Today's topics

• Algorithms
  – Algorithm review
  – Orders of Growth
  – Algorithm review

Upcoming

Reading: Sections 2.1-3.3

Integers
§2.1: Algorithms

The foundation of computer programming.

Most generally, an algorithm just means a definite procedure for performing some sort of task.

We say that a program implements (or "is an implementation of") its algorithm.

A computer program is simply a description of an algorithm, in a language precise enough for a computer to understand, requiring only operations that the computer already knows how to do.

The foundation of computer programming.

§2.1: Algorithms
Algorithm Characteristics

Some important general features of algorithms:

- **Input**: Information or data that comes in.
- **Output**: Information or data that goes out.
- **Definiteness**: Algorithm is precisely defined.
- **Correctness**: Outputs correctly relate to inputs.
- **Generality**: Works for many possible inputs.
- **Finiteness**: Won't take forever to describe or run.
- **Effectiveness**: Individual steps are all do-able.
- **Efficiency**: Takes little time & memory to run.
Our Pseudocode Language: §A2

- **Procedure**
  - `procedure name (argument : type) {
    variable := expression
    informal statement
    begin statement
    statements
    end statement
    {comment}
    if condition then statement
    {comment}
    [else statement]
    for variable := initial to final value
    statement
    while condition statement
    procedure procname (argument (argument))
  }`
  - Not defined in book
  - **Return** expression

- **Statement**
  - `value to final value = initial for variable`
procedure maximum

\( \text{maximum}(L: \text{list of integers}) \)

Example:

- data objects of the type \( L \)
- inputs \( L \) which are
- procedure named \( \text{maximum} \) that takes
- \( \text{maximum}(L: \text{list of integers}) \)

\( \text{maximum}(L: \text{list of integers}) \)

\( \text{maximum}(L: \text{list of integers}) \)
An assignment statement evaluates the expression, then reassigns the variable to the value that results.

- Example assignment statement:
  
  \[ \text{if } x \text{ is 2, changes } v \text{ to 13.} \]

- In pseudocode (but not real code), the expression might be informally stated:

  \[ x := \text{the largest integer in the list} \]

- An assignment statement evaluates the expression:\n
  \[ \text{variable} := \text{expression} \]
Informal statement

- Break down algorithms into detailed steps.
  - Sometimes we may write a statement as an informal English imperative, if the meaning is still clear and precise. E.g., "swap x and y" is still clear and precise. Languages never allow this.
  - When we ask for an algorithm to do so-and-so, writing "Do so-and-so" isn't enough!
    - Keep in mind that real programming languages never allow this.
    - Sometimes we may write a statement as an informal English imperative, if the meaning is still clear and precise. E.g., "swap x and y" is still clear and precise.
Curly braces are used instead in many languages.

- Might be used:
  - Single statement
  - To be used just like a statement
- Allows the sequence of statements to be used just like a single statement
- After a procedure
- After a declaration
- In the body of a `for` or `while` loop.
- In an `if` statement after `then` or `else`.
- In an `if` statement after `then` or `else`.

```
begin
  statement 1
  statement 2
  ...
end
```

Groups a sequence of statements together:

```
begin
  statement 1
  statement 2
  ...
end
```
\{\text{Note that } v \text{ is the largest integer seen so far}\} –

- Natural-language text explaining some aspect of the procedure to human readers.
  - Also called a \textit{remark} in some real-language languages, e.g. BASIC.
- Example, might appear in a \texttt{max} program:
  \begin{verbatim}
  \text{Note that } v \text{ is the largest integer seen so far.}
  \end{verbatim}

- Not executed (does nothing).
Evaluate the propositional expression $\text{condition}$. 

- Variant: $\text{if cond then stmt1 else stmt2}$ 
  - Like before, but if the truth value is False, execute $\text{stmt2}$.
  - Otherwise, just skip on ahead to the next statement after the $\text{if}$ statement.

- If the resulting truth value is True, then execute the statement $\text{stmt1}$.
while condition

statement.

Evaluate the propositional (Boolean) expression condition.

If the resulting value is True, then execute the statement.

