中国科学院大学
University of Chinese Academy of Sciences

CS101

Algorithmic Thinking

What Is Algorithmic Thinking

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

Outline

- What is algorithmic thinking
  - Knuth's characterization
  - Sorting: problem and algorithms
  - Asymptotic notations
- Divide-and-conquer paradigm
- Other interesting paradigms
- P vs. NP

These slides acknowledge sources for additional data not cited in the textbook

1. What is algorithmic thinking?

- A way of thinking to solving problems smartly by
  - designing and using algorithms
  - looking at the world through an algorithmic lens
- A smart way to study computational problems and algorithms

  Problem Name (e.g., the sorting problem)
  - **Input**: specifying the given input data.
  - **Output**: specifying the desired output data.

  Algorithm Name (e.g., bubble sort)
  - **Input**: specifying the given input data.
  - **Output**: specifying the desired output data.
  - **Steps**: specifying the sequence of computational steps.
- What does smart mean?
  - A smart way to **define** algorithms. Knuth's five-point definition
  - A smart way to **measure** algorithms. o, O, Ω, Θ ; P vs. NP
  - Smart ways to **design** algorithms. Algorithmic paradigms
  - Smart variations to **adapt** for problem nuances. E.g., how to balance all parts

1.1 What are algorithms

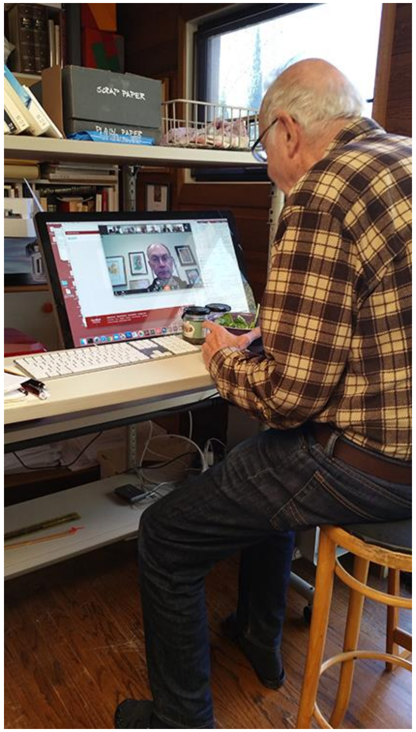
- Derived from *Algoritmi de numero Indorum*
  - Treatise by Persian mathematician Al-Khwarizmi (780-850 CE)

https://www-cs-faculty.stanford.edu/~knuth/alk3.gif

- Knuth's characterization (1968)

  An **algorithm** is a finite set of rules specifying a sequence of computational steps for solving a given problem, with the following five properties.

  - *Finiteness*. An algorithm must always terminate after a finite number of steps.
  - *Definiteness*. Each step of an algorithm must be precisely defined, that is, the actions to be carried out must be rigorously and unambiguously specified.
  - *Input*. An algorithm has zero or more inputs, given before the algorithm begins or during the algorithm's execution.
  - *Output*. An algorithm has one or more outputs, which relate to the inputs.
  - *Effectiveness*. Every operation of an algorithm must be sufficiently rudimentary, such that in principle, the operation can be done by a human using paper and pencil, in finite time.



https://www-cs-faculty.stanford.edu/~knuth/zoomlunch.jpg (2020)

Algorithm vs. non-algorithm

The common divisor (CD) problem

- **Input**: Two positive integers *x* and *y*.
- **Output**: A positive integer *z* such that *x* % *z* = 0 and *y* % *z* = 0.

Method 1 (CD-1), randomly pick and check

- **Input**: Two positive integers *x* and *y*.
- **Output**: A positive integer *z* such that *x* % *z* = 0 and *y* % *z* = 0.
- **Steps:**

```
while true
  randomly pick a positive integer z
  if (x % z == 0) and (y % z == 0) then halt
```

- CD-1 is not an algorithm, as it violates some of the 5 properties
  - May never stop, violating the finiteness property
  - “Randomly picking a positive integer" is not sufficiently rigorous or unambiguous, violating the definiteness property
    - How does one do it from the set of infinitely many positive integers?

