algorithms and data structures

amir yehudayoff

mathematics

definitions who?
we declare the assumptions or “the rules of the game”
what are we talking about

theorems what?
what properties hold

proofs why?
we explain (using the rules of logic) how does the claim follows from the definitions
algorithmic

notation: numbers

the natural numbers

$$\mathbb{N} = \{1, 2, 3, \ldots\}$$

the integers

$$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, 3, \ldots\}$$

the rationals

$$\mathbb{Q} = \{\tfrac{a}{b} : a, b \in \mathbb{Z}, b \neq 0\}.$$

the real numbers $\mathbb{R}$

natural inclusions

$$\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$$

rounding

$$\mathbb{R} \to \mathbb{Z}$$
$$\mathbb{R}^2 \to \mathbb{Z}^2$$

for $x \in \mathbb{R}$

$$\lceil x \rceil = \min\{z \in \mathbb{Z} : z \geq x\}$$

$$\lfloor x \rfloor = \max\{z \in \mathbb{Z} : z \leq x\}$$

quantifiers

$$\forall, \exists$$

example for every positive number $\equiv \forall \epsilon > 0$

example

definition $z \in \mathbb{Z}$ is even if there is $a \in \mathbb{Z}$ so that $z = 2a$

$$\exists a \in \mathbb{Z}\ z = 2a$$

theorem $\forall z \in \mathbb{Z}, \forall k \in \mathbb{Z}$ if z is even then kz is even

proof...

example

definitions

let X be a finite set

a probability distribution is $p : X \to [0, 1]$ such that

$$\sum_{x \in X} p(x) = 1$$

a random variable is $f : X \to \mathbb{R}$

the expected value of f is

$$\mathbb{E}[f] = \sum_{x \in X} p(x) f(x)$$

(a dice)

example

definitions
a probability distribution is ...
a random variable is ...
the expected value is ...

theorem for every two random variables $f, g : X \to \mathbb{R}$

$$\mathbb{E}[f + g] = \mathbb{E}[f] + \mathbb{E}[g]$$

proof...

contra-positive

$$A \Rightarrow B \text{ equivalent to } \neg B \Rightarrow \neg A$$

example

$X, Y \subset Z$

$X \subset Y$ means $x \in X \Rightarrow x \in Y$

$X \subset Y$ equivalent to $(Z \setminus Y) \subset (Z \setminus X)$

contra-positive

$$A \Rightarrow B \text{ equivalent to } \neg B \Rightarrow \neg A$$

proof by contradiction

theorem $\sqrt{2} \notin \mathbb{Q}$

proof if $\sqrt{2} \in \mathbb{Q}$ .... a contradiction

induction: example

theorem for every integer $n \in \mathbb{N}$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

induction: example

theorem for every integer $n \in \mathbb{N}$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

proof

base: $1 = \frac{1 \cdot 2}{2}$

step: assume that true for n (that is, $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$)

then true for $n+1$

$$\sum_{i=1}^{n+1} i = n+1+\sum_{i=1}^{n} i = n+1+\frac{n(n+1)}{2} = \frac{(n+1)(n+2)}{2}$$

induction: general

property $P \subseteq \mathbb{N}$

goal: prove $P = \mathbb{N}$

proofs by induction have two parts

(base) $1 \in P$

(step) for every $n \in \mathbb{N}$ if $n \in P$ then $n + 1 \in P$

strong induction

induction step

$$n \in P \Rightarrow n+1 \in P$$

strong induction step

$$\{1, 2, \ldots, n\} \subset P \Rightarrow n+1 \in P$$

example

definition a natural number $p > 1$ is prime if for every $a, b \in \mathbb{N}$ if $p = a \cdot b$ then $a = 1$ or $b = 1$

example

definition a natural number $p > 1$ is prime if for every $a, b \in \mathbb{N}$ if $p = a \cdot b$ then $a = 1$ or $b = 1$

theorem for every integer $n > 1$, there are primes $p_1, \ldots, p_k$ such that

$$n = \prod_{i=1}^{k} p_i$$

example

definition a natural number $p > 1$ is prime if for every $a, b \in \mathbb{N}$ if $p = a \cdot b$ then $a = 1$ or $b = 1$

theorem for every integer $n > 1$, there are primes $p_1, \ldots, p_k$ such that

$$n = \prod_{i=1}^{k} p_i$$

proof sketch
if n is prime then we are done
otherwise, $n = ab$ with $a, b < n$
by induction, both a, b can be factored

induction: algorithms

we can define a function $F : \mathbb{N} \to \mathbb{R}$ by induction (recursion)

for example

$$F(1) = F(2) = 1$$
$$F(n) = F(n-1) + F(n-2) \qquad n > 2$$

remark

this is a recursive algorithm for F

what is running time? is it good?

