Discrete Mathematics

ALGORITHMS & INTEGERS

The Fundamentals

FPT Education

FPT UNIVERSITY

Department of Mathematics

Võ Văn Nam, Ph.D.

Competency Goals

1. Evaluate whether a given procedure qualifies as an algorithm; interpret and work through basic algorithms in pseudocode, including searching, sorting, and greedy algorithms.

2. Calculate the big-O and big-Theta estimates for a given function; articulate the computational complexity of an algorithm.

3. Execute integer division and modular arithmetic operations; utilize congruence in generating pseudorandom numbers and in cryptographic applications.

4. Convert integers to their base b expansion; perform binary operations and execute modular exponentiation on integers.

5. Determine if a number is prime and perform prime factorization; compute the greatest common divisor (GCD) and least common multiple (LCM) of integers.

Table of Contents

1. Algorithms
2. The Growth of Functions
3. Complexity of Algorithms
4. Integers and Division
5. Primes and Greatest Common Divisors
6. Integers and Algorithms

What's next?

1. Algorithms
2. The Growth of Functions
3. Complexity of Algorithms
4. Integers and Division
5. Primes and Greatest Common Divisors
6. Integers and Algorithms

Algorithms

An **algorithm** is a finite set of precise instructions for performing a computation or for solving a problem.

Example. Describe an algorithm to solve quadratic equations.
Input: a, b, c: integers (coefficients)
Output: Solutions if they exists.

Algorithm:

- **Step 1**. If $a = 0$ then Print (This is not a quadratic equation).
- **Step 2**. Compute $\Delta = b^2 - 4ac$.
- **Step 3**. If $\Delta < 0$ then Print (No solution).
- **Step 4**. If $\Delta = 0$ then compute $x = -b/2a$.
- **Step 5**. If $\Delta > 0$ then compute $x_1 = (-b + \sqrt{\Delta})/(2a), \quad x_2 = (-b - \sqrt{\Delta})/(2a)$.

Properties of Algorithms

- **Correctness** An algorithm must produce the correct output for all possible valid inputs. Correctness ensures that the algorithm solves the problem it was designed to address.
- **Efficiency**
  - *Time Complexity:* Measures the amount of time an algorithm takes to run as a function of the input size. It's usually expressed using Big-O notation.
  - *Space Complexity:* Measures the amount of memory an algorithm uses during execution as a function of the input size. This is also often expressed using Big-O notation.
- **Finiteness** An algorithm must have a finite number of steps and terminate after a finite amount of time. This ensures that the algorithm does not run indefinitely and provides a result in a reasonable time frame.

Properties of Algorithms (cont')

- **Definiteness** Each step of an algorithm must be precisely defined and unambiguous. This means that the operations and the order of operations are clear, leaving no room for interpretation.

- **Input** An algorithm should have well-defined inputs. It may take zero or more inputs, which are the initial data provided before the algorithm begins execution.

- **Output** An algorithm must have at least one output, which is the result produced after executing the algorithm on the input data.

- **Generality** An algorithm should be applicable to a class of problems rather than solving a specific instance of a problem. This means it should be general enough to handle all inputs that fit within its problem domain.

Properties of Algorithms (cont')

- **Determinism** A deterministic algorithm always produces the same output given the same input. The behavior of the algorithm is entirely predictable, with no randomness involved in its execution.

- **Scalability** Scalability refers to the algorithm's ability to handle larger inputs efficiently. A scalable algorithm maintains its performance as the input size grows.

- **Robustness** A robust algorithm can handle unexpected situations, such as invalid inputs or hardware failures, without crashing or producing incorrect results. Robust algorithms are designed to gracefully manage errors.

Some common algorithms

1. Find maximum/minimum element of a finite sequence.
2. Searching algorithms:
   - Linear search algorithm
   - Binary search algorithm
3. Sorting algorithms:
   - Bubble sort algorithm
   - Insertion sort algorithm.
4. Greedy change-making algorithm.

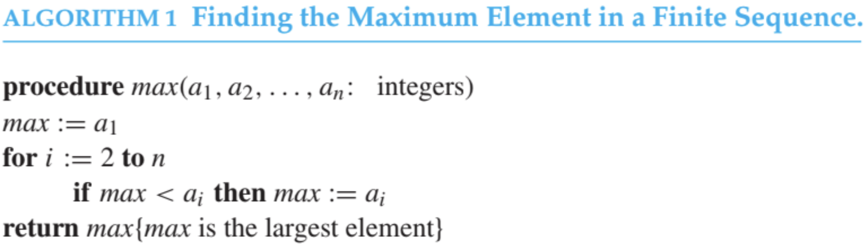
Finding Maximum Element

Input: Sequence of integers $a_1, a_2, \ldots, a_n$.
Output: The maximum number of the sequence.