Algorithm vs. non-algorithm

The common divisor (CD) problem

- **Input**: Two positive integers x and y.
- **Output**: A positive integer z such that $x \% z = 0$ and $y \% z = 0$.

Method 2 (CD-2): Euclid's algorithm

- **Input**: Two positive integers x and y.
- **Output**: A positive integer z such that $x \% z = 0$ and $y \% z = 0$.
- **Steps:**

```
while y ≠ 0
  x, y = y, x % y
z = x
```

Divisors of x = 36
36, 18,12, 9, 6, 4, 3, 2, 1

Divisors of y = 24
12, 8, 6, 4, 3, 2, 1

Common divisors of x = 36 and y = 24
12, 6, 4, 3, 2, 1

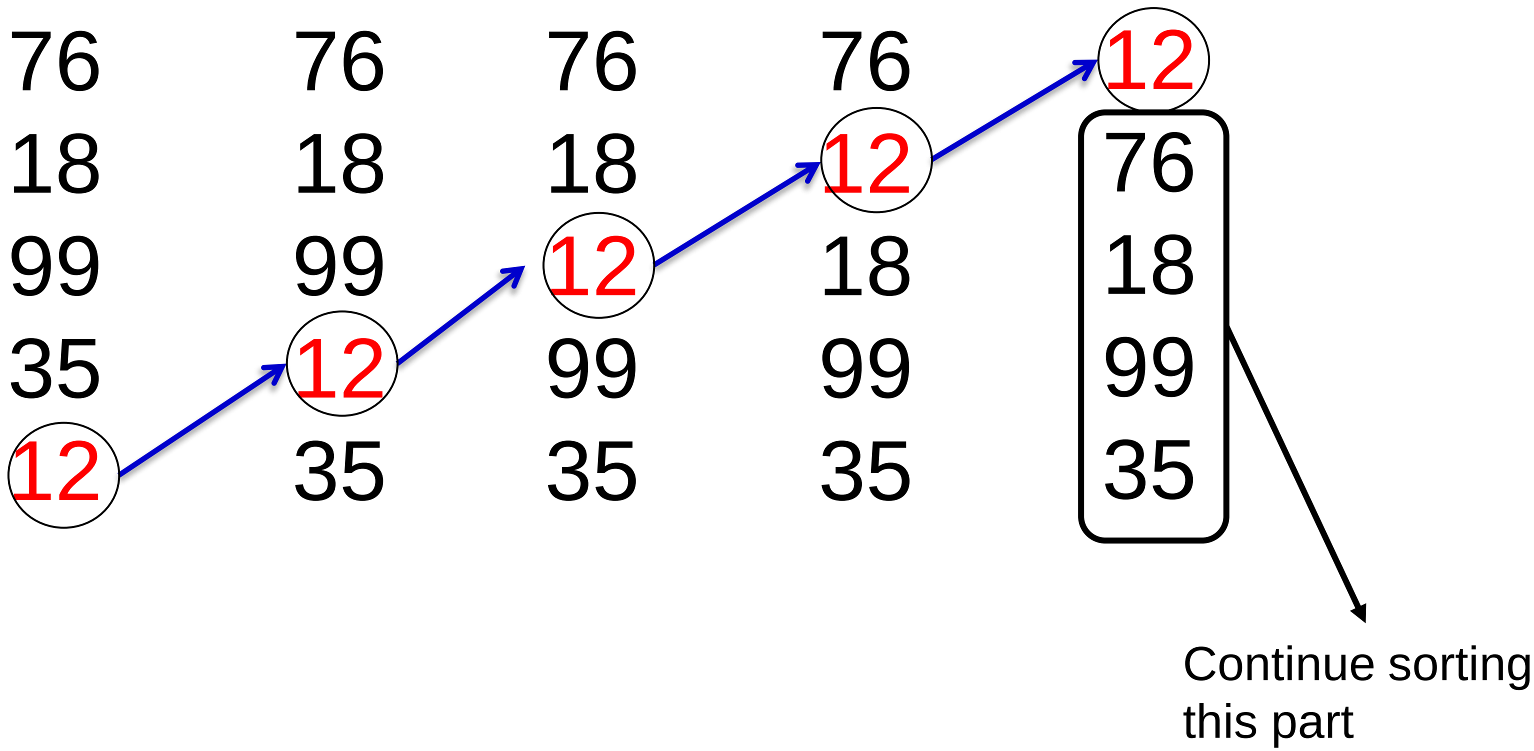
GCD of x = 36 and y = 24
12

- Exercises
  - Show that CD-2 is indeed an algorithm, because it satisfies all 5 properties in Knuth's characterization
  - Show that given two inputs x=36 and y=24, Euclid's algorithm finds the **greatest common divisor** of x and y, i.e., gcd(x, y)=12

1.2 Example: the sorting problem

- Task: given n integers, sort them in order from smallest to largest
- Think: when you play cards, how do you arrange your cards?
  - Draw the cards one by one, and insert them into the arranged cards each time
  - Sort by suit first, and then sort each suit
  - Other methods?

Bubble-sort


76
18
99
35
12
76
18
99
12