remark: can compute $F(n)$ in time $O(\log n)$

induction: analysis of algorithms

induction allows to keep track of “invariants”—properties that help us to understand the algorithm

induction: analysis of algorithms

induction allows to keep track of “invariants”—properties that help us to understand the algorithm

bubble sort

the input is an array $A = (A[1], \ldots, A[n])$ of n numbers
the output should be the array in sorted order

for $i = 1, 2, \ldots, n-1$:
  for $j = 1, 2, \ldots, n-i$:
    if $A[j] > A[j+1]$ then
      swap $A[j]$ and $A[j+1]$

bubble sort

for $i = 1, 2, \ldots, n-1$:
  for $j = 1, 2, \ldots, n-i$:
    if $A[j] > A[j+1]$ then
      swap $A[j]$ and $A[j+1]$

notation
there are two loops
the loop over i is the "outer loop"
the loop over j is the "inner loop"

denote by A^t the state of the array just after the inner loop finished with $i = t$ where A^0 is the original array

correctness

claim for every $t = 0, 1, 2, \ldots, n-1$

$$A^t[n] \geq A^t[n-1] \geq \ldots \geq A^t[n-t+1] \geq \max\{A^t[s] : s \leq n-t\}$$

in words, at time t the last t positions are ordered and largest

correctness

claim for every $t = 0, 1, 2, \ldots, n-1$

$$A^t[n] \geq A^t[n-1] \geq \ldots \geq A^t[n-t+1] \geq \max\{A^t[s] : s \leq n-t\}$$

in words, at time t the last t positions are ordered and largest

proof

the proof is by induction
the base case is $t = 0$
the condition trivially holds (nothing should be satisfied)

correctness

claim for every $t = 0, 1, 2, \ldots, n-1$

$$A^t[n] \geq A^t[n-1] \geq \ldots \geq A^t[n-t+1] \geq \max\{A^t[s] : s \leq n-t\}$$

proof

inductive step: A^{t+1} is constructed from A^t via

```
for j = 1, 2, ..., n − t − 1:
    if A[j] > A[j + 1] then
        swap A[j] and A[j + 1]
```

let j_* be

$$j_* = \arg\max\{A^t[s] : s \leq n-t\}$$

(if there are several options, choose the maximum one)

correctness

claim for every $t = 0, 1, 2, \ldots, n-1$

$$A^t[n] \geq A^t[n-1] \geq \ldots \geq A^t[n-t+1] \geq \max\{A^t[s] : s \leq n-t\}$$

proof

for $j = 1, 2, \ldots, n-t-1$:

  if $A[j] > A[j+1]$ then

    swap $A[j]$ and $A[j+1]$

let $j_* = \arg\max\{A^t[s] : s \leq n-t\}$

the inner loop for $j = j_*, \ldots, n-t-1$ moves $A^t[j_*]$ from position j_* to position $n-t$

the part of the array in position larger than $n-t$ is unchanged

$$A^{t+1}[n] \geq \ldots \geq A^{t+1}[n-t+1] \geq A^{t+1}[n-t] \geq \max\{A^t[s] : s \leq n-t-1\}$$

running time

for $i = 1, 2, \ldots, n-1$:
  for $j = 1, 2, \ldots, n-i$:
    if $A[j] > A[j+1]$ then
      swap $A[j]$ and $A[j+1]$

claim 1

for every A the number of swap operations is at most $\frac{n(n-1)}{2}$

claim 2

for every n there is A of size n such that the number of swap operations is $\frac{n(n-1)}{2}$

summary

basic language of math

proofs and induction

algorithms: invariants and running time

what makes an algorithm good?

what makes an algorithm good?

— correctness
— efficiency (time, space, energy, etc.)
— simplicity
— flexibility
— robustness (to “noise”)
— privacy

asymptotics

computational costs depend on the input

the size of the input x is $|x| = n$
e.g. integers, graphs, real numbers

as n grows, the costs grow

we want a clean language

running time

denote running time[1] on input x by $T(x)$

the running time on inputs of size n is

$$T(n) = \max\{T(x) : |x| \leq n\}$$

e.g. linear, quadratic, exponential

[1] number of steps

doubling

if input size doubles, what happens to running time?

for

$$T(n) = \log_2(n), n, n^2, 2^n$$

different answers

$$T(2n) = T(n) + 1, 2T(n), 4T(n), (T(n))^2$$

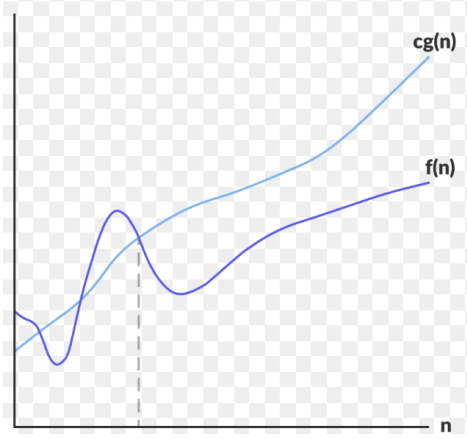
notation

for the rest of this lecture $f, g : \mathbb{N} \to [0, \infty)$

think of $f(n)$ as "cost for inputs of size n"