Algorithm:

- **Step 1**. Set the temporary maximum be the first element.
- **Step 2**. Compare the temporary maximum to the next element, if this element is larger then set the temporary maximum to be this integer.
- **Step 3**. Repeat **Step 2** if there are more integers in the sequence.
- **Step 4**. Stop the algorithm when there are no integers left. The temporary maximum at this point is the maximum of the sequence.

Pseudo Code



ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

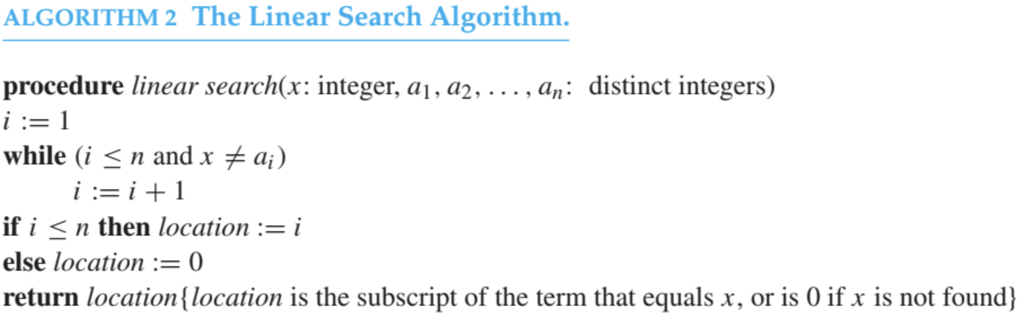
```
procedure $max(a_1, a_2, \ldots, a_n$:  integers)
$max := a_1$
for $i := 2$ to $n$
        if $max < a_i$ then $max := a_i$
return $max$\{$max$ is the largest element\}
```

Linear Search

Input: A sequence of distinct integers $a_1, a_2, \ldots, a_n$, and an integer x.
Output: The location of x in the sequence (is 0 if x is not in the sequence).

Algorithm: Compare x successively to each term of the sequence until a match is found.

Linear Search: Pseudo Code



ALGORITHM 2 The Linear Search Algorithm.

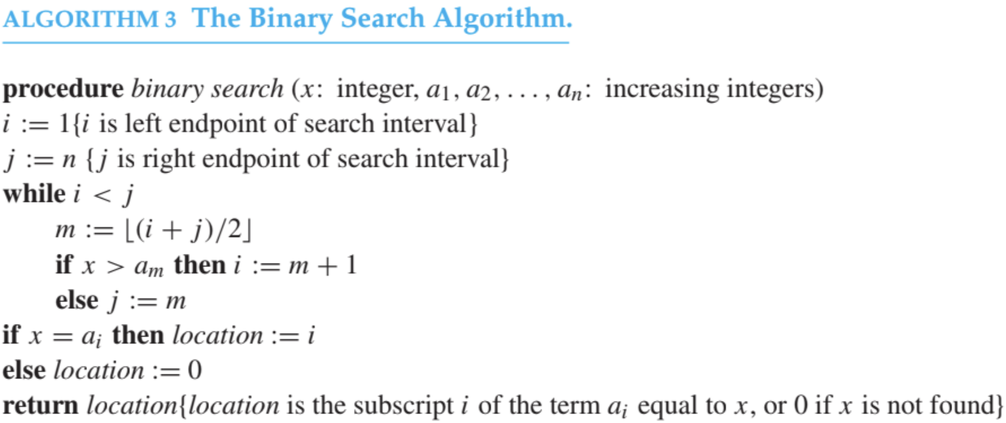
```
**procedure** *linear search*($x$: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
**while** ($i \leq n$ and $x \neq a_i$)
        $i := i + 1$
**if** $i \leq n$ **then** *location* $:= i$
**else** *location* $:= 0$
**return** *location*{*location* is the subscript of the term that equals $x$, or is 0 if $x$ is not found}
```

Binary Search

Input: An increasing sequence of integers $a_1 < a_2 < \cdots < a_n$ and an integer x.
Output: The location of x in the sequence (is 0 if x is not in the sequence).

Algorithm: Compare x to the element at the middle of the list, then restrict the search to either the sublist on the left or the sublist on the right.

