{24}
10^{28}
10^{32}
10^{36}
10^{40}
10^{44}
10^0 10^1 10^2 10^3 10^4 10^5 10^6 10^7 10^8 10^9 10^{10} 10^{11} 10^{12} 10^{13} 10^{14} 10^{15}

3. The number of operations executed by algorithms A and B is A $8n\log(n)$ and B $2n^2$, respectively. Determine n0 such that A is better than B for $n \geq n0$.

$A < B$

$$\frac{8n\log n}{2n} < \frac{2n^2}{2n}$$

$$\boxed{4\log n < n}$$

n	$4\log n$	$4\log n < n$	
2	$4\log(2) = 4$	$4 < 2$	→ ×
4	$4\log(4) = 8$	$8 < 4$	→ ×
8	$4\log(8) = 12$	$12 < 8$	→ ×
16	$= 16$	$16 < 16$	→ ×
32	$= 20$	$20 < 32$	→ ✓

$$\boxed{n_0 = 32}$$

4. The number of operations executed by algorithms A and B is $40n^2$ and $2n^3$, respectively. Determine n0 such that A is better than B for $n \geq n0$.

$$A < B$$

$$\frac{\cancel{40}\cancel{n^2}}{\cancel{2}\cancel{n^2}} < \frac{\cancel{2}\cancel{n^3}}{\cancel{2}\cancel{n^2}}$$

$$20 < n$$

$$\hookrightarrow \quad n > 20 \longrightarrow \boxed{n_0 = 21} \#$$

5. Order the following functions by asymptotic growth rate.

Slower → Faster

- A $3n+100\log(n) \rightarrow O(n)$
- B $4n \rightarrow O(n)$ ≈
- C $2^n \rightarrow O(2^n) \rightarrow$ exp. Faster

$$3n+100\log(n) \approx 4n < 2^n$$

Thank You…

Eng. Alaa Abdulfattah