Polynomial-time algorithms

Running time is polynomial in the representation of an input instance

- In binary (equivalently, the number of keystrokes on your keyboard)
- Not unary!!
- Revisit the RAM model (and operations taking constant time)

Strictly speaking, we should precisely define the instructions of the RAM model and their costs. To do so, however, would be tedious and would yield little insight into algorithm design and analysis. Yet we must be careful not to abuse the RAM model. For example, what if a RAM had an instruction that sorts? Then we could sort in just one instruction. Such a RAM would be unrealistic, since real computers do not have such instructions. Our guide, therefore, is how real computers are designed. The RAM model contains instructions commonly found in real computers: arithmetic (such as add, subtract, multiply, divide, remainder, floor, ceiling), data movement (load, store, copy), and control (conditional and unconditional branch, subroutine call and return). Each such instruction takes a constant amount of time.

The data types in the RAM model are integer and floating point (for storing real numbers). Although we typically do not concern ourselves with precision in this book, in some applications precision is crucial. We also assume a limit on the size of each word of data. For example, when working with inputs of size n, we typically assume that integers are represented by $c \lg n$ bits for some constant $c \geq 1$. We require $c \geq 1$ so that each word can hold the value of n, enabling us to index the individual input elements, and we restrict c to be a constant so that the word size does not grow arbitrarily. (If the word size could grow arbitrarily, we could store huge amounts of data in one word and operate on it all in constant time—clearly an unrealistic scenario.)

Polynomial-time algorithms

Running time is polynomial in the representation of an input instance

Polynomial-time or not?

```
int fib (int n){
   if (n <= 2) return 1;
  return fib (n-1) + fib (n-2);
}
```

```
int fib (int n){//Dynamic Program
   int f[n+1];
   f[1] = f[2] = 1;
   for (int i=3; i <= n; i++)
      f[i]  = f[i-1] + f[i-2];
  return f[n];
}
```

Polynomial-time

- Constant-time if exponentiation is available
- Linear-time (not logarithmic) if not

$$F_n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n$$

http://en.wikipedia.org/wiki/Fibonacci_number

What is the running time of this algorithm?