Continue repeating the above two actions over and over until finally the condition evaluates to False; then proceed to the next statement.
Also equivalent to infinite nested if's, like so:

```
while condition
  statement
  begin
    if condition
      begin
        statement
      end
    end
  end
```

(end (continue infinite nested if's))
for var := initial to final stmt

Semantics: Repeatedly execute stmt, first with variable var := initial, then with var := initial + 1, then with var := initial + 2, etc., then finally with var := final.

Question: What happens if stmt changes the value of var, or the value that initial or final evaluates to?
for
var := initial to final
stmt
• For can be exactly defined in terms of
while, like so:
begin
while var := initial
stmt
begin
var := var + 1
end
end

For var := initial to final stmt
A procedure call statement invokes the named procedure, giving it as its input the value of the argument expression. Various real programming languages refer to procedures as functions, or as subroutines, subprograms, or methods.

procedure (argument)
\begin{verbatim}
return \\
\{the largest integer in the list\}
\{at this point \(v\)'s value is the same as\}
\{found bigger?\} \(v := a_i \land a_i < v\) \land \forall \:\forall \:
\{go thru rest of elements\} \:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\\}
\{largest element so far\}
\{at this point \(v\)'s value is the same as\}
\{the largest integer in the list\}
\end{verbatim}
4.17

Inventing an Algorithm

• Requires a lot of creativity and intuition
• Like writing proofs.
• Requires a lot of creativity and intuition

And practice some more... etc., etc.
And look at more examples...
And practice (preferably, on a computer)
Just look at lots of examples...

And look at lots of examples...

Inventing an Algorithm
Suppose we ask you to write an algorithm to compute the predicate: Computes whether a given natural number is a prime number.

First, start with a correct predicate-logic definition of the desired function:

\[
\forall n: \text{IsPrime}(n) \iff \exists 1 < p < n: d \mid n
\]

Means \( d \) divides \( n \) evenly (without remainder)

\( n \mid d \iff \exists 1 < p < n: d \mid n \)

A: \( \text{IsPrime}(n) \)

- Computes whether a given natural number is a prime number:

\( \{ \text{T, F} \} \leftarrow \text{IsPrime}: \mathbb{N} \)

Algorithm-Inventing Example
Notice that the negated exponential can be rewritten as a universal:

\[ \neg \exists d \in \mathbb{N} \colon d > 1 \text{ and } d \mid n \]

This universal can then be translated directly into a corresponding for loop:

```
for p = 2 to n - 1
    if p \mid n
        return False
            { no divisors were found; n must be prime }
    else
```

The remainder is not 0 means p does not divide n evenly.

\( u \mid p : u > p > 1 \Rightarrow u \mid p : u > p > 1 \text{ and } u \not\mid n \)}

IsPrime example, cont.
Further optimizations are possible:

- **E.g.,** only try divisors that are primes less than \(\sqrt{n}\).
- **E.g.,** only try divisors that are primes less than \(\sqrt{n}\).

Note smaller range of search.

The *IsPrime* algorithm can be further optimized:

**Theorem:** This works because of this theorem: If \(n\) has any (integer) divisors, it must have one less than \(\sqrt{n}\). Proof: Suppose \(n\)'s smallest divisor \(a\) is \(\sqrt{n}\) and let \(b = n/a\). Then \(n = ab\), but if \(a < \sqrt{n}\) then \(b > \sqrt{n}\) (since \(a\) is the smallest divisor) and so \(ab > \sqrt{n}\cdot\sqrt{n} = n\), an absurdity.