Binary Search: Pseudo Code



ALGORITHM 3 The Binary Search Algorithm.

```
procedure binary search ($x$: integer, $a_1, a_2, \ldots, a_n$: increasing integers)
$i := 1$ {$i$ is left endpoint of search interval}
$j := n$ {$j$ is right endpoint of search interval}
while $i < j$
    $m := \lfloor (i + j)/2 \rfloor$
    if $x > a_m$ then $i := m + 1$
    else $j := m$
if $x = a_i$ then location $:= i$
else location $:= 0$
return location {location is the subscript $i$ of the term $a_i$ equal to $x$, or 0 if $x$ is not found}
```

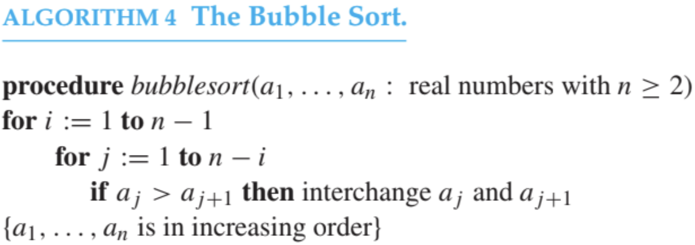
Bubble Sort

Input: A sequence of integers $a_1, a_2, \ldots, a_n$
Output: The sequence in the increasing order.

Algorithm:

1. Successively comparing two consecutive elements of the list to push the largest element to the bottom of the list.
2. Repeat the above step for the first $n - 1$ elements of the list.

Bubble Sort: Pseudo Code

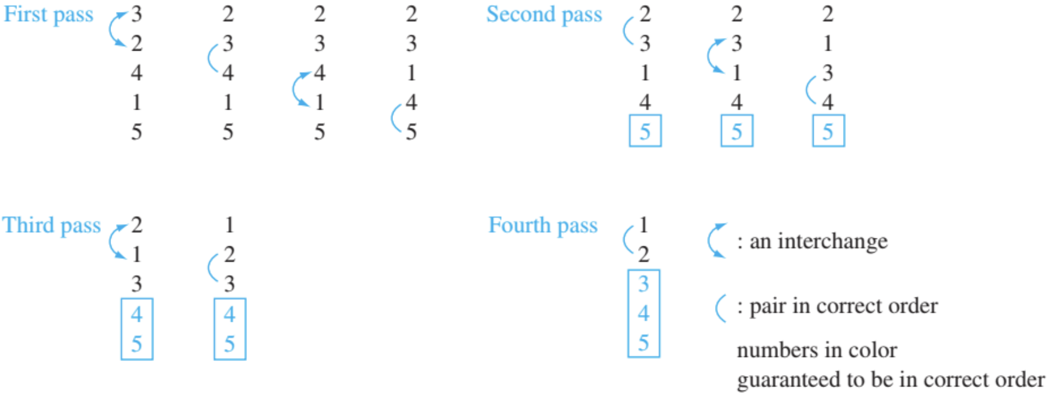


ALGORITHM 4 The Bubble Sort.

```
procedure bubblesort($a_1, \ldots, a_n$ : real numbers with $n \geq 2$)
for $i := 1$ to $n - 1$
    for $j := 1$ to $n - i$
        if $a_j > a_{j+1}$ then interchange $a_j$ and $a_{j+1}$
{$a_1, \ldots, a_n$ is in increasing order}
```

Bubble Sort: Example

Use the bubble sort to put 3, 2, 4, 1, 5 into increasing order.



: an interchange

: pair in correct order

numbers in color
guaranteed to be in correct order

Insertion Sort

Input: Sequence of integers $a_1, a_2, \ldots, a_n$
Output: The sequence in the increasing order.

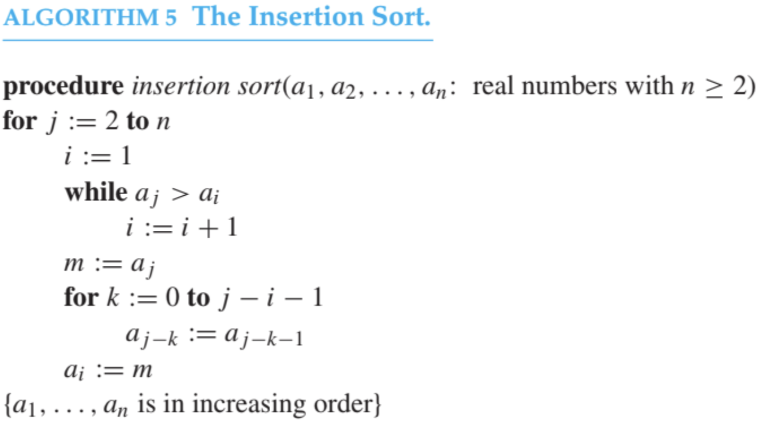
Algorithm:

1. Sort the first two elements of the list.
2. Insert the third element to the list of the first two elements to get a list of 3 elements of increasing order.
3. Insert the fourth element to the list of the first three elements to get a list of 4 elements of increasing order.

