Mat 2345

Week 7

Fall 2013

Student Responsibilities — Week 7

- **Reading**: Textbook, Section 3.3–3.5
- **Assignments**: Sections 3.3, 3.4, and 3.5
- **Attendance**: Autumnally Encouraged

Week 7 Overview

- 3.3 Complexity of Algorithms
- 3.4 The Integers and Division
- 3.5 Primes and Greatest Common Divisors

Section 3.3 — Complexity of Algorithms

- When does an algorithm produce a satisfactory solution to a problem?

- How can we prove an algorithm always produces the correct answer? (seen in next chapter)

Analyzing Efficiency

How can we analyze the efficiency of an algorithm?

- **Time Complexity** — One measure is the number of steps it performs, or the time it takes, to solve a problem when input values are of a specified size
- **Space Complexity** — A second measure is the amount of computer memory required during execution of an implementation of the algorithm, when input values are of a specified size

Time Complexity

- It is obviously important to know whether an algorithm will produce an answer in milliseconds or time measured in years.

- Time complexity can be described in terms of the number of operations required instead of actual computer time — because of the difference in time needed for different computers to perform basic operations.

- It would be quite complicated to break down all operations to the basic bit operations that a computer uses

- Various machines, from personal computers to supercomputers, perform basic bit operations at rates which differ by as much as 1,000 times or more

Space Complexity

- It is obviously important to determine whether an algorithm will require more memory than we have available
- Space complexity can be described in terms of the amount of memory necessary to store one element $\times$ the size of input, plus additional storage required by the algorithm.
- It is often given in terms of the size of input and its storage requirements
- Considerations of space complexity are tied to the particular data structures used to implement the algorithm

Analyzing Time Complexity

We can count comparisons, data movement, arithmetic operations, or other types of "steps."

It just depends upon the problem and what's important to us

There are three types of analysis.

- **Best Case**
  - The run–time conditions are the best we can ever get
    for example: the numbers are already sorted, or there is only one value in the list so it is the max.
  - Doesn't give us very much information about the algorithm besides a lower–bound on execution time.

- **Average Case**
  - can be very difficult to determine.
  - cannot predict behavior for bad cases

- **Worst Case**
  - most commonly used (at undergraduate level)
  - gives an upper–bound on execution time, but can be overly pessimistic

Analyzing the Search Algorithms

based on the **number of comparisons** made:

- Linear Search
- Binary Search (for simplicity, assume there are $n = 2^k$ elements in the input list)

Commonly Used Complexity Terminology

Complexity	**Terminology**
$\Theta(1)$, $\Theta(c)$	`Constant complexity`
$\Theta(\log n)$	`Logarithmic complexity`
$\Theta(n)$	`Linear complexity`
$\Theta(n \log n)$	`nlog(n) complexity`
$\Theta(n^b)$	`Polynomial time complexity`
$\Theta(b^n)$, where b > 1	`Exponential complexity`
$\Theta(n!)$	`Factorial complexity`

Classes of Problems

- Problems for which answers can be found using a computer are called **solvable**

- Problems which are not solvable by computer are called **unsolvable**

One famous example of an unsolvable problem:

The Halting Problem

Can one program determine whether another arbitrary program will halt when executed with a specified input?

(The answer is no – a proof is given in the textbook, pg 176.)

Tractable vs Intractable Problems

- A solvable problem is called **tractable** if there exists an algorithm with polynomial worst–case complexity to solve it.
- Even if a problem is tractable, there's no guarantee it can be solved in a reasonable amount of time for even relatively small input values.
- Most algorithms in use have polynomial complexities of degree 4 or less.

- Solvable algorithms with worst–case time complexities that exceed polynomial times are called **intractable**

- Usually, but not always, an extremely large amount of time is required to solve the problem for the worst cases of even small input values.