Further optimizations are possible:

- E.g., only try divisors that are primes less than \(\sqrt{n}\).
Another example task
procedure linear search

\( (x, \text{integer}, a_1, a_2, \ldots, a_n, \text{distinct integers}) \)

\( i := 1 \) \{start at beginning of list\}

\( i := i + 1 \) \{go to the next position\}

if \( i \leq n \land x = a_i \) then \{it was found\}
else if \( i \leq n \land x \neq a_i \) then \{not done, not found\}

location := i

else location := 0 \{it wasn’t found\}

return location \{index or 0 if not found\}
Search alg. #2: Binary Search

Basic idea: On each step, look at the middle element of the remaining list to eliminate half of it, and quickly zero in on the desired element.
Search alg. #2: Binary Search

procedure binary search

\[ \text{return } \text{location} \]

if \( x \neq a \) then \( \text{location} \leftarrow 0 \)

else \( \text{location} \leftarrow \text{search} \)

end

\( m \leftarrow \left\lfloor \frac{i + j}{2} \right\rfloor \)

\( m \leftarrow \text{midpoint} \)

while interval has \( >1 \) item

\( i \leftarrow m + 1 \) else \( j \leftarrow m \)

{left endpoint of search interval}

{while interval has \( >1 \) item}

{right endpoint of search interval}

\( u \leftarrow j \)

{right endpoint of search interval}

\( l \leftarrow i \)

{left endpoint of search interval}

(x: integer, \( a_1, a_2, \ldots, a_n \): distinct integers)

procedure binary search

Search alg. #2: Binary Search
Practice exercises

2.1.3: Devise an algorithm that finds the sum of all the integers in a list. [2 min]

procedure sum(a1, a2, ..., an: integers)

s := 0
{sum of all the integers}
for i := 1 to n
{s := s + ai}
{sum of all the integers}
{go thru all elems}
{s := s + ai}
{add current item}
{at this point s is the sum of all items}
return s
Suppose you are designing a web site to process user data (e.g., financial records).

Suppose database program A takes \( f^B(n) = 30n^2 + 8 \) microseconds to process any records, while program B takes \( f^A(n) = un^2 + 1 \) microseconds to process the \( n \) records.

Which program do you choose, knowing you'll want to support millions of users?
Visualizing Orders of Growth

On a graph, as you go to the right, the faster-growing function always eventually becomes the larger one...
Concept of order of growth

For large numbers of user records, the exactly order $n^2$ function will always take more time.

- It is (at most) roughly proportional to $n^2$.
- It is, at most, roughly proportional to $n^2$, or $O(n^2)$ is faster-growing than any $O(n)$ function.
- Any function whose exact (tightest) order is $O(n^2)$ is faster-growing than any $O(n)$ function.
- Later we will introduce $\Theta$ for expressing exact order.

Any function whose exact (tightest) order is $O(n^2)$ is faster-growing than any $O(n)$ function.

- It is (at most) roughly proportional to $n^2$.
- It is, at most, roughly proportional to $n^2$.
- We say $f(n) = \Theta(n^2)$ is (at most) order $n^2$, or $O(n^2)$.

We say $f(n) = \Theta(n^2)$ is (at most) order $n^2$, or $O(n^2)$. 

Concept of order of growth
Definition: $O(g)$, at most order $g$

Let $g$ be any function $\in \mathbb{R}$. All just mean that $f \in O(g)$, or $f$ is at most order $g$, or $f$ is at most a constant times $g$ (i.e., proportional to $g$).

Beyond some point, function $f$ is at most a constant $c$ times $g$.

\[
\{ (x) \mid \exists k \in \mathbb{R} \land x < k : f(x) \leq cg(x) \}
\]

Define "at most order $g$", written $O(g)$, to let $g$ be any function $\in \mathbb{R}$.
Points about the definition

However, you should prove that the values you choose do work.

• Note that $f(n)$ is $O(g(n))$ so long as any values of $c$ and $k$ exist that satisfy the definition. Any larger value of $c$ and/or $k$ will also work.

You are not required to find the smallest $c$ and $k$ values that work. (Indeed, in some cases, there may be no smallest values!) But: The particular $c$, $k$ values that make the statement true are not unique: Any

Note that $f(n)$ is $\Omega(g(n))$ so long as any values of $c$ and $k$ exist that satisfy the definition.
4.31

"Big-O" Proof Examples

- Show that \(30n^2 + 8\) is \(O(n^2)\).
  - Let \(c = 2\), \(k = 1\). Assume \(n > k\). Then
    \[ c = 2n^2 = 2n^2 < n^2 + 1, \text{ or } n + 1 > cn. \]

- Show that \(n^2 + 1\) is \(O(n^2)\).
  - Let \(c = 2\), \(k = 1\). Assume \(n > k\). Then
    \[ c = 2n^2 = 2n^2 < n^2 + 1, \text{ or } n + 1 > cn. \]
• Note $30n+8$ isn’t less than $n$ anywhere ($n>0$).