$\vdots$

n. Insert the nth element to the list of the first $n-1$ elements to get a list of increasing order.

Insertion Sort: Pseudo Code



ALGORITHM 5 The Insertion Sort.

```
procedure insertion sort($a_1, a_2, \ldots, a_n$: real numbers with $n \geq 2$)
for $j := 2$ to $n$
        $i := 1$
        while $a_j > a_i$
                $i := i + 1$
        $m := a_j$
        for $k := 0$ to $j - i - 1$
                $a_{j-k} := a_{j-k-1}$
        $a_i := m$
{$a_1, \ldots, a_n$ is in increasing order}
```

Greedy Algorithms

Greedy algorithms are a class of algorithms that make a series of choices, each of which looks the most immediately beneficial, in an attempt to find an optimal solution to a problem. They follow a problem-solving heuristic that selects the best local choice at each step with the hope of finding a global optimum.

Common Examples of Greedy Algorithms:

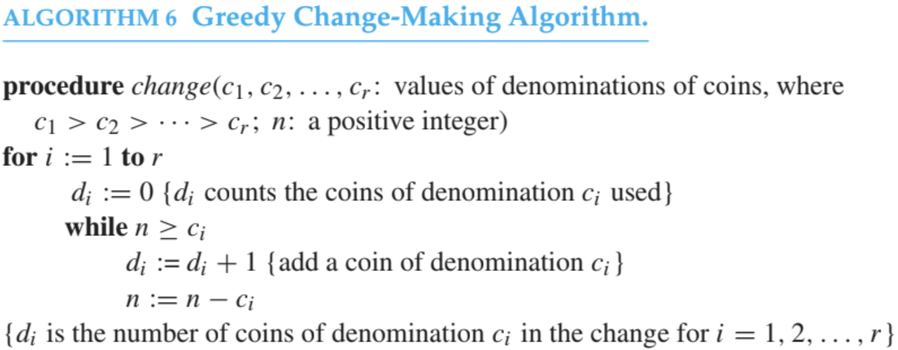
- Fractional Knapsack Problem
- Huffman Coding
- Prim's and Kruskal's Algorithms for Minimum Spanning Tree
- Activity Selection Problem
- Dijkstra's Algorithm for Shortest Path

Greedy Change-Making Algorithm

Input: n cents
Output: The least number of coins using quarters $(= 25$ cents), dimes $(= 10$ cents), nickles $(= 5$ cents) and $(= 1$ cent).

- Greedy change-making algorithms are usually used to solve optimization problems: Finding out a solution to the given problem that either minimizes or maximizes the value of some parameters.
- Selecting the best choice at each step, instead of considering all sequences of steps that may lead to an optimal solution.

Greedy Change-Making Algorithm: Pseudo Code



ALGORITHM 6 Greedy Change-Making Algorithm.

```
procedure change($c_1, c_2, \ldots, c_r$: values of denominations of coins, where
  $c_1 > c_2 > \cdots > c_r$; $n$: a positive integer)
for $i := 1$ to $r$
    $d_i := 0$ {$d_i$ counts the coins of denomination $c_i$ used}
    while $n \geq c_i$
        $d_i := d_i + 1$ {add a coin of denomination $c_i$}
        $n := n - c_i$
{$d_i$ is the number of coins of denomination $c_i$ in the change for $i = 1, 2, \ldots, r$}
```

What's next?

1. Algorithms
2. The Growth of Functions
3. Complexity of Algorithms
4. Integers and Division
5. Primes and Greatest Common Divisors
6. Integers and Algorithms

The Growth of Functions

In calculus, we learned following basic functions listed in the **increasing** order of their **complexity**:

$$1, \ \log n, \ n^k, \ a^n, \ n!$$

where $k > 0$ and $a > 1$. It means that

$$\lim_{n\to+\infty} \frac{\log n}{1} = +\infty, \ \lim_{n\to+\infty} \frac{n^k}{\log n} = +\infty, \ \lim_{n\to+\infty} \frac{a^n}{n^k} = +\infty, \ \lim_{n\to+\infty} \frac{n!}{a^n} = +\infty.$$

Ordering of Basic Functions by Growth

$$1, \ \log n, \ \sqrt[3]{n}, \ \sqrt{n}, \ n, \ n\log n, \ n^2, \ n^3, \ 2^n, \ 3^n, \ n!.$$

Question. Estimate the complexity (the growth) of functions like

$$f(n) = \frac{(n+2)(n\log n + n!)}{3^n + (\log n)^2}$$

Big-O

The function $f(x)$ is called **big-O** of $g(x)$, write $f(x)$ is $O(g(x))$, if there exists a constant C and k such that

