Sheet 2

1. **Write a Python class, Flower, that has three instance variables of type str, int, and float, that respectively represent the name of the flower, its number of petals, and its price. Your class must include a constructor method that initializes each variable to an appropriate value, and your class should include methods for setting the value of each type and retrieving the value of each type.**

Solution

```
class Flower:
    def __init__(self, name: str, petals: int, price: float):
        self.name = name
        self.petals = petals
        self.price = price

    def set_name(self, name: str):
        self.name = name

    def set_petals(self, petals: int):
        self.petals = petals

    def set_price(self, price: float):
        self.price = price

    def get_name(self) -> str:
        return self.name

    def get_petals(self) -> int:
        return self.petals

    def get_price(self) -> float:
        return self.price

    def __str__(self):
        return f"Flower(Name: {self.name}, Petals: {self.petals}, Price
: ${self.price:.2f})"
```

Sheet 2

```
# Example usage
rose = Flower("Rose", 12, 5.99)

print(rose)

# Update details using setter methods
rose.set_name("Red Rose")
rose.set_petals(15)
rose.set_price(7.49)

print(rose)

# Retrieve details using getter methods
print(f"Name: {rose.get_name()}")
print(f"Petals: {rose.get_petals()}")
print(f"Price: ${rose.get_price():.2f}")
```

Sheet 2

2. The CreditCard class in the lecture initializes the balance of a new account to zero. Modify that class so that a new account can be given a nonzero balance using an optional fifth parameter to the constructor. The four-parameter constructor syntax should continue to produce an account with zero balance.

Solution

```
class CreditCard:
    def __init__(self, customer, bank, account, lim, bal = 0.0):
        self.customerName = customer
        self.bankName = bank
        self.accountNumber = account
        self.limit = lim
        self.balance = bal

    def get_customer(self):
        return self.customerName

    def get_bank(self):
        return self.bankName

    def get_account(self):
        return self.accountNumber

    def get_limit(self):
        return self.limit

    def get_balance(self):
        return self.balance

    def charge(self, price):

        if price + self.balance > self.limit:
            return False
        else:
            self.balance += price
```

Sheet 2

```
            return True



    def make_payment(self, amount):
        if amount < 0:
            raise ValueError("Payment amount cannot be negative.")
        self.balance -= amount

    def __str__(self):
        return (f"CreditCard(Customer: {self.customerName}, Bank: {self
.bankName}, "
                f"Account: {self.accountNumber}, Limit: ${self.limit:.2
f}, Balance: ${self.balance:.2f})")



# Example usage
card1 = CreditCard("waleed mamdouh", "Bank of Egypt", "1234 5678 9012 3
456", 1000.0)
print(card1)

card2 = CreditCard("mohamed mamdouh", "Bank of Alahly", "9876 5432 1098
 7654", 2000.0, 500.0)
print(card2)

# Charge and make payments
card1.charge(200.0)
card2.make_payment(100.0)
print(card1)
print(card2)
```

Sheet 2

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

Solution

Given:

- Algorithm **A**: $8n \log n$ operations
- Algorithm **B**: $2n^2$ operations

We want to find the smallest integer n_0 such that for all $n \geq n_0$:

$$8n \log n < 2n^2$$

Step 1: Simplify the Inequality

Divide both sides of the inequality by $2n$ (since $n > 0$):

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

Simplify:

$$4 \log n < n$$

So, the inequality reduces to:

$$4 \log n < n$$

Sheet 2

Step 2: Solve the Inequality

We need to find the smallest integer n_0 such that:

$$4 \log n < n$$

This inequality cannot be solved algebraically, so we will solve it **numerically** by testing values of n.

Step 3: Test Values of n

Let's test integer values of n to find the smallest n_0 where $4 \log n < n$:

n	$4 \log n$	n	Is $4 \log n < n$?
1	$4 \log 1 = 0$	1	$0 < 1$ ✅
2	$4 \log 2 \approx 2.772$	2	$2.772 < 2$ ❌
3	$4 \log 3 \approx 4.394$	3	$4.394 < 3$ ❌
4	$4 \log 4 \approx 5.545$	4	$5.545 < 4$ ❌
5	$4 \log 5 \approx 6.438$	5	$6.438 < 5$ ❌
6	$4 \log 6 \approx 7.167$	6	$7.167 < 6$ ❌
7	$4 \log 7 \approx 7.778$	7	$7.778 < 7$ ❌
8	$4 \log 8 \approx 8.318$	8	$8.318 < 8$ ❌
9	$4 \log 9 \approx 8.803$	9	$8.803 < 9$ ✅
10	$4 \log 10 \approx 9.210$	10	$9.210 < 10$ ✅

Step 4: Determine n_0

From the table:

- For $n = 9$, $4 \log 9 \approx 8.803 < 9$. ✅
- For $n = 10$, $4 \log 10 \approx 9.210 < 10$. ✅

Thus, the smallest integer n_0 where $4 \log n < n$ is $n_0 = 9$.