• But it isn’t even less than $31n$ everywhere.

• It isn’t even less than $31n$ everywhere to the right of $n=8$.

Big-O example, graphically

Value of function $\rightarrow$

Increasing $n \rightarrow$

$30n+8 \in O(n)$
Useful Facts about Big O

• If $g \in O(f)$ and $h \in O(f)$, then $g + h \in O(f)$.

• Big O, as a relation, is transitive:

  \[(f)O \subseteq (g)O \Rightarrow (f)O \subseteq (h)O \Rightarrow (f)O \subseteq (g+h)O\]

• Sums of functions:

  If $f \in O(g)$ and $f \in O(h)$, then $f + g \in O(h)$.

• Big O with constant multiples, roots, and logs:

  If $f \in O(g)$, then $\alpha f \in O(g)$ for all $\alpha > 0$.

  If $f \in O(g)$ and $f \in O(h)$, then $\log f \in O(h)$. ...
More Big-O facts

\[ f_1 \in \Theta(g_1) \land f_2 \in \Theta(g_2) \implies f_1 + f_2 \in \Theta(g_1 + g_2) \]

\[ f_1 \in \Theta(g_1) \land f_2 \in \Theta(g_2) \implies \max\{f_1, f_2\} \in \Theta(\max\{g_1, g_2\}) \]

\[ (f \circ g) \in \Theta(c \circ f) \]

\[ f \in \Theta(g) \implies (c - f) \in \Theta(c - g) \]

\[ f \in \Theta(g) \implies (c + f) \in \Theta(c + g) \]

\[ (f \circ c) \in \Theta(f(c)) \]
Orders of Growth (§1.8) - So Far

...convention...

The latter form is an instance of a more general

\[ (\delta)O = f, \quad \text{or} \quad (\delta)O \subset f \]

often written "\( (\delta)O \) is \( O(f) \)."

• Often, one deals only with positive functions

– Often, one can ignore absolute value symbols.

\[ \{ |(x)\delta| \geq |(x)f| \quad \forall \epsilon > 0 \} \quad \Rightarrow \quad (\exists \delta)O \subseteq f \]

For any \( \delta: \mathbb{R} \rightarrow \mathbb{R} \), "at most order \( \delta \)."
CompSci 102 © Michael Frank

4.36

Order-of-Growth Expressions

\{(x)f_+x = (x)\exists \delta : (x)O \subseteq E \mid \exists \}\equiv: (x)O +_x \exists x

Formally, you can think of any such expression as denoting a set of functions:

\cdot \forall (x)O = O \exists x \text{ means plus some } \exists x \text{ that function that is } O \exists x \text{ plus some function such expression means: } "\forall \text{ some function } f \text{ when used as a term in an arithmetic expression } = O."
Order of Growth Equations

Suppose $E_1$ and $E_2$ are order-of-growth expressions corresponding to the sets of functions $S$ and $T$, respectively. Then the "equation" $E_1 \subseteq E_2$ really means $\forall x \in S \subseteq T, x \in E_1 \Rightarrow x \in E_2$.

Example: $x^2 + O(x) = O(x^2)$ means $\forall x \in S, x \in O(x) \Rightarrow x \in O(x^2)$.

Or simply:

$$\forall x \in S, x \in E \Rightarrow x \in E'$$

$$E = f : A \subseteq E \Rightarrow \exists \exists f \in E, x \in F$$

Suppose $E$ and $F$ are order-of-growth expressions corresponding to the sets of functions $S$ and $T$, respectively.
Useful Facts about Big O

\[
\begin{align*}
((x)O &= \exists \mathcal{S} \cdot \mathcal{E}) & (f)O &= v - \\
((x)O \neq x \mathcal{S} \supset x \cdot \mathcal{S} \cdot \mathcal{E}) & (f)O \neq f - \\
((x \mathcal{S} \supset x)O &= x \cdot \mathcal{S} \cdot \mathcal{E}) & (f)O = \mathcal{S} - \\
((x)O &= x \mathcal{S} \supset \mathcal{E} \cdot \mathcal{E}) & (f)O = v(|f| \mathcal{S} \supset) - \\
((x)O = 1 - x \cdot \mathcal{S} \cdot \mathcal{E}) & (f)O = q - 1 |f| - \\
\end{align*}
\]