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          $c_{ij} = 0$
6          for k = 1 to n
7              $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
8  return C
```

$\Theta(n^3)$

But not cubic in the size of the input

Showing problems to not be solvable in polynomial time

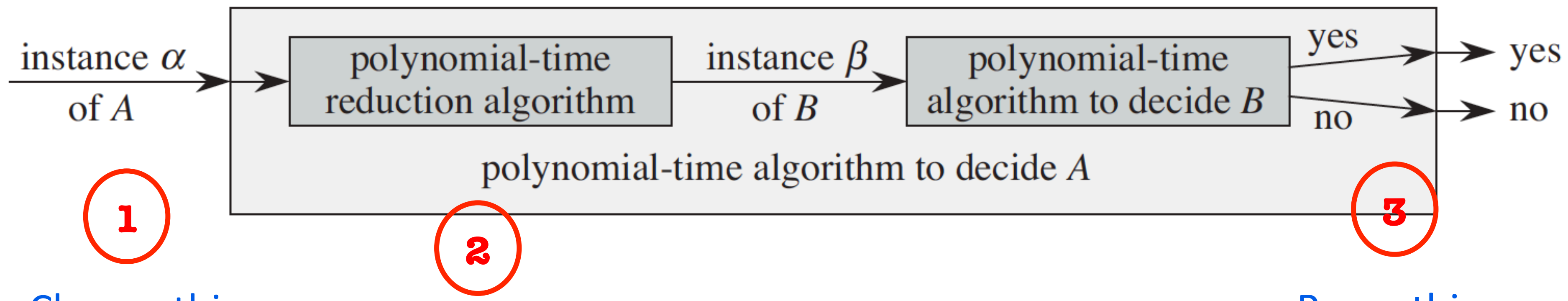
Optimization problem → decision problem

- Has a yes/ no answer
- Easier (no harder) than the optimization problem
- Polynomial-time solution often means a polynomial-time solution to the optimization problem, by "binary search"

Showing problems to not be solvable in polynomial time

Optimization problem → decision problem

Reductions Find a known “hard” problem A, such that a polynomial-time solution to your problem B implies a polynomial-time solution to A


instance α of A
polynomial-time reduction algorithm
instance β of B
polynomial-time algorithm to decide B
yes
no
yes
no
polynomial-time algorithm to decide A
1
2
3

Choose this

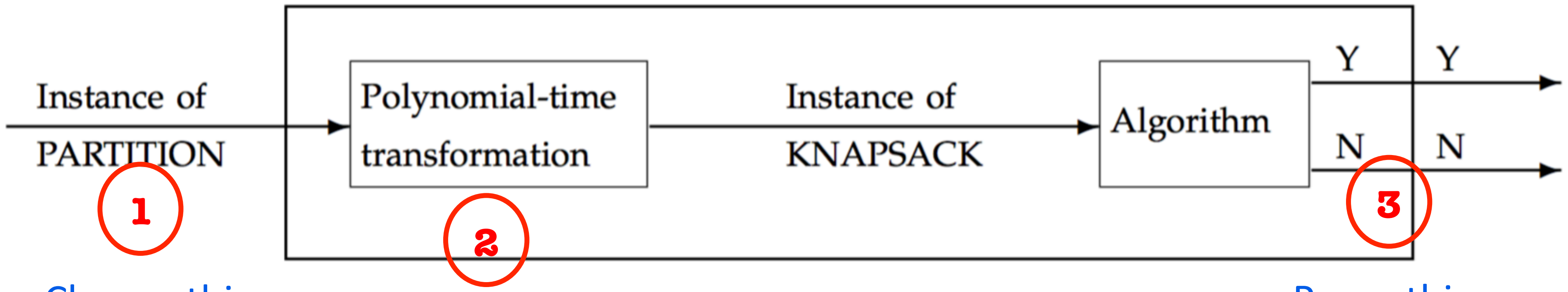
Define this

Prove this

Showing problems to not be solvable in polynomial time

Optimization problem → decision problem

Reductions Find a known "hard" problem A, such that a polynomial-time solution to your problem B implies a polynomial-time solution to A

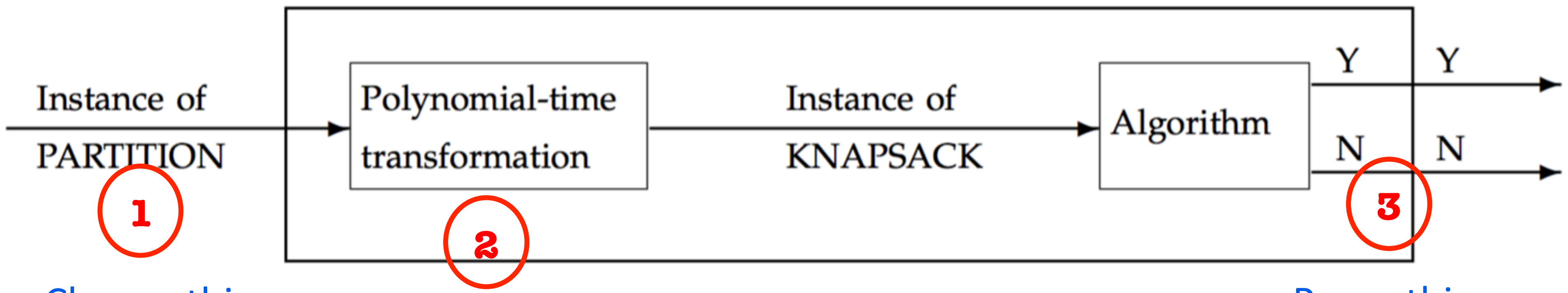


Choose this

Define this

Prove this

Showing problems to not be solvable in polynomial time



Choose this

Define this

Prove this

INSTANCE. Finite set A, and a size $s(a) \in \mathbb{Z}^+$ for each $a \in A$

QUESTION. Is there a subset $A' \subseteq A$ such that

$$\sum_{a \in A'} s(a) = \sum_{a \in A \setminus A'} s(a)$$

An item for each $a \in A$ of PARTITION, with weight $s(a)$ and value $s(a)$. $W = V =$ half the sum of the $s(a)$'s

Proof of correctness:

- Input instance in PARTITION
- Input instance not in PARTITION

Showing problems to not be solvable in polynomial time

Optimization problem → decision problem

Reductions Find a known “hard” problem A, such that a polynomial-time solution to your problem B implies a polynomial-time solution to A

We find the known hard problems in [Garey & Johnson]

- How did they get there?
- The first hard problem

The class of problems **NP**

P = decision problems solved by some polynomial-time algorithm

Observation: For some problems, it seems easier to verify a given solution, than to find one

- Factorization (is n a composite number?)
- Partition
  - Given A’, verify that the sizes of elements in A’ sum to half the total sum
  - Knapsack
  - Given a subset of the items, verify that their weights sum to $\leq$ W, and values to $\geq$V

NP = decision problems verified by some polynomial-time algorithm

Certificates of membership in the language defined by the problem

A language belongs to the class NP if there is an algorithm that accepts an input and a certificate, and verifies that the input belongs to the language, in time polynomial in the size of the input

The class of problems **NP**

- Every decision problem defines a language of all strings for which the decision problem yields a "yes" answer
- For decision problems in the class P, there is a polynomial-time algorithm that determines whether an input belongs to the language
- For decision problems in the class NP, there is a polynomial-time algorithm that verifies whether a given certificate for a given input shows that the input belongs to the language
    - The running time is polynomial in the size of the input (not the certificate)
    - [Hence, certificates should be of polynomial size]
- Every language in the class P is in the class NP (why?)
- Languages such as PARTITION are in NP, but not known to be in P