ANALYSIS OF ALGORITHM

Bùi Tiến Lên

2024

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Contents

1. Introduction
2. Types Of Problem
3. Asymptotic
4. Mathematical Techniques
5. Algorithm Analysis
6. The Master Theorem
7. Workshop

Introduction

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Analysis of Algorithms

Concept 1

Analysing an algorithm has come to mean predicting the **resources** that the algorithm requires.

- **Time Complexity**: How long it takes the algorithm to run
- **Space Complexity**: The extra memory space the algorithm needs besides the input data itself.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Why Analyze Algorithms?

- **Comparison**: To compare different algorithms and choose the most efficient one for a given task.
- **Optimization**: To optimize an algorithm.
- **Predictability**: To predict how an algorithm will behave with different input sizes and under various conditions.
- **Resource Management**: To ensure algorithms are efficient in terms of computational resources, such as CPU time and memory.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Methods of Analysis

There are two methods of analysis

- Empirical analysis
- Theoretical analysis

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Time Complexity

Concept 2

The **time complexity** of an algorithm estimates how much time the algorithm will use given an input.

$$\text{time complexity} = T(\text{input size}) \tag{1}$$

Input size depends on the problem's input data.

- The input size can be the *number of items in the input*
- The input size is the *total number of bits* needed to represent the input.
- Sometimes, it is more appropriate to describe the size of the input with two or more *numbers*.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Time Complexity (cont.)

- Time complexity isn't directly measured in seconds or minutes. It measures how many basic operations are executed.

Concept 3

Basic Operations can be

- Machine operation: assignments, arithmetic operations, ...
- Human operation: move, carry, ...
- General operation

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Space Complexity

Concept 4

The **space complexity** of an algorithm estimates the amount of extra memory space the algorithm needs to perform its operations on an input data.

$$\text{space complexity} = T(\text{input size}) \tag{2}$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Example

- Given an array of n numbers, our task is to calculate the **maximum subarray sum**, i.e., the largest possible sum of a sequence of consecutive values in the array.
- We assume that an empty subarray is allowed, so the maximum subarray sum is always at least 0.
- For example, in the array

-1	2	4	-3	5	2	-5	2

the following subarray produces the maximum sum 10:

-1	2	4	-3	5	2	-5	2

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Algorithm 1

- A straightforward way to solve the problem is to go through all possible subarrays, calculate the sum of values in each subarray and maintain the maximum sum.

```
int best = 0;
for (int a = 0; a < n; a++) {
    for (int b = a; b < n; b++) {
        int sum = 0;
        for (int k = a; k <= b; k++) {
            sum += array[k];
        }
        best = max(best,sum);
    }
}
cout << best << "\n";
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Algorithm 2

- It is easy to make Algorithm 1 more efficient by removing one loop from it. This is possible by calculating the sum at the same time when the right end of the subarray moves.

```
int best = 0;
for (int a = 0; a < n; a++) {
    int sum = 0;
    for (int b = a; b < n; b++) {
        sum += array[b];
        best = max(best,sum);
    }
}
cout << best << "\n";
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Algorithm 3

Consider the subproblem of finding the maximum-sum subarray that ends at position k. There are two possibilities:

1. The subarray only contains the element at position k.
2. The subarray consists of a subarray that ends at position $k-1$, followed by the element at position k.

In the latter case, since we want to find a subarray with maximum sum, the subarray that ends at position $k-1$ should also have the maximum sum. Thus, we can solve the problem efficiently by calculating the maximum subarray sum for each ending position from left to right.

```
int best = 0, sum = 0;
for (int k = 0; k < n; k++) {
    sum = max(array[k],sum+array[k]);
    best = max(best,sum);
}
cout << best << "\n";
```

Types Of Problem

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Types Of Problem

Concept 5

A **decision problem** is a problem for which the answer for every valid input is **yes** or **no**.

- For example, "deciding whether a number is prime" is a decision problem.

Concept 6

A **search problem** is a problem which requires the identification of a solution from within a potentially finite/infinite set of possible solutions.

- For example, "finding a path" is a search problem.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Types Of Problem (cont.)

Concept 7

A **counting problem** requires a total of the solutions to a **search problem**.

- For example, "how many of the first 100 integers are prime?".

Concept 8

A **optimization problem** requires the identification of the best solution to a **search problem** from a given set of solutions.

- For example, "finding the shortest path".

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Tractable & Intractable Problems

Concept 9

If a problem has a reasonable time solution, that is to say that it can be solved in no more than polynomial time, it is said to be **tractable**.