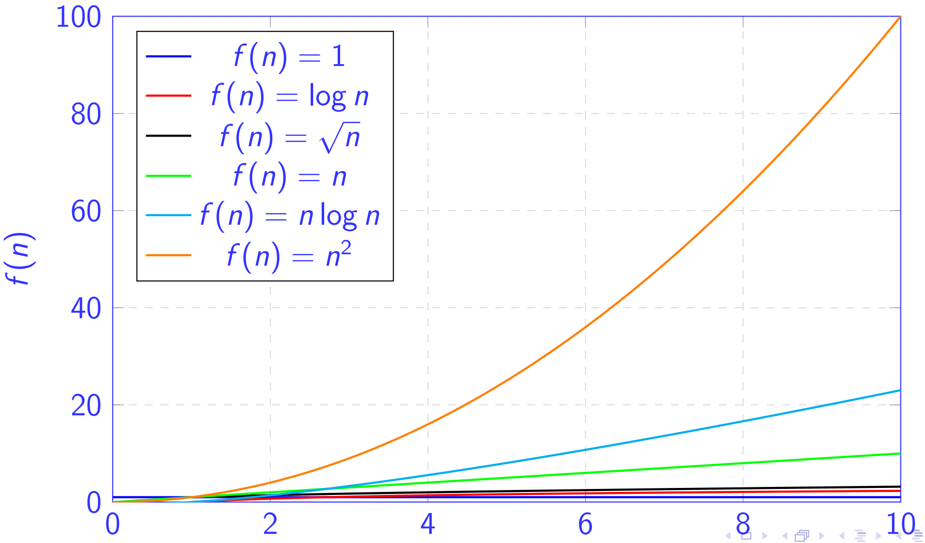
$$|f(x)| \leq C|g(x)|$$

whenever $x > k$.

Note.

1. The definition that $f(x)$ is $O(g(x))$ says that $f(x)$ grows slower that some fixed multiple of $g(x)$ as x grows without bound.
2. The fact that $f(x)$ is $O(g(x))$ is sometimes written $f(x) = O(g(x))$. However, the equals sign in this notation does not represent a genuine equality.
3. However, it is acceptable to write $f(x) \in O(g(x))$ because $O(g(x))$ represents the set of functions that are $O(g(x))$.

The graph of some basic functions



Examples

Example 1. Show that $f(x) = 2x^3 + 3x$ is $O(x^3)$.
Solution. With $C = 3$ and $k = 2$, we have $x^3 > 3x$ whenever $x > 2$ which implies

$$|2x^3 + 3x| = 2x^3 + 3x < 3x^3 = 3|x^3|$$

whenever $x > 2$.
Example 2. Show that $f(x) = 5x^2 - 10000x + 7$ is $O(x^2)$.
Solution. We have to be a little more careful about negative values here because of the $-10000x$ term, but as long as we take $k \geq 2000$, we will not have any negative values since the $5x^2$ term is larger there. We have

$$\begin{aligned} |5x^2 - 10000x + 7| &= 5x^2 - 10000x + 7 \\ &\leq 5x^2 + 7x^2 \\ &= |12x^2| = 12|x^2|. \end{aligned}$$

Big-O for Polynomials

Theorem

Any degree-n polynomial, $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0$ is $O(x^n)$.

Example.

1. The function $p(x) = 2x^3 + 10x$ is not in $O(x^2)$.
2. The function $f(x) = 2x^3 + 10x$ is $O(x^4)$.

Question. Find the smallest integer n such that

1. $x^3 + x^5 \log x$ is $O(x^n)$.
2. $x^5 + x^3(\log x)^4$ is $O(x^n)$.

Properties of Big-O

Theorem

Assume that $f(x)$ is $O(F(x))$ and $g(x)$ is $O(G(x))$. Then,

1. $cf(x)$ is $O(F(x))$ where c is a constant.
2. $f(x) + g(x)$ is $O(\max\{|F(x)|, |G(x)|\})$.
3. $f(x) \cdot g(x)$ is $O(F(x) \cdot G(x))$.

Theorem

Let f, g, h be functions where $f(x)$ is $O(g(x))$ and $g(x)$ is $O(h(x))$. Then, $f(x)$ is $O(h(x))$.

