Logo

Discrete Mathematics

Dr. Han Huang

South China University of Technology

Logo

Chapter 2. Set model

Function order and algorithm complexity

Section 2.2

Contents

Introduction

Functions

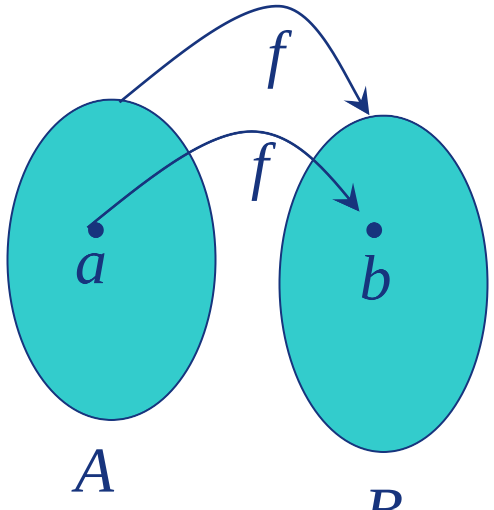
- From calculus, you know the concept of a real-valued function f, which assigns to each number $x \in \mathbf{R}$ one particular value $y=f(x)$, where $y \in \mathbf{R}$.
- *Example*: f is defined by the rule $f(x)=x^2$
- The notion of a function can be generalized to the concept of assigning elements of *any* set to elements of *any* set.
- Functions are also called operators.

Function: Formal Definition

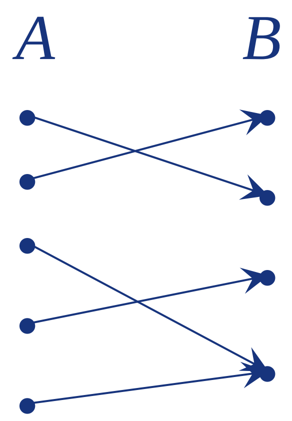
- A *function f from* (*or "mapping"*) *A to B* ($f{:}A\rightarrow B$) is an assignment of exactly one element $f(x)\in B$ to each element $x\in A$.
- Some further generalizations of this idea:
  - **Functions of *n* arguments:**
    f: $(A_1 \times A_2 \ldots \times A_n) \rightarrow B$.
  - **A *partial* (non-*total*) function *f* assigns *zero or one* elements of *B* to each element $x\in A$.**

- We can represent a function f:$A \rightarrow B$ as a set of ordered pairs $f = \{(a, f(a)) \mid a \in A\}$.
- This makes *f* a relation between A and B:
  f is a subset of A x B. But functions are special:
  - for every $a \in A$, there is at least one pair (a,b).
    Formally:
    $\forall a \in A \exists b \in B((a,b) \in f)$
  - for every $a \in A$, there is at most one pair (a,b).
    Formally:
    $\neg\exists a,b,c((a,b) \in f \wedge (a,c) \in f \wedge b \neq c)$
- A relation over numbers can be represent as a set of points on a plane. (A point is a pair (x,y).)
  - A function is then a curve (set of points),
    with only one y for each x.

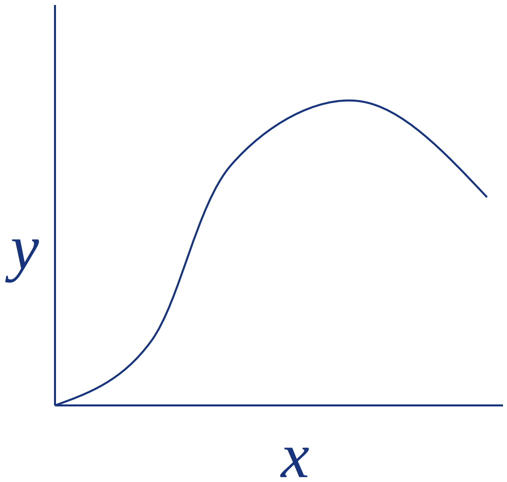
❖Functions can be represented graphically in several ways:



Like Venn diagrams



Bipartite Graph



Plot

Functions We've Seen So Far

- A *proposition* might be viewed as a function from "situations" to truth values {**T**,**F**}
  - p="It is raining."
  - s=our situation here,now
  - $p(s)\in\{\mathbf{T},\mathbf{F}\}$.
- A *propositional operator* can be viewed as a function from *ordered pairs* of truth values to truth values: *e.g.*, $\vee((\mathbf{F},\mathbf{T})) = \mathbf{T}$.

Another example: $\rightarrow((\mathbf{T},\mathbf{F})) = \mathbf{F}$.

More functions so far…

- A *predicate* can be viewed as a function from *objects* to *propositions*:
  $P :\equiv$ "is 7 feet tall";
  P(Xiaokun) = "Xiaokun is 7 feet tall."

- A *set S* over universe *U* can be viewed as a function from the elements of *U* to ...

Logo

Still More Functions

- A *set* S over universe U can be viewed as a function from the elements of U to …

  … {**T**, **F**}, saying for each element of U whether it is in S.

Suppose U={0,1,2,3,4}. Then

S={1,3}➔ S(0)=S(2)=S(4)=**F**, S(1)=S(3)=**T**.

Logo

Still More Functions

- A *set operator* such as ∩ or ∪ can be viewed as a function …

  … from ordered pairs of sets, to sets.
  - **Example: ∩({1,3},{3,4}) = {3}**

Logo

A new notation

- Sometimes we write Y^X to denote the set F of *all* possible functions $f: X \to Y$.
- Thus, $f \in Y^X$ is another way of saying that $f: X \to Y$.

- (This notation is especially appropriate, because for finite X, Y, we have $|F| = |Y|^{|X|}$.)

Some Function Terminology

- If f:$A \rightarrow B$, and $f(a)=b$ (where $a \in A$ & $b \in B$), then we say:
  - ***A*** **is the** ***domain*** **of** ***f*****.**
  - ***B*** **is the** ***codomain*** **of** ***f*****.**
  - ***b*** **is the** ***image*** **of** ***a*** **under** ***f*****.**
  - ***a*** **is a** ***pre-image*** **of** ***b*** **under** ***f.***
    - ⓾**In general,** ***b*** **may have more than 1 pre-image.**
  - **The** ***range*** $R \subseteq B$ **of** ***f*** **is** $R=\{b \mid \exists a\, f(a)=b\}$**.**

We also say the *signature* of *f* is $A \rightarrow B$.

Range versus Codomain

- The range of a function may not be its whole codomain.
- The codomain is the set that the function is *declared* to map all domain values into.
- The range is the *particular* set of values in the codomain that the function *actually* maps elements of the domain to.
- (One might say: The range is the smallest set that could be used as its codomain.)

Range vs. Codomain - Example

- Suppose I declare to you that: "*f* is a function mapping students in this class to the set of grades {A,B,C,D,E}."
- At this point, you know *f*'s codomain is: {A,B,C,D,E}, and its range is unknown.
- Suppose the grades turn out all As and Bs.
- Then the range of *f* is {A,B}, but its codomain is still {A,B,C,D,E}.

Logo

(n-ary) functions on a set

- An *n*-ary function (also: n-ary *operator)* *over* (also: *on*) *S* is any function from the set of ordered *n*-tuples of elements of *S*, to *S* itself.
- *E.g.*, if *S*={**T**,**F**}, $\neg$ can be seen as a unary operator, and $\wedge$,$\vee$ are binary operators on *S*.
- Another example: $\cup$ and $\cap$ are binary operators on the set of all sets.

Logo

Images of Sets under Functions

- Given $f{:}A\rightarrow B$, and $S\subseteq A$,
- The *image* of S under f is the set of all images (under f) of the elements of S.

$$f(S) :\equiv \{f(s) \mid s\in S\}$$
$$:\equiv \{b \mid \exists s\in S{:}\ f(s)=b\}.$$

- The range of f equals the image (under f) of f's domain.

Types and Operations

Logo

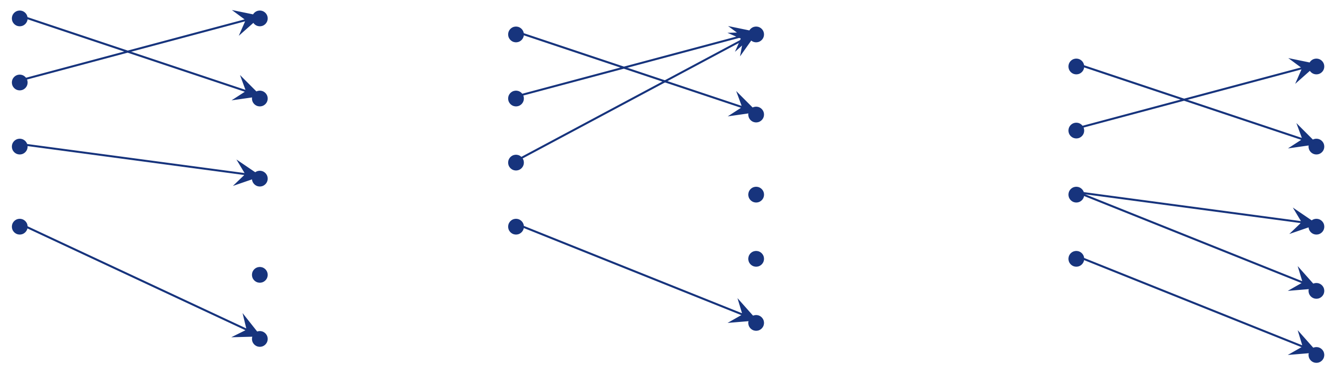
One-to-One Functions

- A function is *one-to-one (1-1)*, or *injective*, or *an injection*, iff every element of its range has *only* 1 pre-image.
  - Formally: given f:$A \rightarrow B$,
    "x is injective" $:\equiv (\neg\exists x,y\text{: } x \neq y \wedge f(x)=f(y))$.
- In other words: only one element of the domain is mapped to any given one element of the range.
  - In this case, domain & range have same cardinality.
    What about codomain?

