Algorithm

A sequence of instructions that describes unambiguously how to solve a problem in finite amount of time.

- They have existed for a long time
  - Division: Babylonia 2500 BC, Egypt 1550 BC
  - Finding primes, greatest common divisor - Greece 240 BC
  - Even code breaking (cryptography) (9th century)

- "Algorithm" ← "Algorism" ← "al-Khowarizmi" Arabic math. 9th century

1 Thank you, Wikipedia

Questions Related to Algorithms

- How do we specify one?

  English, pseudocode \( x := expr(asgt) \)

  code (language)?

- Does the algorithm always give the correct answer?

  correctness

  proofs! - informal, for us

  (Arrays of \( a \) and \( b \)) (Array \( b \)) (\( b \) 'permutation of \( b \)) (\( b \) 'is sorted)
Questions Related to Algorithms

- How long does it take?

  \[ \text{Time complexity} \]

  \[ \text{Space complexity} \]

- How much memory does it take?

- Does it complete in a reasonable amount of time?

  \[ \text{tractability} \quad P = \text{NP} \]

Questions Related to Algorithms

- How do we design them?

  \[ \text{Algorithmic paradigms} \]

  Divide & conquer

  Brute force

  Greedy

- Does an algorithm even exist?

  Solvable? Undecidable? = unsolvable
Example: `max` of two numbers `x`, `y`:

```python
def max(x, y):
    if x >= y
        return x
    else
        return y
```

Use proof by cases to show it's correct:

If `r` is the returned value, we want:

\[
(r = x \lor r = y) \land (r \geq x \land r \geq y)
\]

Does it matter if we say if `x \geq y` or if `x > y`?

Example: Find `max` of a sequence of values `a_1`, `a_2`, ..., `a_n`.

(Question — What if `n = 0`?)

```python
def max_seq([a_1, ..., a_n]):
    m := a_1
    for i := 2 to n  // 1 < i \leq n+1 \land m = \text{max}(a_1, ..., a_i-1)
        m := max(m, a_i)
    return m
```

Look at `a_0`, `a_n-1`? Lower values by 1

```python
i := 2
while i < n:
    i := i + 1
```

For formal pf take CS536
Example: Insertion Sort

```python
def insertion_sort(a[1..n]):
    # means [a1, a2, ..., an]
    for i := 2 to n # means a[1..i-1] is sorted ≤
        for k := i to 2 (by -1) # See note **
            if a[k-1] > a[k]
                swap a[k-1], a[k]
            else break inner loop
    Note ** a[1..k-1] is sorted and a[k..i-1] is sorted
    (but a[k-1] and a[k] might be out of order)
```

Not all problems are solvable (= have programs to solve them)

Example (The halting problem)
If we run program P on input x, will P eventually halt?

No such program exists (proof by contradiction)
- Suppose \( H(P, x) \) solves the halting problem
  \( H(P, x) \) returns true if \( P(x) \) halts, false if \( P(x) \) runs forever
- Define the function \( G(P) \) via:

```python
def G(P):
    if (H(P, P)) # i.e., = true, i.e., halts
        return (i.e., halt !)
    else if \( H(P, P) \) false?
        return (i.e., loops)
```

(Halting Problem)

What is the result of $H(G, G)$?

- If $H(G, G)$ halts then the if $H(P, P)$ test causes an infinite loop (?! ?!)

- If $H(G, G)$ loops forever, then the $H(P, P)$ test halts (?! ?!)

Contradiction!

- Assuming $H$ exists logically implies false, so $H$ doesn’t exist.

\[ ^2 \text{Unless you’re a grammar nerd, you’ve probably never heard of the "interrobang" ? -- that’s ok} \]

- 9 -