Limit Comparison Test

To check if a function $f(n)$ is in Big O of another function $g(n)$, we can use Limit Comparison Test.
Given two functions $f(n)$ and $g(n)$, we compute the limit:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)}$$

There are three possible outcomes:

1. If the limit is a positive finite constant $c > 0$: This implies that $f(n)$ and $g(n)$ grow at the same rate, and we can say $f(n) = O(g(n))$.
2. If the limit is 0: This means that $f(n)$ grows slower than $g(n)$, so $f(n) = O(g(n))$.
3. If the limit is infinity (∞): This means that $f(n)$ grows faster than $g(n)$, and $f(n)$ is not $O(g(n))$. However, we could say that $g(n) = O(f(n))$.

L'Hospital Rule

L'Hospital's Rule is a method used in calculus to find limits of indeterminate forms, specifically of the types $\frac{0}{0}$ and $\frac{\infty}{\infty}$.

Suppose you want to evaluate the limit:

$$\lim_{x \to c} \frac{f(x)}{g(x)}$$

If this limit gives an indeterminate form of $\frac{0}{0}$ or $\frac{\infty}{\infty}$, and if the derivatives $f'(x)$ and $g'(x)$ exist and $g'(x) \neq 0$ near c, then:

$$\lim_{x \to c} \frac{f(x)}{g(x)} = \lim_{x \to c} \frac{f'(x)}{g'(x)}$$

provided the limit on the right-hand side exists or is $\pm\infty$.

Questions

Question 1. Give a big-O estimate that is simple and effective for each of the functions:

a) $n^2 \log n + 3n^2 + 5$

b) $(n^3 + 2^n)(n! + n^2 5^n)$

Question 2. Which of the following functions is $O(\log n)$?
Choose the best answer.

A. $g(n) = 5n + 2015$

B. $g(n) = n^2 - n$

C. $g(n) = n. \log n + 7$

D. $g(n) = 3 \log n + 2$

Big-theta

Big-O estimates only give upper-bounds; to give sharper bounds, we use big-theta notation.

The function $f(x)$ is called **big-theta** of $g(x)$, write "$f(x)$ is $\Theta(g(x))$", if $f(x)$ is $O(g(x))$ and $g(x)$ is $O(f(x))$. In the other words, $f(x)$ is $\Theta(g(x))$ if there are constants $C_1, C_2 > 0$ such that

$$C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$$

whenever $x > k$.

Example. Show that $3x^2 + 8x\log x$ is $\Theta(x^2)$.
Solution. Since $0 \leq 8x\log x \leq 8x^2$, it follows that $3x^2 + 8x\log x \leq 11x^2$ for $x > 1$. Consequently, $3x^2 + 8x\log x$ is $O(x^2)$ and x^2 is $O(3x^2 + 8x\log x)$. Hence, $3x^2 + 8x\log x$ is $\Theta(x^2)$.

Questions

1. Show that $f(x) = 2x^3 + x^2 + 3$ is $\Theta(x^3)$.
2. Is the function $f(x) = x^2 \log x + 3x + 1$ big-theta of x^3?
3. Show that $f(x) = \left\lfloor \frac{x}{2} \right\rfloor$ is $\Theta(x)$.

What's next?

1. Algorithms
2. The Growth of Functions
3. **Complexity of Algorithms**
4. Integers and Division
5. Primes and Greatest Common Divisors
6. Integers and Algorithms

Complexity of Algorithms

- **Space complexity**: Computer memory required to run the algorithm.
- **Time complexity**: Time required to run the algorithm. Time complexity can be expressed in terms of the number of operations used by the algorithm. Those operations can be comparisons or basic arithmetic operations.

In this lecture, we analyze times complexity.

Input Complexities Ordered from Smallest to Largest

1. Constant Complexity: $O(1)$.
2. Logarithmic Complexity: $O(\log n)$.
3. Radical complexity: $O(\sqrt{n})$.
4. Linear Complexity: $O(n)$.
5. Linearithmic Complexity: $O(n \log n)$.
6. Quadratic complexity: $O(n^2)$.
7. Cubic complexity: $O(n^3)$.
8. Exponential complexity: $O(b^n)$ where $b > 1$.
9. Factorial complexity: $O(n!)$.

Examples

1. $O(1)$: Constant time - Accessing an element in an array by index.
2. $O(\log n)$ Logarithmic time - Binary search in a sorted array.
3. $O(n)$: Linear time - Traversing an array or a list.
4. $O(n \log n)$: Linearithmic time - Efficient sorting algorithms like mergesort and heapsort.
5. $O(n^2)$: Quadratic time - Bubble sort or insertion sort in the worst case.
6. $O(2^n)$: Exponential time - Recursive algorithms for solving the Tower of Hanoi.
7. $O(n!)$: Factorial time - Generating all permutations of a set.