Logo

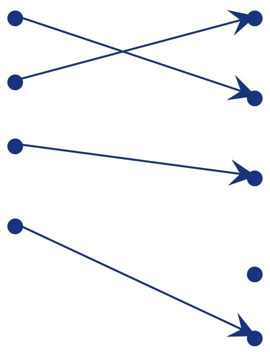
One-to-One Illustration

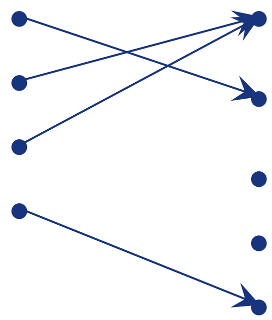
- Are these relations one-to-one functions?

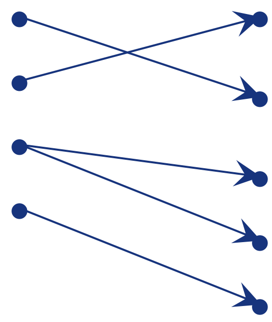


One-to-One Illustration

- Are these relations one-to-one functions?



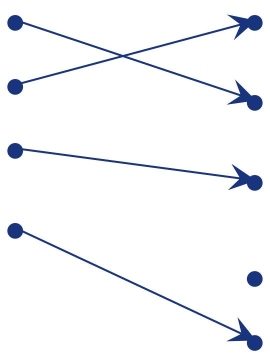




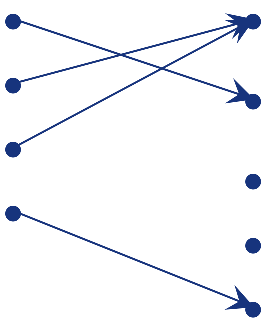
One-to-one

One-to-One Illustration

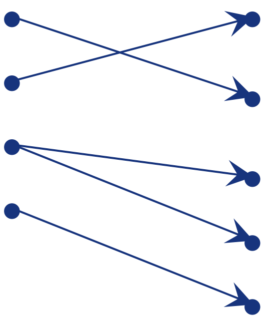
- Are these relations one-to-one functions?



One-to-one

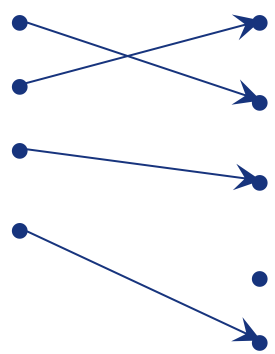


Not one-to-one

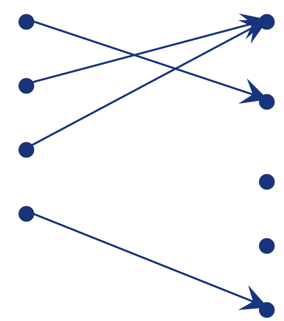


One-to-One Illustration

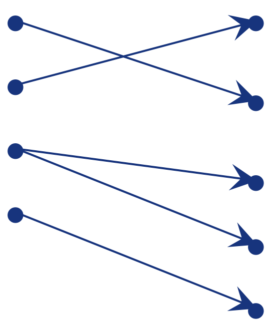
- Are these relations one-to-one functions?



One-to-one



Not one-to-one



Not even a function!

Logo

Sufficient Conditions for 1-1ness

- For functions *f* over numbers, we say:
  - ***f* is *strictly increasing* iff $x>y \rightarrow f(x)>f(y)$ for all *x*,*y* in domain;**
  - ***f* is *strictly decreasing* iff $x>y \rightarrow f(x)<f(y)$ for all *x*,*y* in domain;**
- If *f* is either strictly increasing or strictly decreasing, then *f* must be one-to-one.
  - **Does the converse hold?**

- If *f* is either strictly increasing or strictly decreasing, then *f* is one-to-one.
  - Does the converse hold? NO
- E.g., f:N→N such that
  if x is even then f(x)=x+1
  if x is odd then f(x)=x-1

Logo

Onto (Surjective) Functions

- A function $f\!:\!A\rightarrow B$ is *onto* or *surjective* or *a surjection* iff its range is equal to its codomain ($\forall b\in B, \exists a\in A\!: f(a)=b$).
- Consider *"country of birth of"*: A→B, where A=people, B=countries.

  Is this a function?

  Is it an injection?

  Is it a surjection?

Onto (Surjective) Functions

- A function $f{:}A \rightarrow B$ is *onto* or *surjective* or *a surjection* iff its range is equal to its codomain
- **Consider *"country of birth of"*: A→B, where A=people, B=countries.**
  Is this a function? **Yes (always 1 c.o.b.)**
  Is it an injection? **No (many have same c.o.b.)**
  Is it a surjection? **Probably yes …**

Onto (Surjective) Functions

- A function $f{:}A \to B$ is *onto* or *surjective* or *a surjection* iff its range is equal to its codomain.
- In predicate logic:

$$\forall b \in B \exists a \in A\ f(a)=b$$

Onto (Surjective) Functions

- A function $f{:}A\rightarrow B$ is *onto* or *surjective* or *a surjection* iff its range is equal to its codomain ($\forall b\in B\,\exists a\in A\ f(a)=b$).
- Think: An *onto* function maps the set *A* onto (over, covering) the *entirety* of the set *B*, not just over a piece of it.
- *E.g.*, for domain & codomain **R**, x^3 is onto, whereas x^2 isn't. (Why not?)

Onto (Surjective) Functions

E.g., for domain & codomain **R**,
x^3 is onto, but x^2 is not. (Why not?)

- Consider *f:R→R such that, for all x, f(x)=x²*.
  Consider any negative number a=-b in R.
  $\neg\exists x(x^2=a)$. So *f* is not surjective.
- Consider *f:R→R such that for all x, f(x)=x³*.
  Consider any negative number a=-b in R.
  Let *z* be such that $z^3=b$. Then $(-z)^3=-b=a$

The Identity Function

- For any domain *A*, the *identity function* $I:A\rightarrow A$ (also written I_A) on A is the function such that everything is mapped to itself
- In predicate logic: $\forall a \in A\ I(a)=a$.

The Identity Function

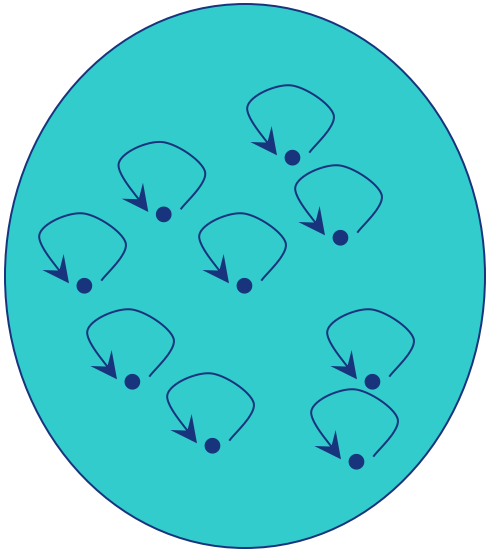
- For any domain *A*, the *identity function* $I:A \rightarrow A$ (also written I_A) on A is the function such that $\forall a \in A\ I(a)=a$.
- Is the identity function
  1. **one-to-one (injective)?**
  2. **onto (surjective)?**

The Identity Function

- For any domain A, the *identity function* $I:A \rightarrow A$ (also written I_A) on A is the function such that $\forall a \in A\ I(a)=a$.
- Is the identity function
  1. one-to-one (injective)?   yes
  2. onto (surjective)?   yes

Identity Function Illustrations

- The identity function:



Domain and range

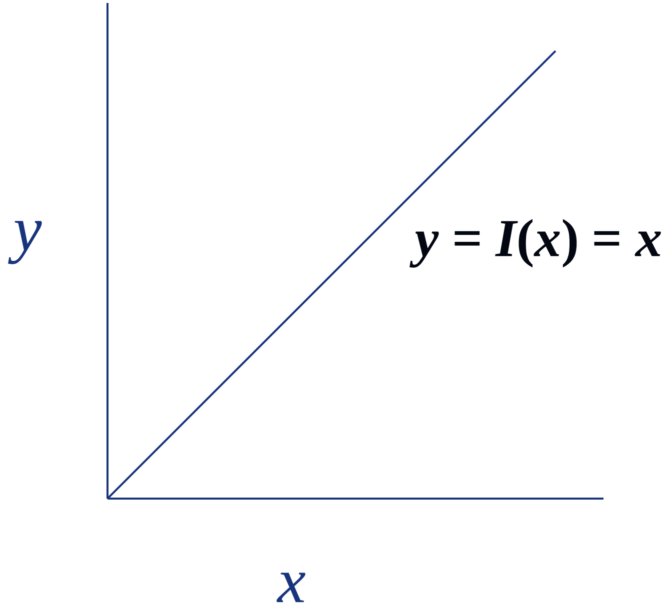


Illustration of Onto

- Are these functions *onto* their depicted co-domains?

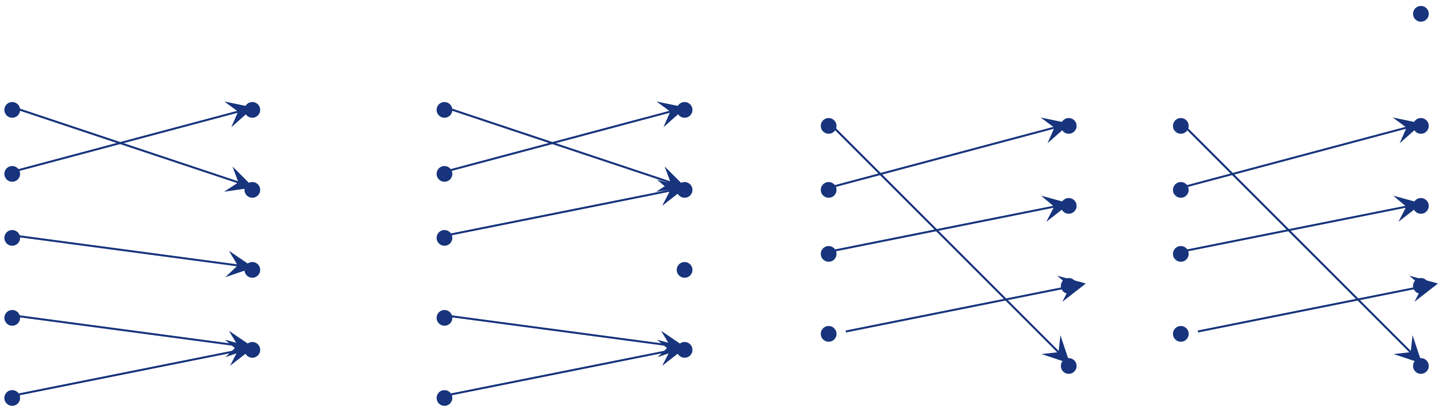


Illustration of Onto

- Are these functions *onto*?