Concept 10

A problem can only be solved with algorithms whose execution time grows too quickly in relation to their input to be solved in polynomial time. These algorithms are said to be **intractable**.

- For example, The **Travelling Salesperson Problem** is an intractable problem.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Approximate Solutions

Concept 11

A **heuristic algorithm** is an algorithm that produces usable solutions to a problem in reasonable (polynomial) time. These solutions may not be proven to be optimal or even formally correct, they can however produce good solutions to the problem.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Decidable/Undecidable Problems

Concept 12

There are problems that can not be solved by algorithms even with unlimited time. They are uncomputable problems.

Concept 13

If a decision problem proves to be uncomputable then it is said to be **undecidable**.

The Halting Problem

Given a description of a program (its code), and some input, is it possible to tell, without running the program, whether it will halt for the given input or loop forever?

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Complexity Classes

Concept 14

Complexity classes are problem groups based on how much time and space they require to solve problems and verify solutions.

Concept 15

The class **P** consists of all those decision problems that can be solved on a deterministic sequential machine in an amount of time that is polynomial in the size of the input.

Concept 16

The class **NP** consists of all those decision problems whose positive solutions can be verified in polynomial time given the right information.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Complexity Classes (cont.)

Concept 17

A decision problem C is **NP-complete** if

1. C is in **NP**, and
2. Every problem in **NP** is reducible to C in polynomial time

- A problem satisfying condition 2 is said to be **NP-hard**, whether or not it satisfies condition 1

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

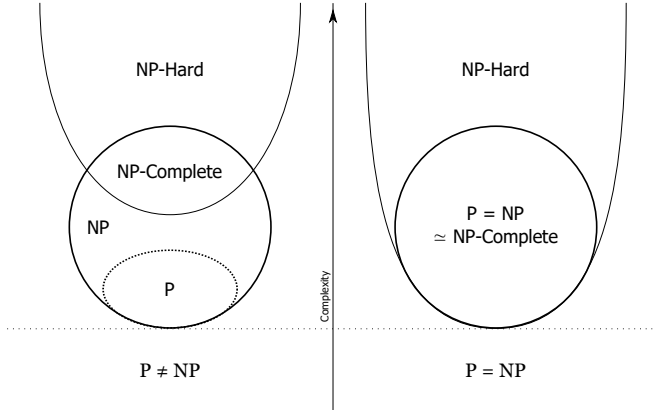
Some NP-complete Problems

- Clique problem
- n-Queens completion
- Boolean satisfiability problem
- Subset sum problem
- Traveling salesman problem
- ...

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

P = NP?

The unsolved problem "the question of whether P = NP?"



Asymptotic

- Big O Notation
- Big Omega Notation
- Big Theta Notation

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Big O Notation

History

- The symbol O was first introduced by number theorist Paul Bachmann in 1894
- The number theorist Edmund Landau adopted it, and was thus inspired to introduce in 1909 the notation o
- In the 1970s the big O was popularized in computer science by Donald Knuth

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Big O Notation (cont.)

Concept 18

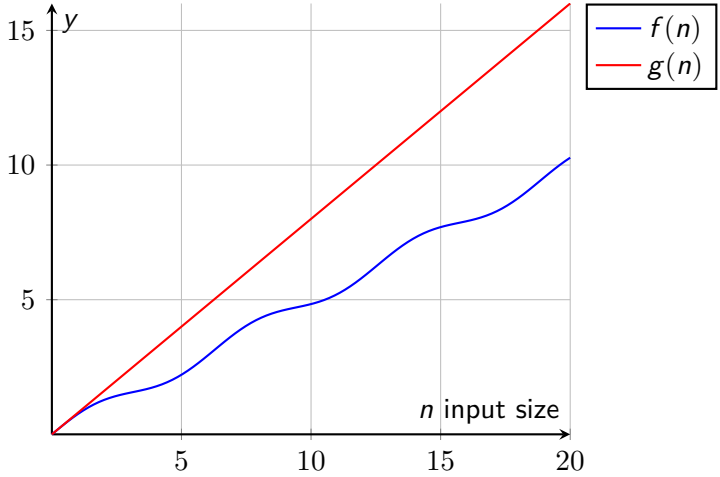
Given function $g(n)$, we denote by $O(g(n))$ the set of functions $\{f(n)\}$, there exist positive constants c and n_0 such that

$$0 \leq f(n) \leq cg(n), \text{ for all } n \geq n_0 \tag{3}$$

- We pronounce $O(g(n))$ as "big-oh of g of n" or sometimes just "oh of g of n"
- **Big O Notation** represents the upper bound on the growth rate of an algorithm's time or space complexity.
- $T(n) = O(g(n))$ to indicate that a function $T(n)$ is a member of the set $O(g(n))$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Big O Notation (cont.)



Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Big O Notation (cont.)

- Common order-of-growth functions $g(n)$

Table 1: Order-of-growth function

name	Order-of-growth function
constant	1
logarithm	$\log n$
linear	n
n-log-n	$n \log n$
quadratic	n^2
cubic	n^3
exponential	2^n
permuation	$n!$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Examples

Example 1

Prove that $T(n) = 2n^4 + 3n^3 + 5n^2 + 2n + 3 = O(n^4)$

Proof

We have

$2n^4+3n^3+5n^2+2n+3 \leq 2n^4+3n^4+5n^4+2n^4+3n^4 = (2+3+5+2+3)n^4 = 15n^4$

where $n \geq 1$ ■

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Examples (cont.)

Theorem 1

Prove that $T(n) = a_0 + a_1 n + ... + a_d n^d$ *where* $a_d > 0$ *then* $T(n) = O(n^d)$

Proof

Where $n \geq 1$, we have $1 \leq n \leq n^2 \leq ... \leq n^d$. Hence,

$$a_0 + a_1 n + a_2 n^2 ... + a_d n^d \leq (|a_0| + |a_1| + |a_2| + ... + |a_d|) n^d$$

So we have $f(n) = O(n^d)$ where $c = |a_0| + |a_1| + |a_2| + ... + |a_d|$ and $K = 1$ ■

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Examples (cont.)

Example 2

What is the order-of-growth of

$T(n)$	$O(\cdots)$
$5n^2 + 3n \log n + 2n + 5$	
$20n^3 + 10n \log n + 5$	
$3 \log n + 2$	
2^{n+2}	
$2n + 100 \log n$	

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Exercises

1. If f is in $O(g)$, what can we say about $af + b$?
2. If f_1 and f_2 are in $O(g)$, what can we say about $f_1 + f_2$?
3. If f_1 is in $O(g)$ and f_2 is in $O(h)$, what can we say about $f_1 + f_2$?
4. If f_1 is in $O(g)$ and f_2 is $O(h)$, what can we say about $f_1 \cdot f_2$?

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

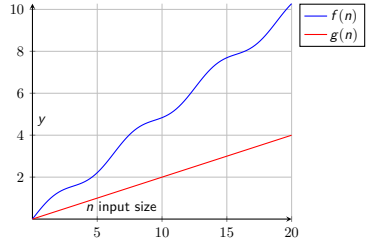
Big Omega Notation

Concept 19

Given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions $\{f(n)\}$, there exist positive constants c and n_0 such that

$$f(n) \geq cg(n), \text{ for all } n \geq n_0 \tag{4}$$

- **Big Omega Notation** represents a lower bound on the growth rate, showing the minimum resources required.



Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

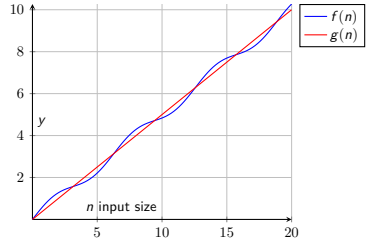
Big Theta Notation

Concept 20

Given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions $\{f(n)\}$, there exist positive constants c_1, c_2, and n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \text{ for all } n \geq n_0 \tag{5}$$

- **Big Theta Notation** represents the growth rate, indicating both the upper and lower limits of an algorithm's complexity.



Mathematical Techniques

- Sum
- Generating Function

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Sum

Closed-form

- Series

$$1 + 2 + 3 + ... + N = \sum_{k=1}^{N} k = \frac{1}{2}(N^2 + N) \tag{6}$$

$$1^2 + 2^2 + 3^2 + ... + N^2 = \sum_{k=1}^{N} k^2 = \frac{1}{6}(2N^3 + 3N^2 + N) \tag{7}$$

$$1^3 + 2^3 + 3^3 + ... + N^3 = \sum_{k=1}^{N} k^3 = \frac{1}{4}(N^4 + 2N^3 + N^2) \tag{8}$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Sum (cont.)

