Complexity Basics

K. Subramani[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University

February 2, 2015

Outline

Outline

Outline

Outline

Outline

Review

Review

Main concepts

Review

Main concepts

1. Alphabet, strings and languages.

Review

Main concepts

1. Alphabet, strings and languages.
2. Problems.

Review

Main concepts

1. Alphabet, strings and languages.
2. Problems.
3. Connection between problems and languages.

Review

Main concepts

1. Alphabet, strings and languages.
2. Problems.
3. Connection between problems and languages.
4. Asymptotics and caclulus.

Review

Main concepts

1. Alphabet, strings and languages.
2. Problems.
3. Connection between problems and languages.
4. Asymptotics and caclulus.
5. Probability and random variables.

Review

Main concepts

1. Alphabet, strings and languages.
2. Problems.
3. Connection between problems and languages.
4. Asymptotics and caclulus.
5. Probability and random variables.
6. Abstract Algebra.

Review

Main concepts

1. Alphabet, strings and languages.
2. Problems.
3. Connection between problems and languages.
4. Asymptotics and caclulus.
5. Probability and random variables.
6. Abstract Algebra.
7. Upper and lower bounds.

Review

Main concepts

1. Alphabet, strings and languages.
2. Problems.
3. Connection between problems and languages.
4. Asymptotics and caclulus.
5. Probability and random variables.
6. Abstract Algebra.
7. Upper and lower bounds.
8. Problem paradigms.

Problem

Problem

Main points

Problem

Main points

1. Instance of a problem e.g., Eulerian tour.

Problem

Main points

1. Instance of a problem e.g., Eulerian tour.
2. Finiteness of instance.

Problem

Main points

1. Instance of a problem e.g., Eulerian tour.
2. Finiteness of instance.
3. The Eulerian path problem.

Problem

Main points

1. Instance of a problem e.g., Eulerian tour.
2. Finiteness of instance.
3. The Eulerian path problem.
4. The Hamilton circuit problem.

Problem

Main points

1. Instance of a problem e.g., Eulerian tour.
2. Finiteness of instance.
3. The Eulerian path problem.
4. The Hamilton circuit problem.
5. The Chess problem.

Problem

Main points

1. Instance of a problem e.g., Eulerian tour.
2. Finiteness of instance.
3. The Eulerian path problem.
4. The Hamilton circuit problem.
5. The Chess problem.
6. Infinite set of instances.

Problem

Main points

1. Instance of a problem e.g., Eulerian tour.
2. Finiteness of instance.
3. The Eulerian path problem.
4. The Hamilton circuit problem.
5. The Chess problem.
6. Infinite set of instances.
7. Decision and search problems.

Solutions

Solutions

Main points

Solutions

Main points

1. What is a solution?

Solutions

Main points

1. What is a solution?
2. What is an algorithm?

Solutions

Main points

1. What is a solution?
2. What is an algorithm?
3. Algorithms as functions.

The GCD problem

The GCD problem

Definition

The GCD problem

Definition

Given two positive numbers (integers) a and b, find their greatest common divisor.

The GCD problem

Definition

Given two positive numbers (integers) a and b, find their greatest common divisor.

Note

The GCD problem

Definition

Given two positive numbers (integers) a and b, find their greatest common divisor.

Note

The GCD problem

Definition

Given two positive numbers (integers) a and b, find their greatest common divisor.

Note

1. *This problem is referred to as* $gcd(a, b)$.

The GCD problem

Definition

Given two positive numbers (integers) a and b, find their greatest common divisor.

Note

1. *This problem is referred to as* $gcd(a, b)$*.*
2. *It is also well-defined when a and b are not positive,*

The GCD problem

Definition

Given two positive numbers (integers) a and b, find their greatest common divisor.

Note

1. *This problem is referred to as* $gcd(a, b)$*.*
2. *It is also well-defined when a and b are not positive, as long as both are not zero.*

The GCD problem

Definition

Given two positive numbers (integers) a and b, find their greatest common divisor.

Note

1. *This problem is referred to as* $gcd(a, b)$.
2. *It is also well-defined when a and b are not positive, as long as both are not zero.*
3. *Euclid observed that* $gcd(a, b) =$

The GCD problem

Definition

Given two positive numbers (integers) a and b, find their greatest common divisor.

Note

1. *This problem is referred to as $gcd(a, b)$.*
2. *It is also well-defined when a and b are not positive, as long as both are not zero.*
3. *Euclid observed that $gcd(a, b) = gcd(b, a \bmod b)$.*

The Euclidean Algorithm

The Euclidean Algorithm

Algorithm

FUNCTION GCD(a, b)
if ($b = 0$) **then**
    return (a)
else
    return (GCD($b, a \bmod b$)).
end if

Types of Analysis

Types of Analysis

How to analyze

Types of Analysis

How to analyze

1. Worst-case.

Types of Analysis

How to analyze

1. Worst-case.
2. Adversary.

Types of Analysis

How to analyze

1. Worst-case.
2. Adversary.
3. Average-case.

Types of Analysis

How to analyze

1. Worst-case.
2. Adversary.
3. Average-case.
4. Other ways.

Time, Space and Scaling

Time, Space and Scaling

Main Parameters

Time, Space and Scaling

Main Parameters

1. Notion of time.

Time, Space and Scaling

Main Parameters

1. Notion of time.
2. Notion of space.

Time, Space and Scaling

Main Parameters

1. Notion of time.
2. Notion of space.
3. Notion of input size.

Time, Space and Scaling

Main Parameters

1. Notion of time.
2. Notion of space.
3. Notion of input size.
4. Scaling with respect to input size.

Time, Space and Scaling

Main Parameters

1. Notion of time.
2. Notion of space.
3. Notion of input size.
4. Scaling with respect to input size.
5. Analysis of the Euclidean algorithm.