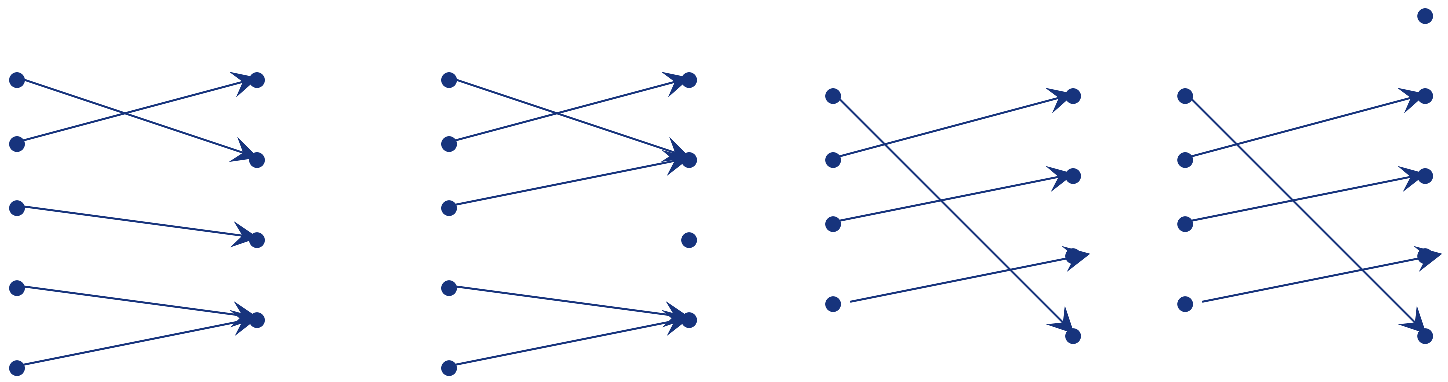


Illustration of Onto

- Are these functions *onto*?

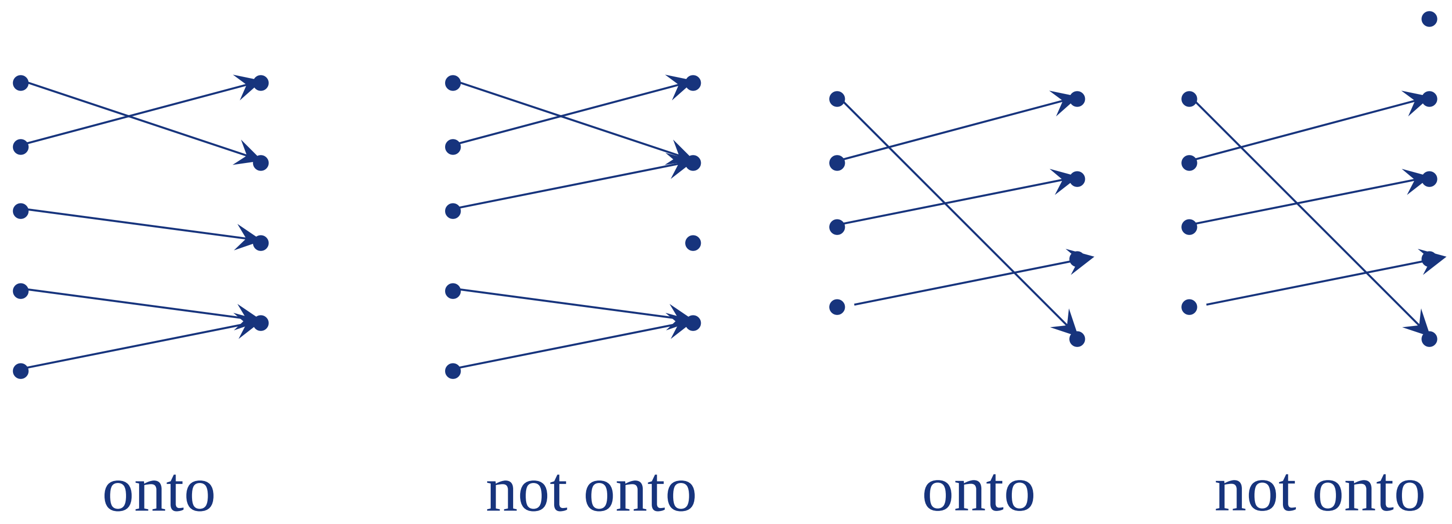


Illustration of Onto

- Are these functions 1-1?

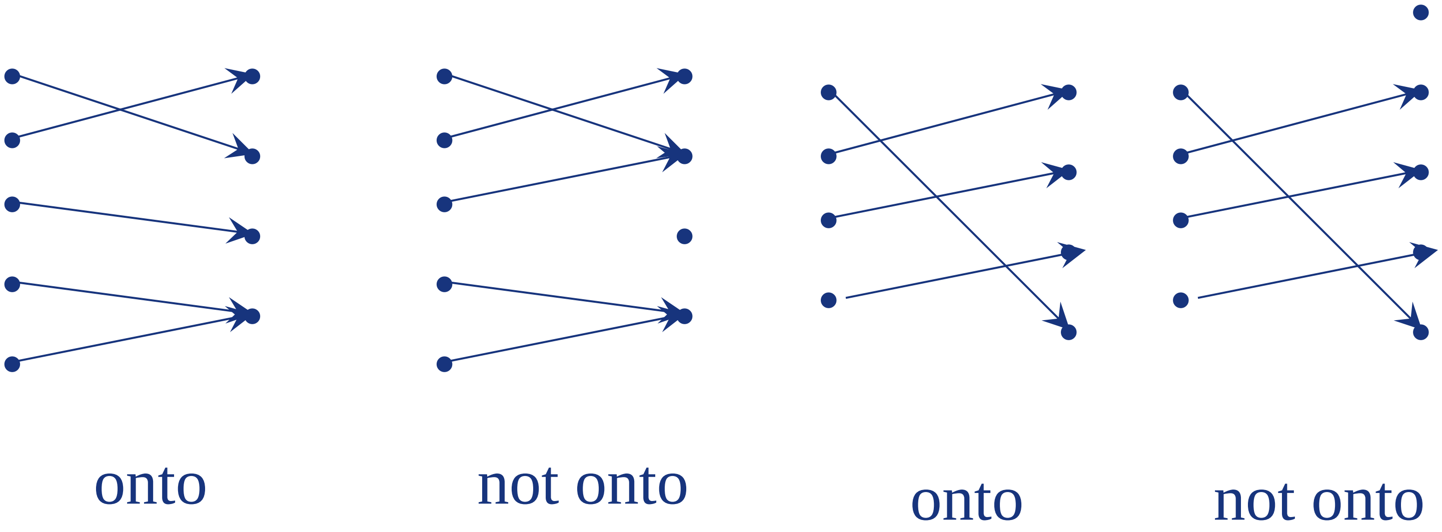
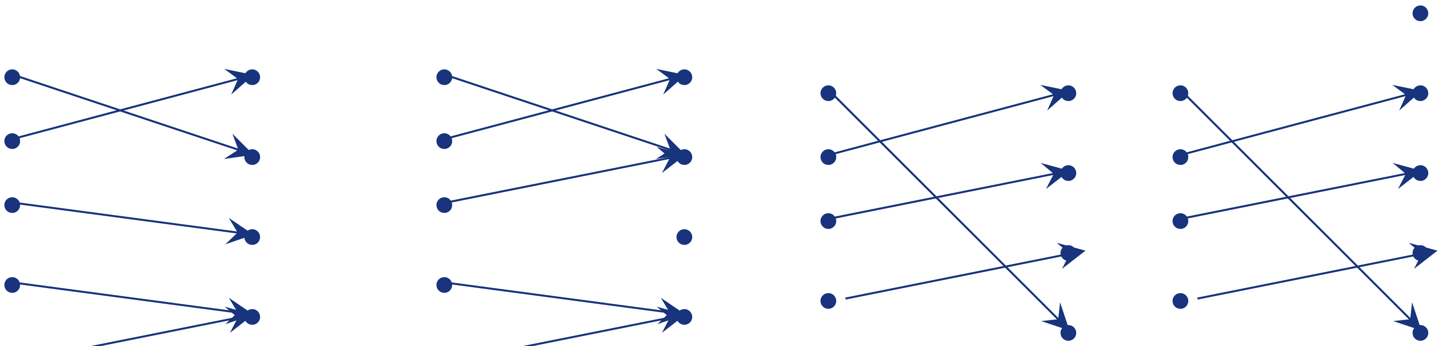


Illustration of Onto

- Are these functions 1-1?



not 1-1
onto

not 1-1
not onto

1-1
onto

1-1
not onto

Application of Functions

Bijections

- A function is said to be *a one-to-one* ***correspondence***, or *a bijection* iff it is both one-to-one and onto.

Logo

Two terminologies for talking about functions

1. injection = one-to-one
2. surjection = onto
3. bijection = one-to-one correspondence

3 = 1&2

Bijections

- For bijections $f{:}A\rightarrow B$, there exists an *inverse* of f, written $f^{-1}: B\rightarrow A$
- Intuitively, this is the function that undoes everything that f does
- Formally, it's the unique function such that

...