- Geometric series

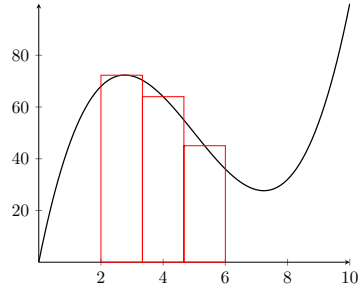
$$1 + r + r^2 + ... + r^n = \sum_{k=0}^{n} r^k = \frac{1 - r^{n+1}}{1 - r} \tag{9}$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Sum (cont.)

Approximation calculation

- Use definite integral



- Harmonic series

$$1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} \approx \int_1^n \frac{1}{x} = \ln n - \ln 1 = \ln n$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Generating Function

Concept 21

Given a sequence of numbers $a_0, a_1, a_2, \ldots, a_k, \ldots$, the function

$$A(z) = \sum_{k \geq 0} a_k z^k$$

is called **ordinary generating function** (OGF) of the series. Denote $\left[z^k\right] A(z)$ be a_k.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Some Generating Functions

Series	Generating function
$1, 1, 1, 1, \ldots, 1, \ldots$	$\frac{1}{1-z} = \sum_{N\geq 0} z^N$
$0, 1, 2, 3, 4, \ldots, N, \ldots$	$\frac{z}{(1-z)^2} = \sum_{N\geq 1} Nz^N$
$1, c, c^2, c^3, \ldots, c^N, \ldots$	$\frac{1}{1-cz} = \sum_{N\geq 0} c^N z^N$
$1, 1, \frac{1}{2!}, \frac{1}{3!}, \frac{1}{4!}, \ldots, \frac{1}{N!}, \ldots$	$e^z = \sum_{N\geq 0} \frac{z^N}{N!}$
$0, 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots, \frac{1}{N}, \ldots$	$\ln \frac{1}{1-z} = \sum_{N\geq 1} \frac{z^N}{N}$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Properties

Given two generating functions $A(z) = \sum_{k \geq 0} a_k z^k$ and $B(z) = \sum_{k \geq 0} b_k z^k$ that represent the sequences $a_0, a_1, \ldots, a_k, \ldots$ and $b_0, b_1, \ldots, b_k, \ldots$, we can perform a number of simple transformations to get generating functions for other sequences

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Properties (cont.)

Operation	Result
right shift	$zA(z) = \sum_{n\geq 1} a_{n-1} z^n$
left shift	$\frac{A(z)-a_0}{z} = \sum_{n\geq 0} a_{n+1} z^n$
differentiation	$A'(z) = \sum_{n\geq 0} (n+1) a_{n+1} z^n$
integration	$\int_0^z A(t)dt = \sum_{n\geq 1} \frac{a_{n-1}}{n} z^n$
scaling	$A(\lambda z) = \sum_{n\geq 0} \lambda^n a_n z^n$
addition	$A(z) + B(z) = \sum_{n\geq 0} (a_n + b_n) z^n$
difference	$(1-z)A(z) = a_0 + \sum_{n\geq 1} (a_n - a_{n-1}) z^n$
convolution	$A(z)B(z) = \sum_{n\geq 0} (\sum_{0\leq k\leq n} a_k b_{n-k}) z^n$
partial sum	$\frac{A(z)}{1-z} = \sum_{n\geq 0} (\sum_{0\leq k\leq n} a_k) z^n$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Example

Example 3

Given a sequence $\{a_n\}_{n\geq 0}$ described by the recurrence

$$a_n = 5a_{n-1} - 6a_{n-2} \qquad n > 1 \text{ where } a_0 = 0 \text{ and } a_1 = 1$$

Proof

Use the generating functio $A(z) = \sum_{n\geq 0} a_n z^n$. Mutitply both sides by z^n and sum on n

$$\begin{aligned} \sum_{n\geq 2} a_n z^n &= 5\sum_{n\geq 2} a_{n-1} z^n - 6\sum_{n\geq 2} a_{n-2} z^n \\ A(z) - a_0 - a_1 z &= 5(A(z)z - a_0 z) - 6(A(z)z^2) \\ A(z) - z &= 5A(z)z - 6A(z)z^2 \end{aligned}$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Example (cont.)

to get the equation

$$A(z) = \frac{z}{1-5z+6z^2} = \frac{z}{(1-3z)(1-2z)} = \frac{1}{1-3z} - \frac{1}{1-2z}$$

Hence, we have $a_n = 3^n - 2^n$ ■

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Exercises

Using generating functions for solving the following recurrence relations.