- In a few instances, an exponential or worse algorithm may be able to solve problems of reasonable size in sufficient time to be useful

Further Algorithm Classifications

- Tractable algorithms are said to belong to **class P** — polynomial time algorithms.
- It is commonly believed that many solvable problems have no polynomial time algorithm to solve them, but that given a possible solution, it can be checked in polynomial time.
- Problems for which a solution can be checked in polynomial time belong to the **class NP**
- NP stands for **Non–deterministic polynomial** time — we **guess** an answer then **check** it in **polynomial** time.

NP–Complete Problem Class

Another important class of problems, called **NP–Complete** problems, are problems in the class NP which have the property that:

> If **any** of the problems in the **NP–Complete** class can be solved in polynomial time, then **all** of them can be

No one has been able to find such an algorithm.
It is suspected that no one ever will.

Section 3.4 — The Integers and Division

- Integers and their properties belong to a branch of Mathematics called **Number Theory**
- If a and b are integers, with $a \neq 0$, we say that ***a* divides *b*** if there is an integer c such that $b = ac$.

Notation: $a \mid b$, a divides b
a is a *factor* of b; b is a *multiple* of a
$a \nmid b$, a does not divide b

Theorem. Let a, b, and c be integers. Then:

The sum of multiples is a multiple:

if $a \mid b$ and $a \mid c$, then $a \mid (b + c)$

If an integer divides a factor, then it divides the product:

if $a \mid b$, then $a \mid bc$ for all integers c

Divisibility is transitive:

if $a \mid b$ and $b \mid c$, then $a \mid c$

Corollary. If a, b, and c are integers such that $a \mid b$ and $a \mid c$, then $a \mid (mb + nc)$ whenever m and n are integers.

Division of Integers

- **Theorem. The "Division Algorithm"**.
  Let **a** be an integer, **d** be a positive integer. Then there are unique integers **q** and **r**, with $0 \leq r < d$, such that

$$\boxed{\boxed{\mathbf{a = dq + r}}}$$

- In the equality given in the division algorithm:

  d is called the **divisor**,

  a is called the **dividend**,

  q is called the **quotient**, and

  r is called the **remainder**

Modular Arithmetic

- Let a be an integer and m be a positive integer. We denote by **(a mod m)** the **remainder** when a is divided by m.

  From this definition, it follows that:
  if $(a \textbf{ mod } m) = r$, then $a = qm + r$ and $0 \le r < m$.

- If a and b are integers, and m is a positive integer, then
  a is congruent to b modulo m if m divides $a - b$.
  This is denoted by: $a \equiv b (\text{mod } m)$

  Note: a **mod** m and b **mod** m will yield the same remainder.

  Consider: $(17 - 5) \bmod 6 \; = \; 12 \bmod 6 \; = \; 0$

  Also: $17 \bmod 6 = 5$ and $5 \bmod 6 = 5$

  Thus: $17 \equiv 5 \pmod 6$

More on Congruency

- **Theorem**. Let m be a positive integer. The integers a and b are congruent modulo m if and only if there is an integer k such that $a = b + km$.

- **Theorem**. Let m be a positive integer. If $a \equiv b (mod \;\; m)$ and $c \equiv d (mod \;\; m)$, then

$$a + c \equiv (b + d)(mod \;\; m)$$

and

$$ac \equiv bd(mod \;\; m)$$

Consider: $7 \equiv 2(\text{mod } 5)$ and $11 = 1(\text{mod } 5)$

Thus: $7 + 11 = 18 \equiv (2{+}1)(\text{mod } 5) = 3(\text{mod } 5)$

And: $7 \cdot 11 = 77 \equiv (2 \cdot 1)(\text{mod } 5) = 2(\text{mod } 5)$

Applications of Congruences

- **Hashing Functions** — assign memory locations to values, records (keys), or computer files for easy retrieval

- **Pseudo-random Numbers** — systematically generate a sequence of numbers that have properties of randomly chosen numbers

- **Cryptology** — encryption, to make a message secret;
  decryption, to determine the original message

Section 3.5 — Primes and Greatest Common Divisors

- A positive integer $p > 1$ is called **prime** if the only positive factors of p are 1 and p.