Bijections

- For bijections $f{:}A \rightarrow B$, there exists an *inverse* of f, written $f^{-1}{:}\ B \rightarrow A$
- Intuitively, this is the function that undoes everything that f does
- Formally, it's the unique function such that

$$f^{-1} \circ f = I_A$$

(recall that I_A is the identity function on A)

Bijections

- Example 1: Let f: $\mathbf{Z} \rightarrow \mathbf{Z}$ be defined as $f(x)=x+1$. What is f^{-1}?
- Example 2: Let g: $\mathbf{Z} \rightarrow \mathbf{N}$ be defined as $g(x)=|x|$. What is g^{-1}?

Bijections

- Example 1: Let f: **Z**→**Z** be defined as f(x)=x+1. What is f^{-1}?
- f^{-1} is the function (let's call it h) h: **Z**→**Z** defined as h(x)=x-1.
- Proof:

$$h \circ f = I$$

h(f(x)) = (x+1)-1 = x

Bijections

- Example 2: Let g: $\mathbf{Z} \to \mathbf{N}$ be defined as $g(x)=|x|$. What is g^{-1}?
- This was a trick question: there is no such function, since g is not a bijection:
  There is no function h such that
  $h(|x|)=x$ and $h(|x|)=-x$
- (NB There is a relation h for which this is true.)

Logo

Operators over functions

- If • ("dot") is an n-ary operator over B, then we can extend • to also denote an operator over <u>functions</u> from A to B.
- *E.g.*: Given any binary operator $\bullet: B\times B\rightarrow B$, and functions f, g: $A\rightarrow B$, we define $(f \bullet g): A\rightarrow B$ to be the function defined by: $\forall a\in A, (f \bullet g)(a) = f(a)\bullet g(a)$.

Function Operator Example

- $+,\times$ (plus,times) are binary operators over **R**. (Normal addition & multiplication.)
- Therefore, we can also "add" and "multiply" *functions* f,g: $\mathbf{R}\rightarrow\mathbf{R}$:
  - $(f+g)$:**R→R, where** $(f+g)(x) = f(x) + g(x)$
  - $(f\times g)$:**R→R, where** $(f\times g)(x) = f(x)\times g(x)$

Logo

Function Composition Operator

- For functions g:$A \rightarrow B$ and f:$B \rightarrow C$, there is a special operator called *compose* ("○").

Note match here.

  - It <u>composes</u> (creates) a new function out of f and g by applying f to the result of applying g.
  - We say $(f \circ g)$:$A \rightarrow C$, where $(f \circ g)(a) :\equiv f(g(a))$.
  - $g(a) \in B$, so $f(g(a))$ is defined and $f(g(a)) \in C$.
  - Note that ○ is non-commutative (i.e., we don't always have $f \circ g = g \circ f$).

Function Composition Operator

"We don't always have $f \circ g = g \circ f$"

Can you express this in predicate logic?

Function Composition Operator

“We don’t always have $f \bigcirc g = g \bigcirc f$ ”

Can you express this in predicate logic?

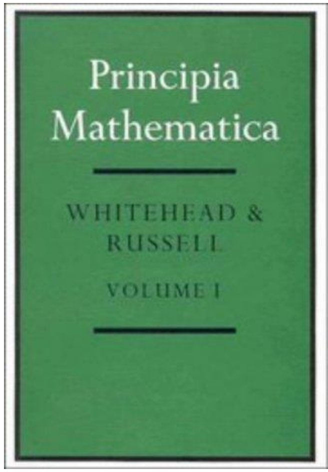
$\neg(\forall f \forall g \forall x (f \bigcirc g(x) = g \bigcirc f(x)))$.

[Do not write: $\forall f \forall g \forall x (f \bigcirc g(x) \neq g \bigcirc f(x))$]

(Note that this formula quantifies over functions as well as ordinary objects – something that is not possible in *First Order* Predicate Logic (FOPL), which is what was taught earlier in this course.)

Aside About Representations

- It is possible to represent any type of discrete structure (propositions, bit-strings, numbers, sets, ordered pairs, functions) in terms of some combination of other structures.
- Perhaps none of these structures is more fundamental than the others. However, logic, and sets are often used as the foundation for all else. *E.g.* in ➔



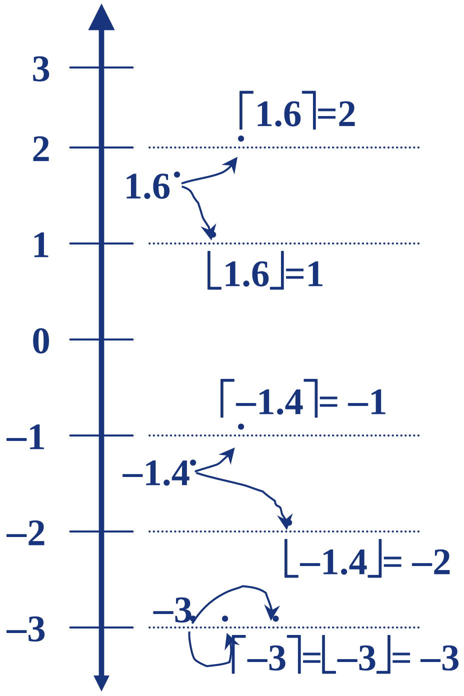
Logo

A Couple of Key Functions

- In discrete math, we frequently use the following two functions over real numbers:
  - **The *floor* function $\lfloor\cdot\rfloor$:R→Z, where $\lfloor x\rfloor$ ("floor of *x*") means the largest integer ≤ *x*. *I.e.*, $\lfloor x\rfloor$:≡ max({*i*∈Z|*i*≤*x*}).**
  - **The *ceiling* function $\lceil\cdot\rceil$:R→Z, where $\lceil x\rceil$ ("ceiling of *x*") means the smallest integer ≥ *x*. $\lceil x\rceil$:≡ min({*i*∈Z|*i*≥*x*})**

Visualizing Floor & Ceiling

- Real numbers "fall to their floor" or "rise to their ceiling."



Do these equalities hold?

- $\lfloor -x \rfloor = -\lfloor x \rfloor$ &
  $\lceil -x \rceil = -\lceil x \rceil$

It depends on whether x is an integer

- If $x \in Z$ then $\lfloor x \rfloor = \lceil x \rceil = x$, so
  $\lfloor -x \rfloor = -x = -\lfloor x \rfloor$ &
  $\lceil -x \rceil = -x = -\lceil x \rceil$
- But if $x \notin Z$, then
  $\lfloor -x \rfloor \neq -\lfloor x \rfloor$ &
  $\lceil -x \rceil \neq -\lceil x \rceil$
- E.g., $\lfloor -3.4 \rfloor = -4 \neq -3 = -\lfloor 3.4 \rfloor$

Plots with floor/ceiling

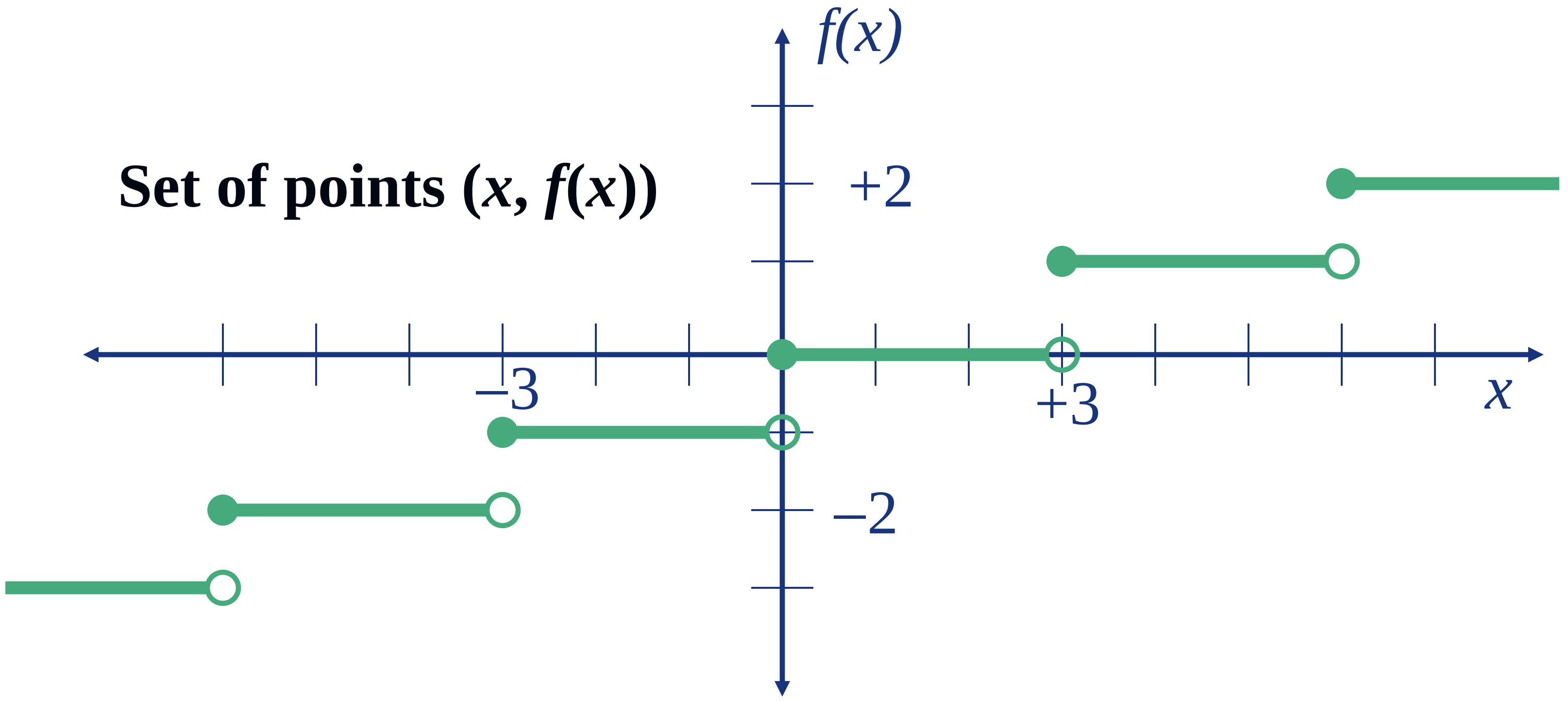
- Note that for $f(x)=\lfloor x \rfloor$, the graph of f includes the point (a, 0) for all values of a such that $a \geq 0$ and $a<1$, but not for the value $a=1$.
- $\{x \in R : \lfloor x \rfloor = 0\}$ = (informally) =

  {0,..,0.1,..0.2,..,…0.9,…} does not include its *limit* 1.
  - Sets that do not include all of their limit points are generally called *open sets*.
- In a plot, we draw a limit point of a curve using an open dot (circle) if the limit point is not on the curve, and with a closed (solid) dot if it is on the curve.