1. $a_n = a_{n-1} + a_{n-2} \quad n > 1$ where $a_0 = 1$ and $a_1 = 1$.
2. $a_n = -a_{n-1} + 6a_{n-2} \quad n > 1$ where $a_0 = 0$ and $a_1 = 1$
3. $a_n = 11a_{n-2} - 6a_{n-3} \quad n > 2$ where $a_0 = 0$ and $a_1 = a_2 = 1$
4. $a_n = 3a_{n-1} - 4a_{n-2} \quad n > 1$ where $a_0 = 0$ and $a_1 = 1$
5. $a_n = a_{n-1} - a_{n-2} \quad n > 1$ where $a_0 = 0$ and $a_1 = 1$

Algorithm Analysis

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Example

Listing 1: Sum the first n integers

```
sum = 0;
for (i = 0; i < n; i++)
    sum = sum + i;
```

- Assigments: T_1
- Comparisions: T_2
- Arithmetic operation: T_3
- Time complexity $T(n) = T_1 + T_2 + T_3$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Performance in Different Scenarios

How does the algorithm perform under different input conditions?

- **Best-case Analysis**: This considers the fastest the algorithm can possibly finish the task for a particular input.
- **Average-case Analysis**: This takes into account the average time/space complexity considering all possible inputs with equal probability.
- **Worst-case Analysis**: This identifies the slowest the algorithm can take for a specific input, representing the worst possible scenario.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Complexities for Simple Statements

- **Sequential statement**: A sequential statement P consists of two parts, P_1 and P_2. The complexities of P, P_1 and P_2 are denoted by $T(n)$, $T_1(n)$ and $T_2(n)$, respectively.
$$T(n) = T_1(n) + T_2(n) \tag{10}$$
- **Branch statement**: A branch statement P consists two sub-statement P_1 and P_2
$$T(n) = \max(T_1(n), T_2(n)) \tag{11}$$
- **Loop statement**: A loop statement P

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Complexities for Functions

1. Simple function.
2. Function that calls other functions but does not call itself.
3. Recursive function.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Example: Linear Search

```
1  int LinearSearch(int n, int a[], int key)
2  {
3      int i = 0;
4      while (i < n)
5      {
6          if (a[i] == key)
7              return i;
8          i++;
9      }
10     return -1;
11 }
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Example: Linear Search (cont.)

Consider the worst-case

lines	statements	$T(n)$
	assigment	
	loop	
	return	
Overall		

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Example: Bubble Sort

```
1  void BubbleSort(int n, int a[])
2  {
3      int i, j;
4      for (i = 0; i <= n - 2; i++)
5          for (j = n - 1; j >= i + 1; j--)
6              if (a[j] < a[j - 1])
7                  swap(a[j], a[j-1];
8  }
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Example: Bubble Sort (cont.)

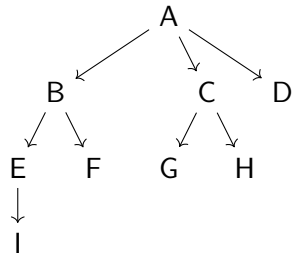
Consider the worst-case

lines	statements	$T(n)$
Overall		

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Function that calls other functions

Consider a function call tree



Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Analysis

We have

1. $T(A) = T(B) + T(C) + T(D) + T_A$
2. $T(B) = T(E) + T(F) + T_B$
3. $T(C) = T(G) + T(H) + T_C$
4. $T(E) = T(I) + T_E$

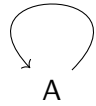
Hence, $T(A) = T(D) + T(F) + +T(G) + T(H) + T(I) + T_A + T_B + T_C + T_E$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Recursive function

Concept 22

A recursive function is a function that calls itself directly or indirectly in order to solve a problem.



Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Step 1: Identify the Recurrence Relation

Determine the recurrence relation that represents the recursive function. A recurrence relation is an equation or inequality that describes the runtime of a recursive function in terms of smaller inputs. You need to consider:

- **Base Case**: The simplest case that doesn't involve recursion. This often has a constant time complexity.
- **Recursive Case**: The part of the function that involves recursive calls. Identify how many times the function calls itself and with what input sizes.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Step 2: Write the Recurrence Relation

Express the time complexity of the recursive function as a recurrence relation. For example:

- A classic recursive function like the Fibonacci sequence can be written as:

$$T(n) = \begin{cases} C_1 & n = 0, 1 \\ T(n-1) + T(n-2) + C_2 & n > 1 \end{cases}$$