Sheet 2

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 ≥ n0.

Solution

Given:

- Algorithm **A**: $40n^2$ operations
- Algorithm **B**: $2n^3$ operations

We want to find the smallest integer n_0 such that for all $n \geq n_0$:

$$40n^2 < 2n^3$$

Step 1: Simplify the Inequality

Divide both sides of the inequality by $2n^2$ (since $n > 0$):

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

Simplify:

$$20 < n$$

So, the inequality reduces to:

$$n > 20$$

Step 2: Determine n_0

From the simplified inequality $n > 20$, the smallest integer n_0 satisfying this condition is:

$$n_0 = 21$$

Sheet 2

5. Order the following functions by asymptotic growth rate.

- 3n+100log(n)
- 4n
- 2^n

Solution

Given Functions:

1. $3n + 100\log(n)$
2. $4n$
3. 2^n

Step 1: Identify Dominant Terms

- For each function, identify the term that grows the fastest as $n \to \infty$:
  1. $3n + 100\log(n)$: Dominant term is $3n$ ($O(n)$).
  2. $4n$: Dominant term is $4n$ ($O(n)$).
  3. 2^n: Dominant term is 2^n ($O(2^n)$).

Step 2: Compare Growth Rates

- $O(n)$ (linear growth) grows slower than $O(2^n)$ (exponential growth).
- Both $3n + 100\log(n)$ and $4n$ are $O(n)$, so they grow at the **same rate**.

Step 3: Order the Functions

From slowest-growing to fastest-growing:

1. $3n + 100\log(n)$ and $4n$ (both $O(n)$).
2. 2^n ($O(2^n)$).

Final Order:

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

Sheet 2

6. Order the following functions by asymptotic growth rate.

- n^2 +10n
- n^3
- nlog(n)

Solution

Step 1: Identify Dominant Terms

For each function, identify the term that grows the fastest as $n \to \infty$:

1. $n^2 + 10n$:
   - The dominant term is n^2 ($O(n^2)$).
2. n^3:
   - The dominant term is n^3 ($O(n^3)$).
3. $n \log(n)$:
   - The dominant term is $n \log(n)$ ($O(n \log(n))$).

Step 2: Compare Growth Rates

The growth rates of the dominant terms are:

1. $O(n \log(n))$: Grows faster than linear but slower than quadratic.
2. $O(n^2)$: Quadratic growth.
3. $O(n^3)$: Cubic growth.

Step 3: Order the Functions

From slowest-growing to fastest-growing:

1. $n \log(n)$ ($O(n \log(n))$).
2. $n^2 + 10n$ ($O(n^2)$).
3. n^3 ($O(n^3)$).

Final Order:

$$n \log(n) < n^2 + 10n < n^3$$

Sheet 2

7. Show that $(n+1)^5$ is $O(n^5)$.

Solution

Goal

Show that $(n+1)^5$ is $O(n^5)$, meaning there exist constants $C > 0$ and $n_0 \geq 1$ such that:

$$(n+1)^5 \leq C \cdot n^5 \quad \text{for all } n \geq n_0.$$

Step 1: Simplify $(n+1)^5$

For large n, $n+1 \approx n$, so:

$$(n+1)^5 \approx n^5.$$

Step 2: Compare $(n+1)^5$ and n^5

Divide both sides of the inequality $(n+1)^5 \leq C \cdot n^5$ by n^5:

$$\frac{(n+1)^5}{n^5} \leq C.$$

Simplify the left-hand side:

$$\frac{(n+1)^5}{n^5} = \left(1 + \frac{1}{n}\right)^5.$$

Step 3: Analyze $\left(1 + \frac{1}{n}\right)^5$

As $n \to \infty$, $\frac{1}{n} \to 0$, so:

$$\left(1 + \frac{1}{n}\right)^5 \to 1.$$

This means that for large n, $\left(1 + \frac{1}{n}\right)^5$ is close to 1. Therefore, we can choose $C = 2$ (or any constant greater than 1) and find n_0 such that:

$$\left(1 + \frac{1}{n}\right)^5 \leq 2 \quad \text{for all } n \geq n_0.$$

Sheet 2

Step 4: Find n_0

Solve for n in the inequality:

$$\left(1+\frac{1}{n}\right)^5 \leq 2.$$

Take the 5th root of both sides:

$$1+\frac{1}{n} \leq 2^{1/5}.$$

Calculate $2^{1/5}$:

$$2^{1/5} \approx 1.1487.$$

Subtract 1 from both sides:

$$\frac{1}{n} \leq 0.1487.$$

Solve for n:

$$n \geq \frac{1}{0.1487} \approx 6.72.$$

Since n must be an integer, we choose $n_0 = 7$.