Plots with floor/ceiling: Example

- Plot of graph of function $f(x) = \lfloor x/3 \rfloor$:

Set of points $(x, f(x))$



Applications

Logo

假设f定义如下表。

A	B	C	D	E	F	G	H	I	J	K	L	M
D	E	S	T	I	N	Y	A	B	C	F	G	H
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
J	K	L	M	O	P	Q	R	U	V	W	X	Z

即f(A)=D,f(B)=E,f(C)=S,…等等。

试找出给定密文“QAIQORSFDOOBUIPQKJBYAQ”对应的明文。

解由表知，f^{-1}如下表所示。

A	B	C	D	E	F	G	H	I	J	K	L	M
H	I	J	A	B	K	L	M	E	N	O	P	Q
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
F	R	S	T	U	C	D	V	W	X	Y	G	Z

将密文“QAIQORSFDOOBUIPQKJBYAQ”中的每一个字母在f^{-1}中找出其对应的象就可得出对应的明文：“THETRUCKARRIVESGONIGHT”。

为什么函数比一般的映射对人来说更重要?

A和B（蒙眼）玩游戏，A用声音为B指路。

前面是一条沟。

A说：“沟、沟、沟。”

没想到B酷爱英语，以为是“go、go、go”，B大胆地往前走，结果掉沟里了。

这个例子说明严密定义很重要，不然不清楚你说的沟是什么。

"沟"声音

沟

Go

not 1-1

这是个映射，而不是函数

这说明了函数的重要性，因为一对多的映射确实容易出问题。

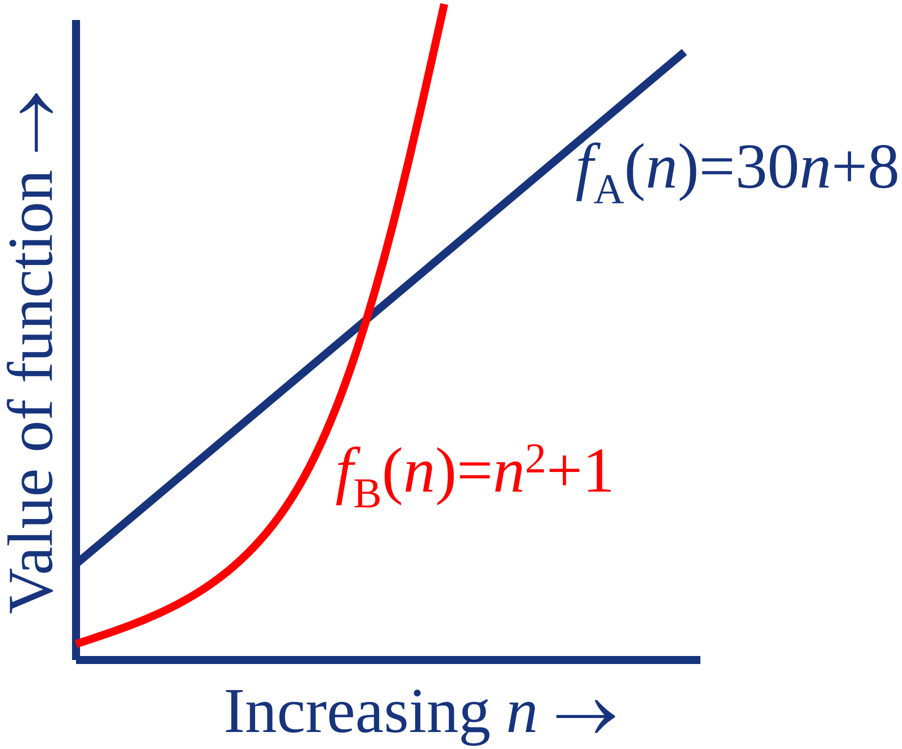
Growth of Functions

Orders of Growth - Motivation

- Suppose you are designing a web site to process user data (*e.g.*, financial records).
- Suppose database program A takes $f_A(n)=30n+8$ microseconds to process any n records, while program B takes $f_B(n)=n^2+1$ 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...



Logo

Concept of order of growth

- **We say $f_A(n)=30n+8$ is *(at most) order n*, or $O(n)$.**
  - **It is, at most, roughly *proportional* to *n*.**
- **$f_B(n)=n^2+1$ is *order* n^2, or $O(n^2)$.**
  - **It is (at most) roughly proportional to n^2.**
- **Any function whose *exact* (tightest) order is $O(n^2)$ is faster-growing than any $O(n)$ function.**
  - **Later we will introduce Θ for expressing *exact* order.**
- **For large numbers of user records, the exactly order n^2 function will always take more time.**

Definition: *O(g)*, *at most order g*

Let g be any function $\mathbf{R}\rightarrow\mathbf{R}$.

- Define "*at most order g*", written $O(g)$, to be: $\{f:\mathbf{R}\rightarrow\mathbf{R} \mid \exists c,k: \forall x>k: f(x) \leq cg(x)\}$.
  - **"Beyond some point *k*, function *f* is at most a constant *c* times *g* (*i.e.*, proportional to *g*)."**
- "*f* is *at most order g*", or "*f* is $O(g)$", or "$f=O(g)$" just means that $f\in O(g)$.
- Often the phrase "at most" is omitted.

Points about the definition

- Note that *f* is O(*g*) so long as *any* values of *c* and *k* exist that satisfy the definition.
- But: The particular *c*, *k*, values that make the statement true are *not* unique: **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!)

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

"Big-O" Proof Examples

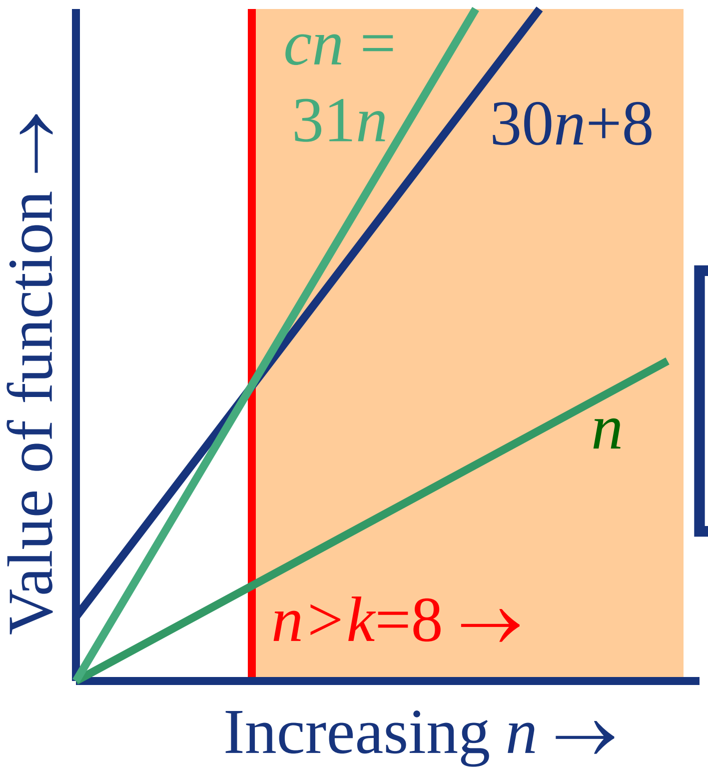
- Show that $30n+8$ is $O(n)$.
  - **Show $\exists c,k$: $\forall n>k$: $30n+8 \le cn$.**

    Let $c=31$, $k=8$. Assume $n>k=8$. Then $cn = 31n = 30n + n > 30n+8$, so $30n+8 < cn$.
- Show that n^2+1 is $O(n^2)$.
  - **Show $\exists c,k$: $\forall n>k$: $n^2+1 \le cn^2$.**

    Let $c=2$, $k=1$. Assume $n>1$. Then $cn^2 = 2n^2 = n^2+n^2 > n^2+1$, or $n^2+1< cn^2$.

Big-O example, graphically

- Note $30n+8$ isn't less than n *anywhere* ($n>0$).
- It isn't even less than $31n$ *everywhere*.
- But it *is* less than $31n$ everywhere to the right of $n=8$.



$\boxed{30n+8 \in O(n)}$

Useful Facts about Big O

- Big O, as a relation, is transitive:
  $f \in O(g) \wedge g \in O(h) \rightarrow f \in O(h)$
- O with constant multiples, roots, and logs...
  $\forall f$ (in $\omega(1)$) & constants $a, b \in \mathbf{R}$, with $b \geq 0$,
  af, f^{1-b}, and $(\log_b f)^a$ are all $O(f)$.
- Sums of functions:
  If $g \in O(f)$ and $h \in O(f)$, then $g+h \in O(f)$.

Logo

More Big-O facts

- $\forall c>0,\ O(cf)=O(f+c)=O(f-c)=O(f)$
- $f_1\in O(g_1) \wedge f_2\in O(g_2) \rightarrow$
  - $f_1 f_2 \in O(g_1 g_2)$
  - $f_1+f_2 \in O(g_1+g_2)$
    $= O(\max(g_1,g_2))$
    $= O(g_1)$ if $g_2\in O(g_1)$
  - (Very useful!)