- A simple recursive function like a binary search can be written as:

$$T(n) = \begin{cases} C_1 & n = 1 \\ T(n/2) + C_2 & n > 1 \end{cases}$$

- A general recursive function

$$T(n) = \begin{cases} C_1 & n = 0 \\ f(T(0)...T(n-1)) & n > 0 \end{cases}$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Step 3: Solve the Recurrence Relation

There are different methods to solve recurrence relations, such as:

- **Substitution**: Substitute values to determine patterns.
- **Recursion Tree**: Visualize the recursive calls as a tree, and calculate the total cost.
- **Master Theorem**: A formulaic approach to solve recurrences of a specific form, often used with divide-and-conquer algorithms.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Factorial function

```
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return (n * factorial(n - 1));
}
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Analysis

1. We have the recurrence relation

$$T(n) = \begin{cases} C_1 & n = 0 \\ T(n-1) + C_2 & n > 0 \end{cases}$$

2. Solve the recurrence relation using the subsitution method

$$\begin{aligned} T(n) &= T(n-1) + C_2 \\ T(n) &= (T(n-2) + C_2) + C_2 = T(n-2) + 2C_2 \\ T(n) &= (T(n-3) + C_2) + 2C_2 = T(n-3) + 3C_2 \\ &\dots \\ T(n) &= T(n-i) + iC_2 \\ &\dots \\ T(n) &= C_1 + nC_2 \end{aligned}$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Analysis (cont.)

Hence, we have

$$T(n) = C_1 + nC_2 = O(n)$$

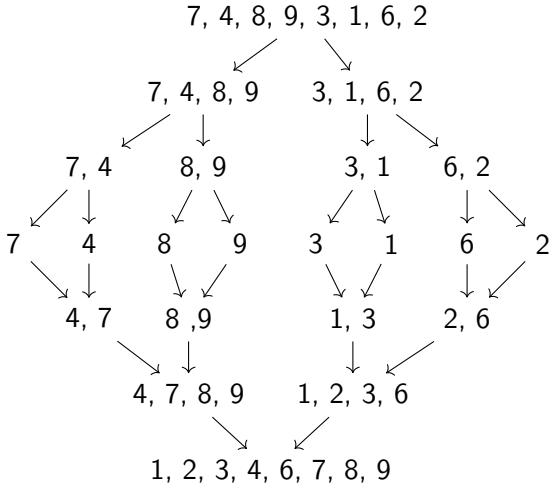
Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Merge Sort Function

```
List MergeSort(List L)
{
    List L1, L2;
    if (L.length <= 1)
        return L;
    else
    {
        Split(L, L1, L2);
        return (Merge(MergeSort(L1), MergeSort(L2)));
    }
}
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Merge Sort Function (cont.)



Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Analysis

1. We have the recurrence relation

$$T(n) = \begin{cases} C_1 & n = 1 \\ 2T\left(\frac{n}{2}\right) + C_2 n & n > 1 \end{cases}$$

2. Solve the recurrence relation using the subsitution method

$$\begin{aligned} T(n) &= 2T\left(\tfrac{n}{2}\right) + C_2 n \\ T(n) &= 2\left[2T\left(\tfrac{n}{4}\right) + C_2\tfrac{n}{2}\right] + C_2 n = 4T\left(\tfrac{n}{4}\right) + 2C_2 n \\ T(n) &= 4\left[2T\left(\tfrac{n}{8}\right) + C_2\tfrac{n}{4}\right] + C_2 n = 8T\left(\tfrac{n}{8}\right) + 3C_2 n \\ &\dots \\ T(n) &= 2^i T\left(\tfrac{n}{2^i}\right) + iC_2 n \end{aligned}$$

The subsituition is ended when $\frac{n}{2^i} = 1$, so $2^i = n, i = \log n$. Hence, we have

$$T(n) = nT(1) + \log n C_2 n = C_1 n + C_2 n \log n = O(n \log n)$$

The Master Theorem

- The Master Theorem
- Multiplicative Function

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Introduction to The Master Theorem

The Master Theorem is a method used to determine the time complexity of **divide and conquer** algorithms. It provides a solution for recurrence relations of the form

$$T(n) = \begin{cases} 1 & n = 1 \\ a \cdot T\left(\frac{n}{b}\right) + f(n) & n > 1 \end{cases} \tag{12}$$

- n is the size of the input.
- $a \geq 1$ is the number of subproblems in the recursion.
- n/b $(b > 1)$ is the size of each subproblem. All subproblems are assumed to have the same size.
- $f(n)$ is the cost of the work done outside the recursive call, which includes the cost of dividing the problem and the cost of merging the solutions.