- A positive integer that is greater than 1 and is not prime is called **composite**.

- **The Fundamental Theorem of Arithmetic**.
  Every positive integer can be written uniquely as the product of primes, where the prime factors are written in order of non–decreasing size.

Showing Primality

Theorem. If n is a composite integer, then n has a prime divisor less than or equal to $\sqrt{n}$.

Show 103 is prime, using the above theorem and the fact $\sqrt{103} < 11$.

How to factor: attempt to divide by known primes beginning with 2, 3, 5, . . .

Theorem. There are infinitely many primes.
See proof by contradiction, pg 212.

Distribution of Prime Numbers

The Prime Number Theorem. The ratio of the number of primes not exceeding x and $\frac{x}{\ln(x)}$ approaches 1 as x grows without bound.

In other words, the number of primes $\leq x$ is $\sim \frac{x}{\ln(x)}$.

The probability that a random integer $< x$ is prime is $\sim \frac{1}{\ln x}$

Further, the probability that an integer n is prime is $\sim \frac{1}{\ln(n)}$.

Goldbach's Conjecture

Every even integer $n > 2$ is the sum of two primes.

(Has been checked for all even numbers up to 2×10^{17})

The Twin Primes Conjecture

There are infinitely many twin primes — primes that differ by 2.

(e.g., $< 3, 5 >, < 17, 19 >, < 4967, 4969 >$)
Largest known (2012): $3,756,801,695,685 \times 2^{666669} \pm 1$, numbers with 200,700 digits; `http://primes.utm.edu/top20/page.php?id=1`

Relatively Prime Integer Pairs

- Let a and b be integers, not both zero. The largest integer d such that $d \mid a$ and $d \mid b$ is called the **greatest common divisor**, denoted **gcd(a, b)**.

  Find the greatest common divisors:

  gcd(12, 39):

  gcd(23, 103):

  gcd(8, 9):

  gcd(28, 42):

- The integers a and b are **relatively prime** if their greatest common divisor is 1.

Pairwise Relatively Prime Integers

- The integers $a_1, \ a_2, \ \ldots, \ a_n$ are **pairwise relatively prime** if $\gcd(a_i, \ a_j) = 1$ whenever $1 \leq i < j \leq n$.

Are the following sets of integers pairwise relatively prime?

- 22, 28, and 31
- $(2^3 \cdot 5 \cdot 11)$, $(3^3 \cdot 7)$, and $(13 \cdot 23^5)$
- $(2^3 \cdot 5 \cdot 11)$, $(3^2 \cdot 11^2)$, and 29

Least Common Multiple

The **Least Common Multiple** of the positive integers **a** and **b**, denoted **lcm(a, b)**, is the smallest positive integer that is divisible by both a and b.

What is the least common multiple of each of the following pairs?

- $\text{lcm}(22, 28) =$
- $\text{lcm}(2^3 \cdot 5 \cdot 11, 3^3 \cdot 7) =$
- $\text{lcm}(13 \cdot 23^5, 13^2) =$
- $\text{lcm}(2^3 \cdot 5 \cdot 11, 3^2 \cdot 11^2) =$

Product of gcd and lcm

Theorem. Let **a** and **b** be positive integers. Then:

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

Consider:

- $600 = 2^3 \cdot 3 \cdot 5^2$ and $56250 = 2 \cdot 3^2 \cdot 5^5$
- $\operatorname{lcm}(600, 56250) = 2^3 \cdot 3^2 \cdot 5^5$
- $\gcd(600, 56250) = 2 \cdot 3 \cdot 5^2$
- $\text{product}(600, 56250) = 2^4 \cdot 3^3 \cdot 5^7$
- $\text{product}(\operatorname{lcm}(600, 56250), \gcd(600, 56250)) = 2^4 \cdot 3^3 \cdot 5^7$

Why are the last two products equivalent?