Orders of Growth - So Far

- For any g:R→R, "*at most order g*", O(g) ≡ {f:R→R | ∃c,k ∀x>k |f(x)| ≤ |cg(x)|}.
  - **Often, one deals only with positive functions and can ignore absolute value symbols.**
- "f∈O(g)" is often written as "f is O(g)" or "f=O(g)".
  - **The latter form is an instance of a more general convention...**

Order-of-Growth Expressions

- "O(f)" when used as a term in an arithmetic expression means: "some function f such that $f \in O(f)$".
- *E.g.*: "$x^2+O(x)$" means "x^2 plus some function that is $O(x)$".
- Formally, you can think of any such expression as denoting a set of functions:
- $x^2+O(x) :\equiv \{g \mid \exists f \in O(x): g(x)= x^2+f(x)\}$

Logo

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" $E_1=E_2$ really means

$$\forall f\in S, \exists g\in T : \quad f=g$$

or simply $S\subseteq T$.

- Example: $x^2 + O(x) = O(x^2)$ means

$$\forall f\in O(x): \exists g\in O(x^2): \; x^2+f(x)=g(x)$$

Logo

Useful Facts about Big O

- $\forall$ *f,g* & constants $a,b \in$ R, with $b \geq 0$,
  - $af = O(f)$; (*e.g.* $3x^2 = O(x^2)$)
  - $f + O(f) = O(f)$; (*e.g.* $x^2 + x = O(x^2)$)
- Also, if $f = \Omega(1)$ (*at least* order 1), then:
  - $|f|^{1-b} = O(f)$; (*e.g.* $x^{-1} = O(x)$)
  - $(\log_b |f|)^a = O(f)$. (*e.g.* $\log x = O(x)$)
  - $f = O(fg)$ (*e.g.* $x = O(x \log x)$)
  - $fg \neq O(g)$ (*e.g.* $x \log x \neq O(x)$)
  - $a = O(f)$ (*e.g.* $3 = O(x)$)

Case

- Let $f(x)=a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0$
- $a_n, a_{n-1}, \ldots, a_1, a_0$ are real number.

$$
\begin{aligned}
|f(x)| &= |a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0| \\
&\le |a_n|x^n+|a_{n-1}|x^{n-1}+\ldots+|a_1|x+|a_0| \\
&= x^n(|a_n|+|a_{n-1}|x^{-1}+\ldots+|a_1|x^{-(n-1)}+|a_0|x^{-n}) \\
&\le x^n(|a_n|+|a_{n-1}|+\ldots+|a_1|+|a_0|) \qquad (x>1)
\end{aligned}
$$

- $f(x)\le Cx^n$ $\qquad C=|a_n|+|a_{n-1}|+\ldots+|a_1|+|a_0|$ $\qquad k=1$
- $f(x)$ **is** $O(x^n)$

Definition: Θ(*g*), *exactly order g*

- If $f \in O(g)$ and $g \in O(f)$, then we say "*g and f are of the same order*" or "*f is (exactly) order g*" and write $f \in \Theta(g)$.
- Another, equivalent definition:
  $\Theta(g) \equiv \{f: \mathbf{R} \rightarrow \mathbf{R} \mid$
  $\exists c_1 c_2 k > 0\ \forall x > k: |c_1 g(x)| \leq |f(x)| \leq |c_2 g(x)|\ \}$
  - "Everywhere beyond some point k, $f(x)$ lies in between two multiples of $g(x)$."

Rules for Θ

- Mostly like rules for O( ), except:
- $\forall$ f,g>0 & constants $a,b \in$ R, with b>0,
  $af \in \Theta(f)$, but ← Same as O.
  $f \notin \Theta(fg)$ unless $g=\Theta(1)$ ← Unlike O.
  $|f|^{1-b} \notin \Theta(f)$, and ← Unlike O.
  $(\log_b |f|)^c \notin \Theta(f)$. ← Unlike O.
- The functions in the latter two cases we say are *strictly of lower order* than $\Theta(f)$.

Θ example

- Determine whether:
- Quick solution:

$$\left(\sum_{i=1}^{n} i\right) \in \Theta(n^2)$$

$$\left(\sum_{i=1}^{n} i\right) = n(n-1)/2$$

$$= n \cdot \Theta(n)/2$$

$$= n \cdot \Theta(n)$$

$$= \Theta(n^2)$$

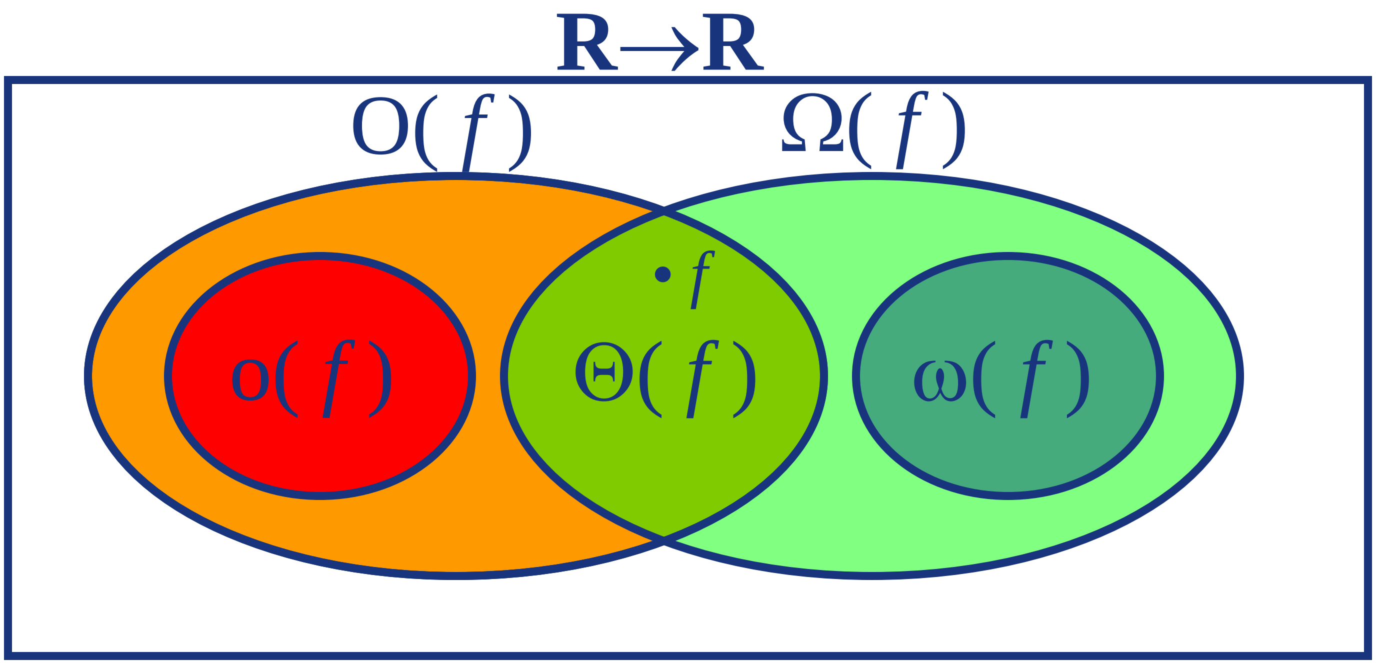
Logo

Other Order-of-Growth Relations

- $\Omega(g) = \{f \mid g \in O(f)\}$
  "The functions that are *at least order g*."
- $o(g) = \{f \mid \forall c>0\ \exists k\ \forall x>k : |f(x)| < |cg(x)|\}$
  "The functions that are *strictly lower order than g*." $o(g) \subset O(g) - \Theta(g)$.
- $\omega(g) = \{f \mid \forall c>0\ \exists k\ \forall x>k : |cg(x)| < |f(x)|\}$
  "The functions that are *strictly higher order than g*." $\omega(g) \subset \Omega(g) - \Theta(g)$.

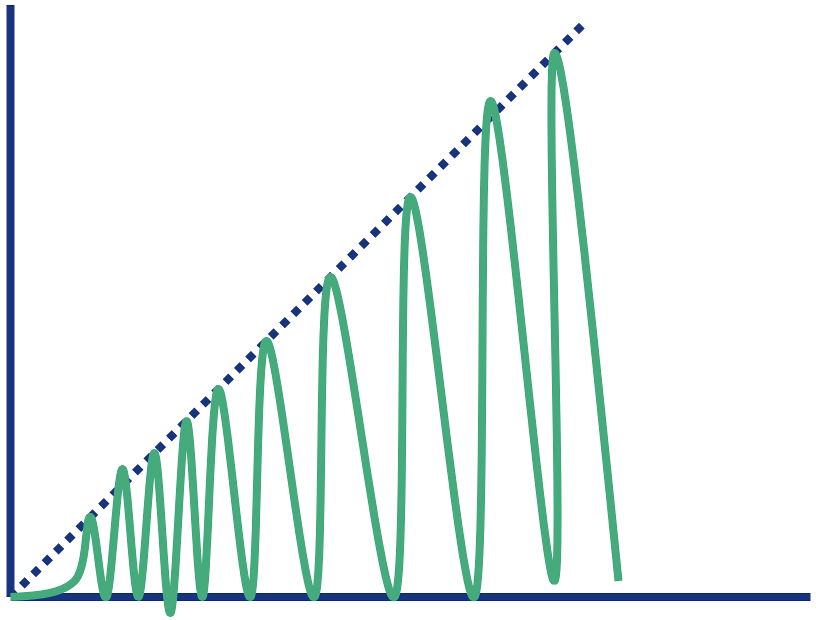
Relations Between the Relations

- Subset relations between order-of-growth sets.