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Introduction to The Master Theorem (cont.)

The Master Theorem has the constants a, b, and the function $f(n)$

Theorem 2

The theorem has three cases:

1. ***Case 1**: If $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$. This occurs when the recursive work dominates.*
2. ***Case 2**: If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \cdot \log_b n)$. This happens when the recursive work and non-recursive work are balanced.*
3. ***Case 3**: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ for some $c < 1$ and sufficiently large n, then $T(n) = \Theta(f(n))$. This case arises when the non-recursive work dominates.*

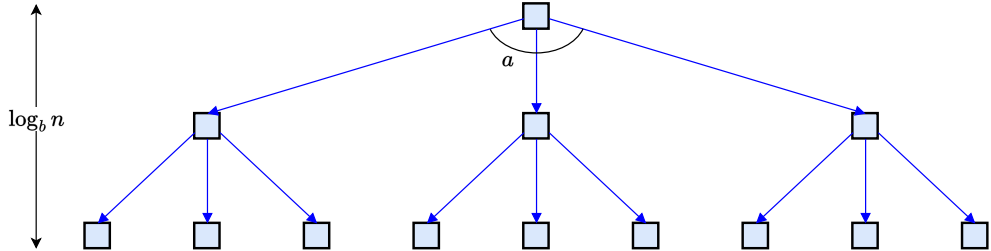
Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Simple Example

First, consider an algorithm with a recurrence of the form

$$T(n) = aT\left(\frac{n}{b}\right)$$

- The tree has a depth of $\log_b n$ and depth i contains a^i nodes. So there are $a^{\log_b n} = n^{\log_b a}$ leaves, and hence the runtime is $\Theta(n^{\log_b a})$.



Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Simple Explanation

Apply a substitution chain

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$
$$T(n) = a\left(aT\left(\frac{n}{b^2}\right) + f\left(\frac{n}{b}\right)\right) + f(n) = a^2 T\left(\frac{n}{b^2}\right) + af\left(\frac{n}{b}\right) + f(n)$$
$$\ldots$$

We have

$$T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j f\left(\frac{n}{b^j}\right)$$

Assume that $n = b^k$ or $k = \log_b n$, the substitution process is ended when $i = k$.

$$T\left(\frac{n}{b^i}\right) = T\left(\frac{n}{b^k}\right) = T\left(\frac{b^k}{b^k}\right) = T(1) = 1$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Simple Explanation (cont.)

Finally, we have

$$\begin{aligned} T(n) &= a^k + \sum_{j=0}^{k-1} a^j f\left(b^{k-j}\right) \\ &= n^{\log_b a} + \sum_{j=0}^{k-1} a^j f\left(b^{k-j}\right) \\ &= T_1(n) + T_2(n) \end{aligned} \tag{13}$$

where $T_1(n) = n^{\log_b a}$ is the recursive work and $T_2(n) = \sum_{j=0}^{k-1} a^j f\left(b^{k-j}\right)$ is the non-recursive work. The complexity can be

$$T(n) = \max(T_1, T_2)$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Multiplicative Function

Concept 23

A function $f(n)$ is a **multiplicative function** if $f(m.n) = f(m).f(n)\ \forall\, m, n$

- The function $f(n) = n^k$ is a muliplicative function
- The fucntion $f(n) = \log n$ is not a muliplicative function

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Multiplicative Function $f(n)$

Assume that $f(n)$ is a multiplicative function thì ta có đó $f(b^i) = f(b.b...b) = f(b).d(b)...d(b) = f(b)^i$. We have

$$T_2(n) = \sum_{j=0}^{k-1} a^j f\left(b^{k-j}\right)$$

$$T_2(n) = f(b)^k \sum_{j=0}^{k-1} \left(\frac{a}{f(b)}\right)^j$$

$$T_2(n) = f(b)^k \frac{\left(\frac{a}{f(b)}\right)^k - 1}{\frac{a}{f(b)} - 1}$$

$$T_2(n) = \frac{a^k - f(b)^k}{a - f(b)} f(b) \tag{14}$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Multiplicative Function $f(n)$ (cont.)