remark we do not assume f, g are monotone

big O



definition

we write $f(n) \leq O(g(n))$ if there exist $C, N > 0$ such that for all $n > N$

$$f(n) \leq Cg(n)$$

remark $f \leq O(g)$ or $f(n) \in O(g(n))$ or $f(n) = O(g(n))$

big O

examples

$n + 10 \leq O(n)$
$n^2 + n \leq O(n^2)$
$2\sin(n) \leq O(1)$

non-examples

$n + 10 \not\leq O(\sqrt{n})$
$n^2 + n \not\leq O(n \log_2(n))$
$\log_2(n) \not\leq O(\log_2(1 + \log_3(n)))$

example: bubble sort

for $i = 1, 2, \ldots, n-1$:
  for $j = 1, 2, \ldots, n-i$:
    if $A[j] > A[j+1]$ then
      swap $A[j]$ and $A[j+1]$

claim

the number of swaps for arrays of size n is $O(n^2)$

big Ω

definition

we write $f(n) \geq \Omega(g(n))$ if there exist $c, N > 0$ such that for all $n > N$

$$f(n) \geq cg(n)$$

claim $f \geq \Omega(g)$ iff $g \leq O(f)$

e.g. $n^2 \geq \Omega(?)$

big Θ

definition

we write $f(n) = \Theta(g(n))$ if

$$f(n) \leq O(g(n)) \text{ and } f(n) \geq \Omega(g(n))$$

e.g. $\sqrt{n} + 100 + \frac{n^2}{3} + n = \Theta(?)$

summary

$f \leq O(g)$

$f \geq \Omega(g)$

$f = \Theta(g)$

example: factorial

inductively define $f(1) = 1$ and $f(n) = n \cdot f(n-1)$ for $n > 1$

stated differently

$$f(n) = n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n = \prod_{i=1}^{n} i$$

example: factorial

claims $n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$

1. $n! \leq O(n^n)$

example: factorial

claims $n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$

1. $n! \leq O(n^n)$

2. $n! \leq O(2^{-n/2} n^n)$

example: factorial

claims $n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$

1. $n! \leq O(n^n)$
2. $n! \leq O(2^{-n/2} n^n)$
3. $n! \geq \Omega(4^{-n} n^n)$

example: factorial

claims $n! = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$

1. $n! \leq O(n^n)$
2. $n! \leq O(2^{-n/2} n^n)$
3. $n! \geq \Omega(4^{-n} n^n)$
4. $n! = \Theta\left(\sqrt{n}(n/e)^n\right)$

binary search

input: $a_1 \leq a_2 \leq \ldots \leq a_n$ and x

output: $i \in [n]$ such that $a_i = x$ or "no"

algorithm: compare x to $a_{\lceil n/2 \rceil}$ and recurse

binary search

denote by $T(n)$ the running time for n

$$T(1) \leq C$$
$$T(n) \leq T(\lceil n/2 \rceil) + C$$

where $C > 0$ is a constant that depends on "computer"

claim $T(n) \leq O(\log n)$

remark do not write $T(1) \leq O(1)$

binary search

denote by $T(n)$ the running time for n

$$T(1) \leq C$$
$$T(n) \leq T(\lceil n/2 \rceil) + C$$

where $C > 0$ is a constant that depends on "computer"

claim $T(n) \leq O(\log n)$

remark do not write $T(1) \leq O(1)$

claim $T(n) \leq 10C \log_2(n)$ for $n \geq 2$

properties

exercise

if $f_1 \leq O(g_1)$ and $f_2 \leq O(g_2)$ then

$$f_1 + f_2 \leq O(g_1 + g_2)$$

and

$$f_1 \cdot f_2 \leq O(g_1 \cdot g_2)$$

(similarly for Ω, Θ)

remarks

— false for f/g and $f - g$
— transitivity
— cannot use for $f_i \leq O(g_i)$ for $i = 1, \ldots, n$

small o

definition

we write $f(n) \leq o(g(n))$ if for every $\epsilon > 0$ there exists $N > 0$ such that for all $n > N$

$$f(n) \leq \epsilon g(n)$$

remark if $g > 0$ equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

examples $n \leq o(n^2)$ and $\frac{1}{n} \leq o(1)$

small ω

definition

we write $f(n) \geq \omega(g(n))$ if for every $K > 0$ there exists $N > 0$ such that for all $n > N$

$$f(n) \geq Kg(n)$$

remark if $g > 0$ equivalent to $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

examples $n^3 \geq \omega(n^2)$ and $\log(n) \geq \omega(1)$

one

$o(1)$ means “small”

$\omega(1)$ means “large”

higher dimensions

now $f, g : \mathbb{N}^2 \to [0, \infty)$

definition

we write $f(n, m) \leq O(g(n, m))$ if there exists $C > 0$ and $N > 0$ such that for all n, m such that $(n, m) \notin [N] \times [N]$

$$f(n, m) \leq Cg(n, m)$$

remarks

— there are a few options for a definition here
— similarly for $\Omega, \Theta, o, \omega$
— higher dimensions

summary

costs of algorithms

asymptotic notation