Why o(f)⊂O(x)−Θ(x)

- A function that is O(x), but neither o(x) nor Θ(x):



Logo

Strict Ordering of Functions

- Temporarily let's write $f \prec g$ to mean $f \in o(g)$, $f \sim g$ to mean $f \in \Theta(g)$
- Note that:

$$f \prec g \Leftrightarrow \lim_{x \to \infty} \frac{f(x)}{g(x)} = 0.$$

- Let $k>1$. Then the following are true:
  $1 \prec \log \log n \prec \log n \sim \log_k n \prec \log^k n \prec n^{1/k} \prec n \prec n \log n \prec n^k \prec k^n \prec n! \prec n^n \ldots$

Logo

Review: Orders of Growth

Definitions of order-of-growth sets, $\forall g{:}\mathbf{R}\rightarrow\mathbf{R}$

- $O(g) :\equiv \{f \mid \exists\, c>0\ \exists k\ \forall x>k\ |f(x)| \le |cg(x)|\}$
- $o(g) :\equiv \{f \mid \forall c>0\ \exists k\ \forall x>k\ |f(x)| < |cg(x)|\}$
- $\Omega(g) :\equiv \{f \mid g \in O(f)\}$
- $\omega(g) :\equiv \{f \mid \forall c>0\ \exists k\ \forall x>k : |cg(x)| < |f(x)|\}$
- $\Theta(g) :\equiv O(g) \cap \Omega(g)$

Applications

Quick proof

- **Quick proof:**
- **1.** $n! = \omega\left(2^n\right)$
- **2.** $n! = o\left(n^n\right)$

Quick proof

- **Quick proof:**
- **1.** $n! = \omega\left(2^n\right)$
- **2.** $n! = o\left(n^n\right)$

These simple examples show you the way to prove it.

Solution:

1. $\lim\limits_{n\to\infty} \dfrac{n!}{2^n} = \infty \therefore \; n! = \omega\left(2^n\right)$

2. $\lim\limits_{n\to\infty} \dfrac{n!}{n^n} = 0 \therefore \; n! = o\left(n^n\right)$

Time Complexity

What is *complexity*?

- The word *complexity* has a variety of different technical meanings in different research fields.
- There is a field of *complex systems*, which studies complicated, difficult-to-analyze *non-linear* and *chaotic* natural & artificial systems.
- Another concept: *Informational or descriptional complexity*: The amount of *information* needed to completely describe an object.
  - As studied by Kolmogorov, Chaitin, Bennett, others…
- In this course, we will study *algorithmic or computational complexity*.

§ 2.3: Algorithmic Complexity

- The *algorithmic complexity* of a computation is, most generally, a measure of how *difficult* it is to perform the computation.
- That is, it measures some aspect of the *cost* of computation (in a general sense of "cost").
  - Amount of resources required to do a computation.
- Some of the most common complexity measures:
  - "Time" complexity: # of operations or steps required
  - "Space" complexity: # of memory bits req'd

Our focus

Complexity Depends on Input

- Most algorithms have different complexities for inputs of different sizes.
  - ***E.g.* searching a long list typically takes more time than searching a short one.**
- Therefore, complexity is usually expressed as a *function* of the input length.
  - **This function usually gives the complexity for the *worst-case* input of any given length.**

Complexity & Orders of Growth

- Suppose algorithm A has worst-case time complexity (w.c.t.c., or just *time*) $f(n)$ for inputs of length n, while algorithm B (for the same task) takes time $g(n)$.
- Suppose that $f \in \omega(g)$, also written $\boxed{f \succ g}$.
- Which algorithm will be *fastest* on all sufficiently-large, worst-case inputs?

Example 1: Max algorithm

- **Problem:** Find the *simplest form* of the *exact* order of growth (Θ) of the *worst-case* time complexity (w.c.t.c.) of the *max* algorithm, assuming that each line of code takes some constant time every time it is executed (with possibly different times for different lines of code).

Complexity analysis of *max*

procedure $max(a_1, a_2, \ldots, a_n\text{: integers})$

$v := a_1$ t_1

for $i := 2$ **to** n t_2

if $a_i > v$ **then** $v := a_i$ t_3

return v t_4

Times for *each* execution of each line.

First, what's an expression for the *exact* total worst-case time? (Not its order of growth.)

Complexity analysis, cont.

procedure $max(a_1, a_2, \ldots, a_n$: integers)

$v := a_1$ — t_1

for $i := 2$ **to** n — t_2

if $a_i > v$ **then** $v := a_i$ — t_3

return v — t_4

Times for *each* execution of each line.

w.c.t.c.:

$$\boxed{t(n) = t_1 + \left(\sum_{i=2}^{n} (t_2 + t_3) \right) + t_4}$$

Complexity analysis, *cont.*

Now, what is the simplest form of the exact (Θ) order of growth of *t*(*n*)?

$$t(n) = t_1 + \left(\sum_{i=2}^{n} (t_2 + t_3) \right) + t_4$$

$$= \Theta(1) + \left(\sum_{i=2}^{n} \Theta(1) \right) + \Theta(1) = \Theta(1) + (n-1)\Theta(1)$$

$$= \Theta(1) + \Theta(n)\Theta(1) = \Theta(1) + \Theta(n) = \Theta(n)$$

Example 2: Linear Search

procedure *linear search* (x: integer,
$a_1, a_2, \ldots, a_n$: distinct integers)

$\boldsymbol{i := 1}$	t_1
while $\boldsymbol{(i \leq n \wedge x \neq a_i)}$	t_2
$\quad \boldsymbol{i := i + 1}$	t_3
if $\boldsymbol{i \leq n}$ **then** $\boldsymbol{location := i}$	t_4
else $\boldsymbol{location := 0}$	t_5
return ***location***	t_6

Logo

Linear search analysis

- Worst case time complexity order:

$$t(n) = t_1 + \left(\sum_{i=1}^{n} (t_2 + t_3) \right) + t_4 + t_5 + t_6 = \Theta(n)$$

- **Best case:**

$$t(n) = t_1 + t_2 + t_4 + t_6 = \Theta(1)$$

- **Average case, if item is present:**

$$t(n) = t_1 + \left(\sum_{i=1}^{n/2} (t_2 + t_3) \right) + t_4 + t_5 + t_6 = \Theta(n)$$

Review §2.3: Complexity

- Algorithmic complexity = *cost* of computation.
- Focus on *time* complexity for our course.
  - Although space & energy are also important.
- Characterize complexity as a function of input size: **Worst-case, best-case, or average-case.**
- Use orders-of-growth notation to concisely summarize the growth properties of complexity functions.

Logo

Example 3: Binary Search

procedure ***binary search*** **(*x*:integer, $a_1, a_2, \ldots,$ a_n: distinct integers, sorted smallest to largest)**

$i := 1$
$j := n$ $\Bigg\}\ \Theta(1)$

while $i<j$ begin

$m := \lfloor (i+j)/2 \rfloor$
if $x>a_m$ then $i := m+1$ else $j := m$ $\Bigg\}\ \Theta(1)$

end

if $x = a_i$ then *location* $:= i$ else *location* $:= 0$
return *location* $\Bigg\}\ \Theta(1)$

Key question:

How many loop iterations?

Binary search analysis

- Suppose that n is a power of 2, *i.e.*, $\exists k$: $n=2^k$.
- Original range from $i=1$ to $j=n$ contains n items.
- Each iteration: Size $j-i+1$ of range is cut in ~half.
- Loop terminates when size of range is $1=2^0$ ($i=j$).
- Therefore, the number of iterations is:

$$k = \log_2 n = \Theta(\log_2 n) = \Theta(\log n)$$

- Even for $n \neq 2^k$ (not an integral power of 2), time complexity is still $\Theta(\log_2 n) = \Theta(\log n)$.

Logo

Time Complexity of Algorithm

Logo

Names for some orders of growth

- $\Theta(1)$ Constant
- $\Theta(\log_c n)$ Logarithmic (same order $\forall c$)
- $\Theta(\log^c n)$ Polylogarithmic
- $\Theta(n)$ Linear
- $\Theta(n^c)$ Polynomial (for any c)
- $\Theta(c^n)$ Exponential (for $c>1$)
- $\Theta(n!)$ Factorial

(With c a constant.)

Logo

Review §2.3: Complexity

- Algorithmic complexity = *cost* of computation.
- Focus on *time* complexity for our course.
  - Although space & energy are also important.
- Characterize complexity as a function of input size: Worst-case, best-case, or average-case.
- Use orders-of-growth notation to concisely summarize the growth properties of complexity functions.

Logo

Problem Complexity

- The complexity of a computational *problem* or *task* is (the order of growth of) the complexity of the algorithm with the lowest order of growth of complexity for solving that problem or performing that task.
- ***E.g.* the problem of searching an ordered list has *at most logarithmic* time complexity. (Complexity is O(log *n*).)**

Tractable *vs.* intractable

- **A problem or algorithm with at most polynomial time complexity is considered *tractable* (or *feasible*). P is the set of all tractable problems.**
- **A problem or algorithm that has complexity greater than polynomial is considered *intractable* (or *infeasible*).**
- **Note that $n^{1,000,000}$ is *technically* tractable, but really very hard. $n^{\log \log \log n}$ is *technically* intractable, but easy. Such cases are rare though.**

Logo

Computer Time Examples

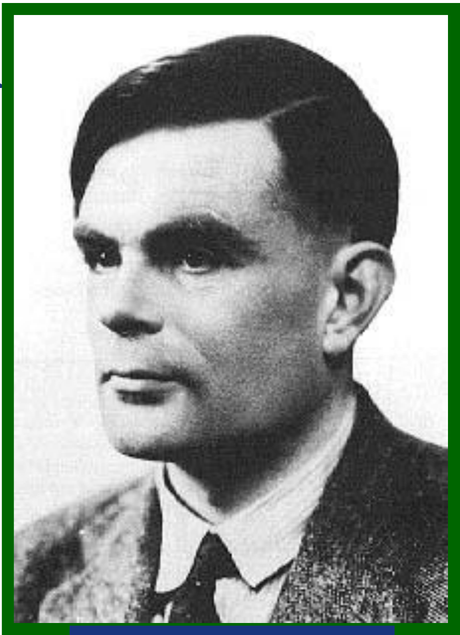
#ops(n)	(1.25 bytes) $n=10$	(125 kB) $n=10^6$
$\log_2 n$	3.3 ns	19.9 ns
n	10 ns	1 ms
$n \log_2 n$	33 ns	19.9 ms
n^2	100 ns	16 m 40 s
2^n	1.024 μs	$10^{301,004.5}$ Gyr
$n!$	3.63 ms	Ouch!