Question

What is the best Big-O complexity in terms of the number of comparisons for the following algorithm?

```
procedure ABC($a_1, a_2, \ldots, a_n$: integers)
for $i = 2$ to $n$
    for $k = 1$ to $i - 1$
        if $a_k = a_i$ then
            print($k$)
            break
```

Choose the correct answer.

A. $O(n^2)$

B. $O(n \log n)$

C. $O(\log n)$

D. $O(n)$

What's next?

1. Algorithms
2. The Growth of Functions
3. Complexity of Algorithms
4. **Integers and Division**
5. Primes and Greatest Common Divisors
6. Integers and Algorithms

Integers and Division

Let a, b be integers with $a \neq 0$. We say the integer a divides b if there is an integer m such that $b = ma$.

Note. If a divides b, we also write:

- b is divisible by a.
- b is a multiple of a.
- a is a factor of b.
- $a|b$.

Theorem

Let a, b, c be integers. Then

- If $a|b$ and $a|c$ then $a|(b+c)$.
- If $a|b$ then $a|bc$ for all c.
- If $a|b$ and $b|c$ then $a|c$.

The Division Algorithm

- Let a be an integer and d be a positive integer. Then, there are unique integers q and r with $0 \leq r < d$ such that $a = qd + r$.
- In this division algorithm, a is called the dividend, d is the divisor, q is the quotient and r is the remainder. We write,

$$q = a \text{ div } d, \quad r = a \text{ mod } d.$$

Question. Find the remainder and the quotient of the division:

1. -23 is divided by 7.
2. -125 is divided by 11.

Modular Arithmetic

Let a, b be integers and m be a positive integer. We say a is **congruent** to b **modulo** m is the have the same remainders when being divided by m. We use notation $a \equiv b \mod m$. If they are not congruent, we write $a \not\equiv b \mod m$.

Properties

1. $a \equiv b \mod m \Leftrightarrow a - b \equiv 0 \mod m \Leftrightarrow a = b + km$ for some integer k.
2. If $a \equiv b \mod m$ and $c \equiv d \mod m$ then $a + c \equiv b + d \mod m$ and $ac \equiv bd \mod m$.

FPT Education
FPT UNIVERSITY

Applications of Congruences

Pseudorandom numbers can be generated using **Linear congruential method**:

- x_0 is given, and
- $x_n = ax_{n-1} + c \mod m, \; n = 2, 3, 4, \ldots$
  where m is called **modulus**, a is the **multiplier**, c is the **increment** and x_0 is the **seed**.

Cryptography

0	1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	D	E	F	G	H	I	J	K	L	M
13	14	15	16	17	18	19	20	21	22	23	24	25
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Caesar's cipher.

$$f(p) = p + 3 \mod 26.$$

What's next?

1. Algorithms
2. The Growth of Functions
3. Complexity of Algorithms
4. Integers and Division
5. Primes and Greatest Common Divisors
6. Integers and Algorithms

Primes and Greatest Common Divisors

- A positive integer p greater than 1 is called a **prime number** if the only prime factors of p are 1 and p.
- An integer greater than 1 that is not prime is called **composite number**.

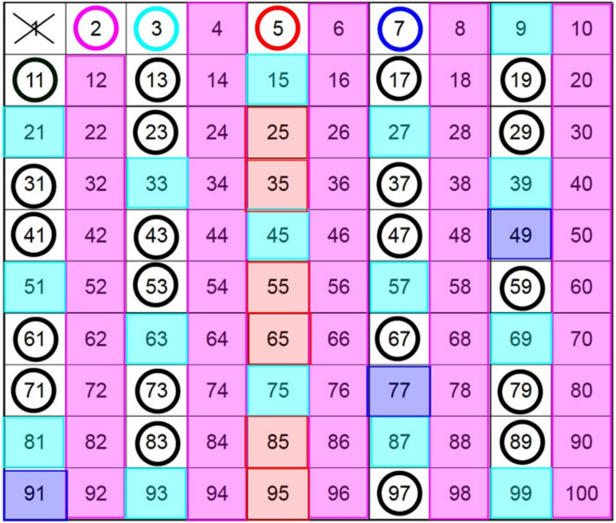
The Fundamental Theorem of Arithmetic

Any integer greater than 1 can be written uniquely as a product of powers of distinct primes.

Theorem

There are infinitely many primes.

Sieve of Eratosthenes



~~1~~	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Greatest Common Divisor

Problem. At the school sports day, you need to divide participants into groups. You have 84 students and 126 teachers, and you want to organize them into the largest possible equal-sized groups where each group has the same number of students and teachers. *How many groups are possible?*

Let a and b be two integers, not both 0. The greatest integer d that is a divisor of both a and b is called **greatest common divisor** of a and b, denoted by $\gcd(a, b)$.