35
76
18
12
99
35
76
12
18
99
35
12
76
18
99
35
Continue sorting
this part

Bubble-sort

12	12	12	12	12
76	76	76	18	18
18	18	18	76	76
99	35	35	35	35
35	99	99	99	99

Continue sorting this part

Bubble sort meets Knuth's characterization

- *Finiteness*: the double loop always terminates
- *Definiteness*: the meaning of each step is clear
- *Input*: array A (of length *n*) to be sorted
- *Output*: the sorted array A, sharing space with Input
- *Effectiveness*: basic operations are comparison and swap
  - Both operations are sufficiently rudimentary
  - People can use pen and paper to do these operations accurately

Input: An array A of length n to be sorted, e.g., A=[6, 2, 4, 1, 5, 9].

Output: A sorted array A, e.g., A=[1, 2, 4, 5, 6, 9].

Steps:

```
for i = 1 to n-1            // for each round
    for j = 1 to n-i            // compare every adjacent pair
        if A [j] > A [j + 1] then exchange A [j] with A [j + 1];
```

Bubble-sort

- Need $n-1+n-2+\cdots+1=n\times(n-1)/2$ comparison operations
- In the worst case, need $n\times(n-1)/2$ swap operations
  - $n, n-1, \ldots, 2, 1$
- Can we improve the algorithm?

Quick-sort

$p, r = n, 1$

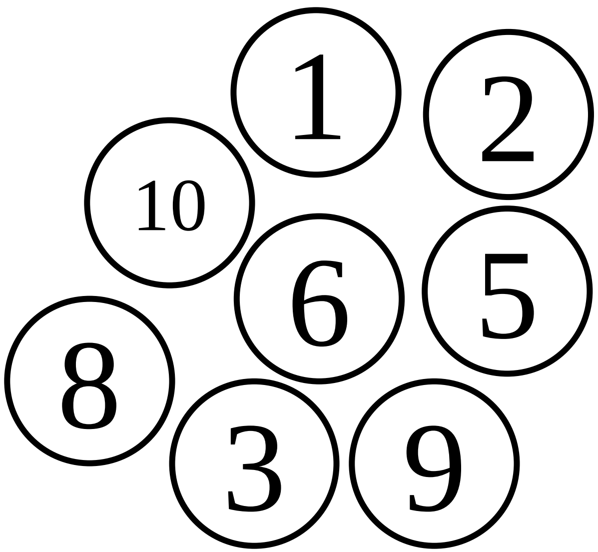
QuickSort(A, p, r)