Assume time = 1 ns (10^{-9} second) per op, problem size = *n* bits, and #ops is a function of *n*, as shown.

Unsolvable problems

- Turing discovered in the 1930's that there are problems unsolvable by *any* algorithm.
  - **Or equivalently, there are undecidable yes/no questions, and uncomputable functions.**
- Classic example: the *halting problem*.
  - **Given an arbitrary algorithm and its input, will that algorithm eventually halt, or will it continue forever in an "*infinite loop*?"**

The Halting Problem (Turing'36)



Alan Turing
1912-1954

- The *halting problem* was the first mathematical function proven to have *no* algorithm that computes it!
  - We say, it is *uncomputable.*
- The desired function is *Halts*(*P*,*I*) :≡ the truth value of this statement:
  - *"Program P, given input I, eventually terminates."*
- **Theorem:** *Halts* is uncomputable!
  - *I.e.*, there does *not* exist *any* algorithm *A* that computes *Halts* correctly for *all* possible inputs.
- Its proof is thus a *non*-existence proof.
- **Corollary:** General impossibility of predictive analysis of arbitrary computer programs.

Proving the Theorem

of the Undecidability of the Halting Problem

- Given any *arbitrary* program *H*(*P*,*I*),
- Consider algorithm *Foiler*, defined as:

procedure *Foiler*(*P*: a program)
    halts **:=** *H*(*P*,*P*)
    if *halts* **then** loop forever

Foiler makes a *liar* out of *H*, by simply doing the *opposite* of whatever *H* predicts it will do!

- Note that *Foiler*(*Foiler*) halts iff *H*(*Foiler*,*Foiler*) = **F**.
- So *H* does **not** compute the function *Halts*!

P *vs.* NP

- **NP** is the set of problems for which there exists a tractable algorithm for *checking a proposed solution* to tell if it is correct.
- We know that **P$\subseteq$NP**, but the most famous unproven conjecture in computer science is that this inclusion is *proper*.
  - *i.e.*, that **P$\subset$NP** rather than **P=NP**.
- Whoever first proves this will be famous!

(or disproves it!)

Key Things to Know

- Definitions of algorithmic complexity, time complexity, worst-case time complexity.
- Names of specific orders of growth of complexity.
- How to analyze the worst case, best case, or average case order of growth of time complexity for simple algorithms.

Exercise

Logo

Exercises

- 1. Function f is defined as $f: Z \rightarrow Z, f(x) = |x| - 4x$, so f is ( )
- A. onto
- B. one-to-one
- C. both onto and one-to-one
- D. neither onto nor one-to-one

Exercises

- 1. Function f is defined as $f: Z \to Z, f(x) = |x| - 4x$, so f is (B)
- A. onto
- B. one-to-one
- C. both onto and one-to-one
- D. neither onto nor one-to-one

Exercises

- 2. Function f is defined as $f: Z \times Z \to Z$. Which function is not onto? (   )
- A) $f(m,n) = m + n$
- B) $f(m,n) = m^2 + n^2$
- C) $f(m,n) = m$
- D) $f(m,n) = m - n$

Exercises

- 2. Function f is defined as $f: Z \times Z \to Z$. Which function is not onto? (B)
- A) $f(m,n) = m + n$
- B) $f(m,n) = m^2 + n^2$
- C) $f(m,n) = m$
- D) $f(m,n) = m - n$

Exercises

- 3. Which functions are onto from Z to Z?(   )
- A) $f(n) = n^3$.
- B) $f(n) = n^2 + 1$.
- C) $f(n) = n^3 + n^2 + 1$.
- D) $f(n) = n - 1$.

Exercises

- 3. Which functions are onto from Z to Z?(D)
- A) $f(n) = n^3$.
- B) $f(n) = n^2 + 1$.
- C) $f(n) = n^3 + n^2 + 1$.
- D) $f(n) = n - 1$.

Exercises

- 4. Which of the following functions is a bijection from *R* to *R*. (　)
- A） $f(x)=1/x$
- B） $f(x)=-3x^2+7$
- C） $f(x)=(x+1)/(x+2)$
- D） $f(x)=x^5+1$

Exercises

- 4. Which of the following functions is a bijection from R to R. (D)
- A） $f(x)=1/x$
- B） $f(x)=-3x^2+7$
- C） $f(x)=(x+1)/(x+2)$
- D） $f(x)=x^5+1$

Exercises

- 5. Suppose that: “f is a function mapping students in this class to the set of grades {1, 2, 3 ,4, 5}.” Moreover, the grades turn out all 3 and 4. Then the range of f is （　　）
- A) {1, 2, 3 ,4, 5}
- B) {3 ,4}
- C) {3 ,4} or {1, 2, 3 ,4, 5}
- D) unknown

Exercises

- 5. Suppose that: “f is a function mapping students in this class to the set of grades {1, 2, 3 ,4, 5}.” Moreover, the grades turn out all 3 and 4. Then the range of f is （ B ）
- A) {1, 2, 3 ,4, 5}
- B) {3 ,4}
- C) {3 ,4} or {1, 2, 3 ,4, 5}
- D) unknown

Exercises

- 7. Let A = {a, b, c, d, e} and B = {a, b, c, d, e, f, g, h}. B ∩ A=_______

Exercises

- 7. Let A = {a, b, c, d, e} and B = {a, b, c, d, e, f, g, h}. $B \cap A$=_______

- {a,b,c,d,e}

Exercises

- 8. Let $f(x) = \lfloor x^2/3 \rfloor$, S={−2, −1, 0, 1, 2, 3}, f(S)=________

Exercises

- 8. Let $f(x) = \lfloor x^2/3 \rfloor$, S={－2,－1, 0, 1, 2, 3}, f(S)=________

- {0, 1, 3}

Exercises

- 9. R is the real number domain. For $\forall x \in R$, $f(x) = x + 2$, $g(x) = x - 2$ and $h(x) = 3x$. Hence, $h \circ (g \circ f)$=______

Exercises

- 9. R is the real number domain. For $\forall x \in R$, $f(x) = x + 2$, $g(x) = x - 2$ and $h(x) = 3x$. Hence, $h \circ (g \circ f)$=______

- $3x$

Exercises

- 10. R is the real number domain. For $\forall x \in R$, f (x) =sinx, g (x) =x^2, and $h(x) = 3x$. Hence, f ∘ g ∘ h (x) =______

Exercises

- 10. R is the real number domain. For $\forall x \in R$, f (x) =sinx, g (x) =x^2, and $h(x) = 3x$. Hence, f∘g∘h (x) =______

- sin ($9x^2$)

Exercises

- 11. Suppose that $f(\mathrm{x})=x^2+1$ and $g(x)=x+2$ are functions from R to R. $f+g=$______

Exercises

- 11. Suppose that $f(x)=x^2+1$ and $g(x)=x+2$ are functions from R to R. $f+g=$______

- x^2+x+3

Exercises

- 12. $|\{f:\{0,1\}^n \to \{0,1\}^n\}|=$______

Logo

Exercises

- 12. $|\{f : \{0,1\}^n \to \{0,1\}^n\}| =$ ______

$$(2^n)^{2^n}$$

Logo

Exercises

- 13. Let $f(x) = \lfloor x^2/3 \rfloor$, S={0,1,2,3,4,5}, f(S)=$\underline{\{\quad\}}$

Exercises

- 13. Let $f(x) = \lfloor x^2/3 \rfloor$, S={0,1,2,3,4,5}, f(S)={__}

- {0,1,3,5,8}

Exercises

- 14. When y is a constant independent of x, what is the θ( ) complexity of the function $f(x) = (x^2 + y \ln x)^2$?

Logo

Exercises

- 14. When y is a constant independent of x, what is the θ( ) complexity of the function $f(x) = (x^2 + y \ln x)^2$?

- $\Theta(x^4)$

Logo

Exercises

- 16. Let f be a bijection function from A to B. Let S and T be subsets of B. Show that:
- $f^{-1}(S \cap T) = f^{-1}(S) \cap f^{-1}(T)$.

Exercises

- 16. Let f be a bijection function from A to B. Let S and T be subsets of B. Show that:
- $f^{-1}(S \cap T) = f^{-1}(S) \cap f^{-1}(T)$.

$f^{-1}(S \cap T) \Rightarrow f^{-1}(S) \cap f^{-1}(T)$

对任意$x \in f^{-1}(S \cap T)$，有
$f(x) \in S \cap T$
$\Rightarrow f(x) \in S \cap f(x) \in T$
$\Rightarrow x \in f^{-1}(S) \cap x \in f^{-1}(T)$

$f^{-1}(S) \cap f^{-1}(T) \Rightarrow f^{-1}(S \cap T)$

对任意$x \in f^{-1}(S) \cap f^{-1}(T)$，有
$x \in f^{-1}(S) \cap x \in f^{-1}(T)$
$\Rightarrow f(x) \in S \cap f(x) \in T$
$\Rightarrow f(x) \in S \cap T$
$\Rightarrow x \in f^{-1}(S \cap T)$

Exercises

- 18. Please prove that the time complexity of $x^2+4x+17$ is $O(x^3)$, but the complexity of x^3 is not $O(x^2+4x+17)$

Exercies

- 18.

$x^2 + 4x + 17 \leq 3x^3$ for all $x > 17$, so $x^2 + 4x + 17$ is $O(x^3)$, with witnesses $C = 3$, $k = 17$. However, if x^3 were $O(x^2 + 4x + 17)$, then $x^3 \leq C(x^2 + 4x + 17) \leq 3Cx^2$ for some C, for all sufficiently large x, which implies that $x \leq 3C$ for all sufficiently large x, which is impossible. Hence, x^3 is not $O(x^2 + 4x + 17)$.

Logo

End of the Section 2.2