Let a and b be two positive integers. The smallest positive integer d that is divisible by both a and b is called the **least common multiple** of a and b, denoted by $\operatorname{lcm}(a, b)$.

Find gcd and lcm

To find gcd and lcm of a and b, we write a, b as products of powers of distinct primes:

$$\begin{aligned} a &= p_1^{a_1} p_2^{a_2} \cdots p_n^{a_n}, \\ b &= p_1^{b_1} p_2^{b_2} \cdots p_n^{b_n}. \end{aligned}$$

Then, we obtain

- $\gcd(a, b) = p_1^{\min(a_1,b_1)} p_2^{\min(a_2,b_2)} \dots p_n^{\min(a_n,b_n)}$.
- $\operatorname{lcm}(a, b) = p_1^{\max(a_1,b_1)} p_2^{\max(a_2,b_2)} \dots p_n^{\max(a_n,b_n)}$.

Pairwise Relatively Prime

Theorem

Let a, b be positive integers. Then,

$$ab = \gcd(a, b) \cdot \operatorname{lcm}(a, b).$$

1. Two integers a, b are called **relatively prime** if $\gcd(a, b) = 1$.
2. A set of integers are called **pairwise relatively prime** if any two integers in the set are relatively prime.

Question. Which sets are pairwise relatively prime?

1. $\{13,\ 24,\ 49\}$.
2. $\{14,\ 23,\ 35,\ 61\}$.

What's next?

1. Algorithms
2. The Growth of Functions
3. Complexity of Algorithms
4. Integers and Division
5. Primes and Greatest Common Divisors
6. **Integers and Algorithms**

Integers and Algorithms

Representations of Integers

Let b be an integer greater than 1 and n be a positive integer. Then, n can be expressed uniquely in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_0$$

where $a_k, \ldots, a_1, a_0$ are nonnegative integers and less than b. This representation is called **base** b **expansion of** n, and denoted by $n = (a_k a_{k-1} \ldots a_0)_b$

Some Important Representations

1. **Binary** expansion: 0, 1.
2. **Octal** expansion: $0,\ 1,\ 2, \ldots,\ 7$.
3. **Hexadecimal** expansion: $0,\ 1,\ 2, \ldots,\ 9,\ A,\ B,\ C,\ D,\ E,\ F$.

Some representations of the Integers 0 through 15

Decimal	0	1	2	3	4	5	6	7
Hexadecimal	0	1	2	3	4	5	6	7
Octal	0	1	2	3	4	5	6	7
Binary	0	1	10	11	100	101	110	111
Decimal	8	9	10	11	12	13	14	15
Hexadecimal	8	9	A	B	C	D	E	F
Octal	10	11	12	13	14	15	16	17
Binary	1000	1001	1010	1011	1100	1101	1110	1111

Algorithms for Integer Operations

Let a and b be in the binary expansions.

$$a = (a_{n-1}a_{n-2}\dots a_0)_2, \quad b = (b_{n-1}b_{n-2}\dots b_0)_2.$$

Addition Algorithm

```
Procedure Addition $(a, b)$
$c := 0$
for $j := 0$ to $n-1$
    $d := \lfloor (a_j + b_j + c)/2 \rfloor$
    $s_j := a_j + b_j + c - 2d$
    $c := d$
$s_n := c$
return $(s_0, s_1, \dots, s_n)$
```

Multiplication Algorithm

```
Procedure Multiplication $(a, b)$
for $j := 0$ to $n-1$
    if $b_j = 1$ then $c_j := a$ shifted $j$ places
    else $c_j := 0$
$p := 0$
for $j := 0$ to $n-1$
    $p := p + c_j$
return $p$
```

Euclidean Algorithm

Theorem

Let $a > b$ be positive integers. Put $r = a \mod b$. Then,

$$\gcd(a, b) = \gcd(b, r).$$

GCD Algorithm

```
**Procedure** GCD($a, b$: positive integers)
$x := a$
$y := b$
**while** $y \neq 0$
    $r := x \mod y$
    $x := y$
    $y := r$
**Print**($x$)
```

Modular Exponentiation Algorithm

Problem. Let b and m be positive integers. Compute $b^n \mod m$.

Example. Compute $3^{71} \mod 13$

```
**Procedure** ModExp($b, m$: positive integers, $n = (a_k, \ldots, a_0)_2$)
$x := 1$
power $:= b \mod m$
**for** $i := 0$ **to** $k$
    **if** $a_i = 1$ **then** $x := (x * \text{power}) \mod m$
    power $:=$ power $*$ power $\mod m$
**Print**($x$)
```