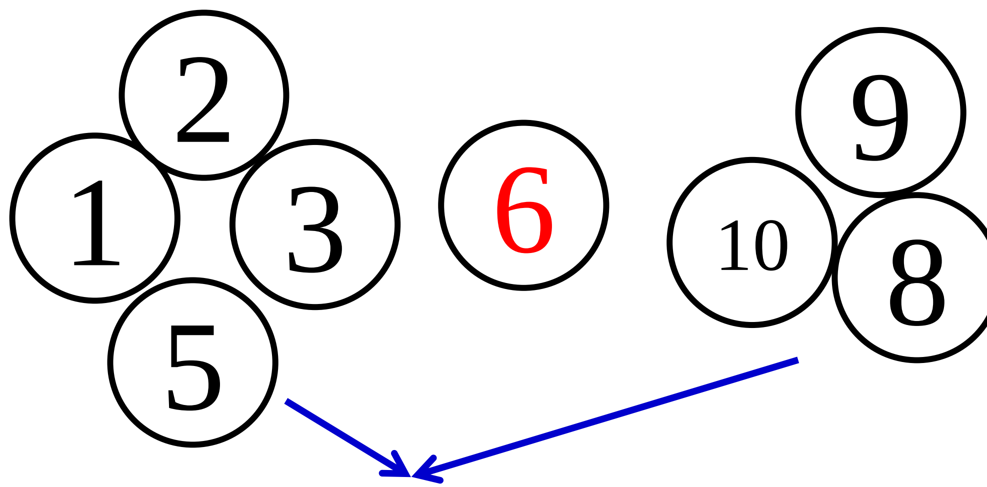
If $p < r$

1. q = Partition(A, p, r)
2. QuickSort(A, p, q-1)
3. QuickSort(A, q+1, r)

Step 1: random choose one element as the key

Step 2: compare other elements with the key, and divide all elements into two parts