Step 5: Verify the Inequality

For $n \geq 7$:

$$\left(1+\frac{1}{n}\right)^5 \leq 2.$$

Thus:

$$(n+1)^5 \leq 2 \cdot n^5 \quad \text{for all } n \geq 7.$$

Sheet 2

8. Show that 2^{n+1} is $O(2^n)$

Solution

Step 1: Simplify 2^{n+1}

Using the properties of exponents:

$$2^{n+1} = 2^n \cdot 2^1 = 2 \cdot 2^n.$$

Step 2: Compare 2^{n+1} and 2^n

We want to show that 2^{n+1} is bounded by $C \cdot 2^n$ for some constant C. From the simplification above:

$$2^{n+1} = 2 \cdot 2^n.$$

Thus, we can choose $C = 2$ and $n_0 = 1$, and the inequality:

$$2^{n+1} \leq 2 \cdot 2^n$$

holds for all $n \geq 1$.

Sheet 2

9. Show that n^2 is $\Omega(n\log(n))$.

Solution

To show that n^2 is $\Omega(n\log(n))$, we need to prove that there exist positive constants c and n_0 such that for all $n \geq n_0$, the following inequality holds:

$$n^2 \geq c \cdot n\log(n)$$

Step-by-Step Proof:

1. **Simplify the Inequality:**
   Divide both sides of the inequality by n (since $n > 0$):
   $$n \geq c \cdot \log(n)$$
2. **Choose a Constant c:**
   Let $c = 1$. Then the inequality becomes:
   $$n \geq \log(n)$$
3. **Verify the Inequality for $n \geq n_0$:**
   We need to find n_0 such that for all $n \geq n_0$, $n \geq \log(n)$.
   - For $n = 1$: $1 \geq \log(1) = 0$ (true).
   - For $n = 2$: $2 \geq \log(2) \approx 0.693$ (true).
   - For $n = 10$: $10 \geq \log(10) \approx 2.302$ (true).

   In general, for $n \geq 1$, $n \geq \log(n)$ holds because n grows much faster than $\log(n)$.
4. **Conclusion:**
   Since we have found $c = 1$ and $n_0 = 1$ such that for all $n \geq n_0$, $n^2 \geq c \cdot n\log(n)$, we conclude that n^2 is $\Omega(n\log(n))$.

Sheet 2

10. Show that nlog(n) is Ω(n).

Solution

To show that $n \log(n)$ is $\Omega(n)$, we need to prove that there exist positive constants c and n_0 such that for all $n \geq n_0$, the following inequality holds:

$$n \log(n) \geq c \cdot n$$

Step-by-Step Proof:

1. **Simplify the Inequality:**
   Divide both sides of the inequality by n (since $n > 0$):
   $$\log(n) \geq c$$
2. **Choose a Constant c:**
   Let $c = 1$. Then the inequality becomes:
   $$\log(n) \geq 1$$
3. **Verify the Inequality for $n \geq n_0$:**
   We need to find n_0 such that for all $n \geq n_0$, $\log(n) \geq 1$.
   - For $n = 2$: $\log(2) \approx 0.693$ (false).
   - For $n = 10$: $\log(10) \approx 2.302$ (true).

   In general, $\log(n) \geq 1$ holds for $n \geq e$ (where $e \approx 2.718$), since $\log(e) = 1$.

   Thus, we can choose $n_0 = 3$ (or any integer greater than e).
4. **Conclusion:**
   Since we have found $c = 1$ and $n_0 = 3$ such that for all $n \geq n_0$, $n \log(n) \geq c \cdot n$, we conclude that $n \log(n)$ is $\Omega(n)$.

Sheet 2

Note:

For any two bases a and b, the logarithm of n with respect to these bases is related by the change of base formula:

$$\log_a(n) = \frac{\log_b(n)}{\log_b(a)}$$

Here, $\log_b(a)$ is a constant. Since Big-O, Big-Omega, and Big-Theta notation ignore constant factors, the choice of base does not affect the asymptotic behavior.