Also, if $O \geq I$ (at least order 1), then:

(\exists x \mathcal{S} \supset x \cdot \mathcal{E}) & (f)O = (f)O + f - \\
(\exists x \mathcal{S} \supset \mathcal{E} \cdot \mathcal{E}) & (f)O = f v - \\

A \exists \mathcal{S} \supset \mathcal{E} \cdot \mathcal{E} \quad \text{with } b \geq 0,
Definition: \( g \)

\[ \text{exactly order } g \]

\( f \) and \( g \) are of the same order if \( f \) and \( g \) are of the same order, or \( f \) is (exactly) order \( g \) and write \( \Theta(g) \) (exactly order \( g \)), then we say \( g \) and \( f \) have the same order, or \( f \) is (exactly) order \( g \) and \( g \) is (exactly) order \( f \).

Another, equivalent definition:

\[ \exists c_1, c_2 > 0 \text{ such that } c_1 g(x) \leq |f(x)| \leq c_2 g(x) \text{ for all } x \in \mathbb{R} \]

where \( f \) is (exactly) order \( g \) and \( g \) is (exactly) order \( f \).

\( \Theta \)
The functions in the latter two cases we say are strictly of lower order than \( f(\Theta) \).

\[
\begin{align*}
\text{Unlike with } O. & \quad \Rightarrow \quad (\forall \Theta \not\in \mathcal{O}(\log q)) \\
\text{Unlike with } O. & \quad \Rightarrow \quad \text{and, } (\forall \Theta \not\in \mathcal{O}(q) - 1) \\
\text{Unlike } O. & \quad \Rightarrow \quad (I) \Theta = \delta \text{ unless } (\delta f) \Theta \not\in f \\
\text{Same as with } O. & \quad \Rightarrow \quad \text{but, } (I) \Theta \in f \\
\text{A } f, g > 0 \text{ constants } a, b \in \mathbb{R}, \text{ with } b > 0, & \quad \Rightarrow \quad \text{Mostly like rules for } O(\cdot), \text{ except:}
\end{align*}
\]
Determine whether:

Quick solution: \[
\left( \sum_{i=1}^{\infty} u \right) \Theta u = \left( \sum_{i=1}^{\infty} u \right) \left( \sum_{i=1}^{\infty} u \right)
\]

Determine whether:

\[
\sum_{i=1}^{\infty} u
\]
Other Order-of-Growth Relations
Subset relations between order-of-growth sets.

Relations Between the Relations

Superset relations between order-of-growth sets.

Relations Between the Relations
A function that is $O(x)$, but neither $o(x)$ nor $\Theta(x)$.

Why $o(\Omega(x) - \Theta(x))$?
Strict Ordering of Functions

\[ \text{Temporarily let } f, g \text{ write } f \gg g \text{ to mean } f(n) \gg g(n) \text{ for all sufficiently large } n. \]

**Note that:**

- Let \( k > 1 \). Then the following are true:
  - \( \log n \log \log n \sim n^{1/k} \log n \log \log n \]
  - \( \log \log n \sim n^{1/k} \log n \log \log n \]
Definitions of order-of-growth sets:

\( (S) \cup (R) \equiv: (S) \emptyset \)

\( \{ (f) \in S \mid f \} \equiv: (S) _0 \)

\( \{ (f) \in S \mid f \} \equiv: (S) _\infty \)

\( \{ (x) \in S \mid \gamma < x \wedge \gamma \in E \wedge 0 < \gamma \} \equiv: (S) _0 \)

\( \{ (x) \in S \mid \gamma < x \wedge \gamma \in E \wedge 0 < \gamma \} \equiv: (S) _0 \)