Three cases

- **Case 1**: $a > f(b)$. then $a^k > f(b)^k$

$$T(n) = \max(T_1, T_2) = T_1(n) = O(a^k) = O(n^{\log_b a})$$

- **Case 2**: $a < f(b)$. then $a^k < f(b)^k$

$$T(n) = \max(T_1, T_2) = T_2(n) = O(f(b)^k) = O(n^{\log_b f(b)})$$

- **Case 3**: $a = f(b)$.

$$T_2(n) = f(b)^k \sum_{j=0}^{k-1} \left(\frac{a}{f(b)}\right)^j = a^k \sum_{j=0}^{k-1} 1 = a^k k$$

$$T(n) = O\left(n^{\log_b a} \log_b n\right)$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Examples

Example 4

Consider a recursive function

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

Proof

- We have $a = 4, b = 2, f(n) = n$
- and $f(b) = f(2) = 2 < 4 = a$
- hence,

$$T(n) = O\left(n^{\log_b a}\right) = O\left(n^2\right)$$

■

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Examples (cont.)

Example 5

Consider a recursive function

$$T(n) = 4T\left(\frac{n}{2}\right) + n^2$$

Proof

- We have $a = 4, b = 2, f(n) = n^2$
- and $f(b) = f(2) = 4 = a$
- hence,

$$T(n) = O\left(n^{\log_b a} \log_b n\right) = O\left(n^2 \log n\right)$$

■

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Examples (cont.)

Example 6

Consider a recursive function

$$T(n) = 4T\left(\frac{n}{2}\right) + n^3$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Examples (cont.)

Proof

- We have $a = 4, b = 2, f(n) = n^3$
- and $f(b) = f(2) = 8 > 4 = a$
- hence,

$$T(n) = O\left(n^{\log_b d(b)}\right) = O\left(n^3\right)$$

■

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Examples (cont.)

Example 7

Consider a recursive function

$$T(n) = 2T\left(\frac{n}{2}\right) + n \log n$$

Proof

- The recursive work

$$T_1(n) = n^{\log_b a} = n^{\log_2 2} = n$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Examples (cont.)

- The non-recursive work

$$T_2(n) = \sum_{j=0}^{k-1} a^j d(b^{k-j}) = \sum_{j=0}^{k-1} 2^j 2^{k-j} \log 2^{k-j}$$
$$T_2(n) = 2^k \sum_{j=0}^{k-1} (k-j) = 2^k \frac{k(k+1)}{2}$$
$$T_2(n) \sim 2^k k^2$$

- In this case, we have $n = 2^k$ và $k = \log_2 n$, so

$$T_2(n) \sim n \log^2 n$$

- In summary, we have

$$T(n) = O\left(n \log^2 n\right)$$

■

Workshop

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Quiz

1. What is analysis of algorithms?

. . . . . .
. . . . . .
. . . . . .

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Exercises

Find the complexities of

1.
```
for(i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        b[i][j] += c;

for(i = 0; i < n; i++)
    for (j = i+1; j < n; j++)
        b[i][j] -= c;
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Exercises (cont.)

2.

```
for(i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        a[i][j] = b[i][j]+c[i][j];
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Exercises (cont.)

3.

```
for(i = 0; i<n; i++)
    for (j = 0; j<n; j++)
        for(k=a[i][j]=0; k<n; k++)
            a[i][j]+=b[i][k]+c[k][j];
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Exercises (cont.)

4. The recurrence relations

$$T(n) = T\left(\frac{n}{2}\right) + 1$$

$$T(n) = 2T\left(\frac{n}{2}\right) + \log n$$

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Exercises (cont.)

5.

```
void HanoiTower(int n, int a, int b, int c) {
    if (n > 0) {
        HanoiTower(n - 1, a, c, b);
        cout << "move from " << a << " to " << c << endl;
        HanoiTower(n - 1, b, a, c);
    }
}
```

Introduction
Types Of Problem
Asymptotic
Big O Notation
Big Omega Notation
Big Theta Notation
Mathematical Techniques
Sum
Generating Function
Algorithm Analysis
The Master Theorem
The Master Theorem
Multiplicative Function
Workshop

Exercises (cont.)

6.

```
void Permute(int k, int n, int a[]) {
    if (k == 0) {
        for (int i = 0; i < n; i++)
            cout << a[i] << " ";
        cout << endl;
    }
    else {
        for (int i = 0; i < k; i++) {
            swap(a[i], a[k - 1]);
            Permute(k - 1, n, a);
            swap(a[i], a[k - 1]);
        }
    }
}
```

References

Cormen, T. H. (2009).
Introduction to algorithms.
MIT press.

Sedgewick, R. (2002).
Algorithms in Java, Parts 1-4, volume 1.
Addison-Wesley Professional.

Walls and Mirrors (2014).
Data Abstraction And Problem Solving with C++.
Pearson.