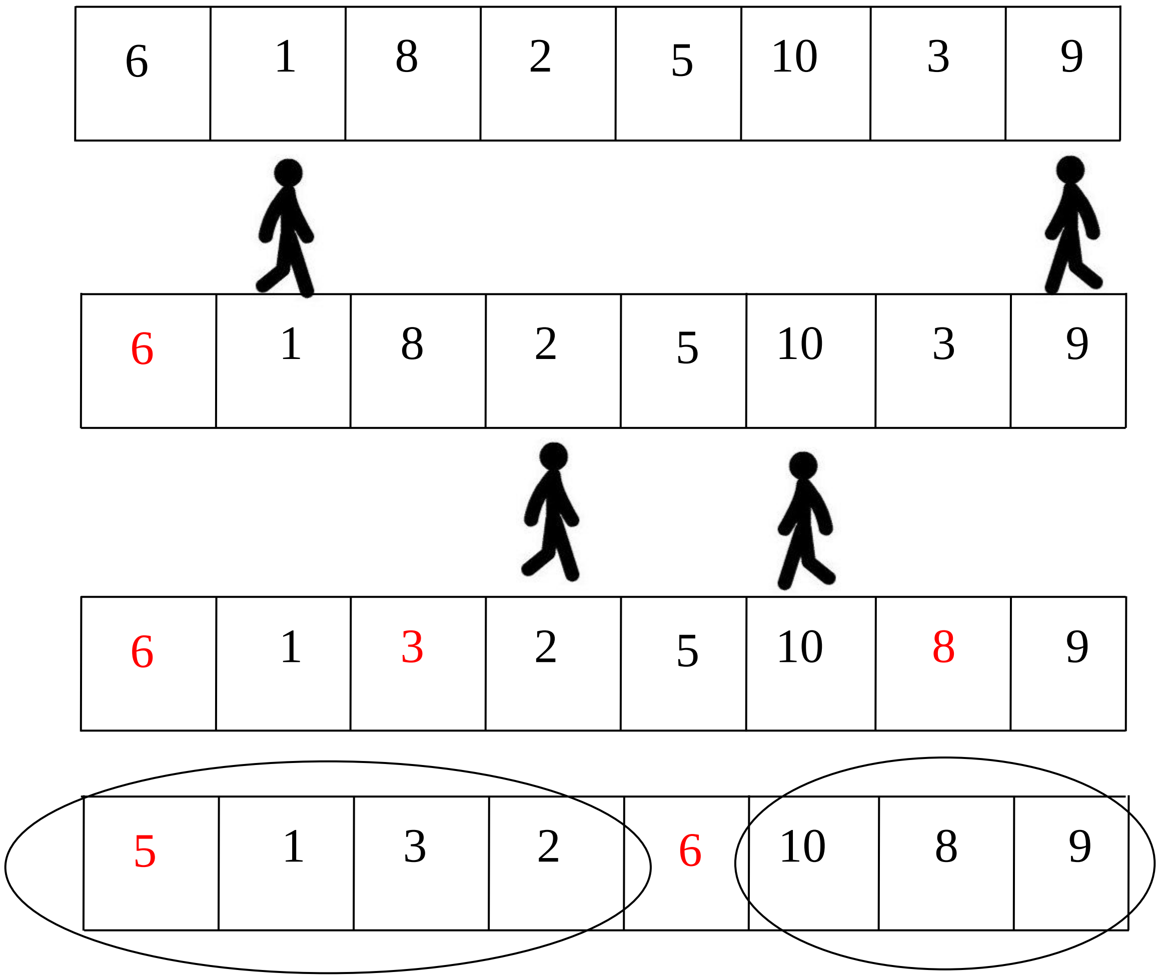




Step 3: sort each part recursively

Quick-sort

Key element: 6



Quick-sort

- How many comparison operations do we need?
  - Worst case: $n \times (n-1)/2$
    - When?
  - Average case: ?

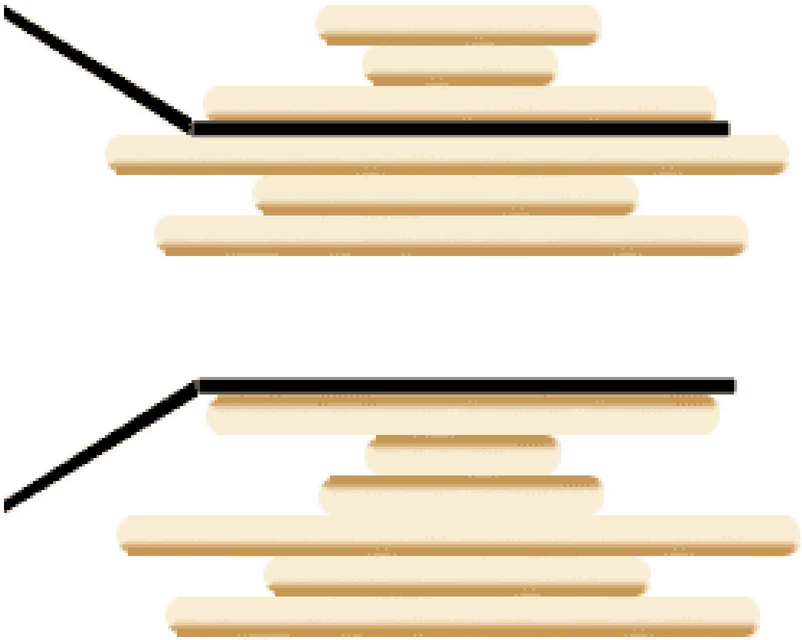
```
p, r = n, 1
QuickSort(A, p, r)
    If p < r

        1.    q = Partition(A, p, r)
        2.    QuickSort(A, p, q-1)
        3.    QuickSort(A, q+1, r)
```

Human sorter

- Design a team computer for quicksort
- Each student needs to design their own human sorter
- After discussion and evaluation, each team chooses one design, and runs it in reality

Thinking problem

- Pancake Sorting problem
  - Sort a disordered stack of pancakes in order of size when a spatula can be inserted at any point in the stack and used to flip all pancakes above it.
  - Goal: minimize the number of flip operations



Pancake Sorting problem

- 2, 1, 4, 5, 3
- 5, 4, 1, 2, 3
- 3, 2, 1, 4, 5
- 1, 2, 3, 4, 5