Intrinsic Complexity

Intrinsic Complexity

Definition

Intrinsic Complexity

Definition

The intrinsic complexity of a problem is the complexity of the most efficient algorithm that solves it.

The integer multiplication problem

The integer multiplication problem

Two integer multiplication

The integer multiplication problem

Two integer multiplication

Compute the product of two numbers a and b having n digits each.

The Master Method for solving certain recurrences

The Master Method for solving certain recurrences

The Master theorem

The Master Method for solving certain recurrences

The Master theorem

Consider the recurrence

The Master Method for solving certain recurrences

The Master theorem

Consider the recurrence

$$T(n) = a \cdot T(\frac{n}{b}) + f(n).$$

The Master Method for solving certain recurrences

The Master theorem

Consider the recurrence

$$T(n) = a \cdot T(\frac{n}{b}) + f(n).$$

If $f(n) \in O(n^c)$, where $c < \log_b a$,

The Master Method for solving certain recurrences

The Master theorem

Consider the recurrence

$$T(n) = a \cdot T(\frac{n}{b}) + f(n).$$

If $f(n) \in O(n^c)$, where $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

The Grade school approach

The Grade school approach

Grade school algorithm

The Grade school approach

Grade school algorithm

1. Multiply *a* with each digit of *b* performing appropriate shifting.

The Grade school approach

Grade school algorithm

1. Multiply *a* with each digit of *b* performing appropriate shifting.
2. Add up the individual products.

The Grade school approach

Grade school algorithm

1. Multiply *a* with each digit of *b* performing appropriate shifting.
2. Add up the individual products.

Note

The Grade school approach

Grade school algorithm

1. Multiply a with each digit of b performing appropriate shifting.
2. Add up the individual products.

Note

The above algorithm requires

The Grade school approach

Grade school algorithm

1. Multiply *a* with each digit of *b* performing appropriate shifting.
2. Add up the individual products.

Note

The above algorithm requires $\Theta(n^2)$ *multiplications and*

The Grade school approach

Grade school algorithm

1. Multiply a with each digit of b performing appropriate shifting.
2. Add up the individual products.

Note

The above algorithm requires $\Theta(n^2)$ multiplications and $\Theta(n^2)$ additions.

An improved algorithm

An improved algorithm

A slightly more sophisticated approach

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$,

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.
3. Hence, $a \cdot b =$

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.
3. Hence, $a \cdot b = 10^{n} \cdot x \cdot u$

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.
3. Hence, $a \cdot b = 10^{n} \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u)$

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.
3. Hence, $a \cdot b = 10^n \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u) + y \cdot v$.

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.
3. Hence, $a \cdot b = 10^n \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u) + y \cdot v$.
4. Recall that in the decimal algebra, multiplying by powers of 10 is tantamount to left shifting.

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.
3. Hence, $a \cdot b = 10^n \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u) + y \cdot v$.
4. Recall that in the decimal algebra, multiplying by powers of 10 is tantamount to left shifting.
5. Thus, if $T(n)$ is the time taken to multiply two integers, then

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.
3. Hence, $a \cdot b = 10^n \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u) + y \cdot v$.
4. Recall that in the decimal algebra, multiplying by powers of 10 is tantamount to left shifting.
5. Thus, if $T(n)$ is the time taken to multiply two integers, then

$$T(n) = 4 \cdot T(\frac{n}{2}) + O(n)$$

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.
3. Hence, $a \cdot b = 10^n \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u) + y \cdot v$.
4. Recall that in the decimal algebra, multiplying by powers of 10 is tantamount to left shifting.
5. Thus, if $T(n)$ is the time taken to multiply two integers, then

$$T(n) = 4 \cdot T(\frac{n}{2}) + O(n)$$

6. Solving the above recurrence gives $T(n) =$

An improved algorithm

A slightly more sophisticated approach

1. Let $a = \langle x, y \rangle$ and $b = \langle u, v \rangle$, where x and u are the first $\frac{n}{2}$ digits of the numbers a and b respectively.
2. It follows that $a = 10^{\frac{n}{2}} \cdot x + y$ and $b = 10^{\frac{n}{2}} \cdot u + v$.
3. Hence, $a \cdot b = 10^n \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u) + y \cdot v$.
4. Recall that in the decimal algebra, multiplying by powers of 10 is tantamount to left shifting.
5. Thus, if $T(n)$ is the time taken to multiply two integers, then

$$T(n) = 4 \cdot T(\frac{n}{2}) + O(n)$$

6. Solving the above recurrence gives $T(n) = \Theta(n^2)$.

An asymptotically superior algorithm

An asymptotically superior algorithm

A neat observation

$$(x+y)\cdot(u+v)-x\cdot u-y\cdot v=$$

An asymptotically superior algorithm

A neat observation

$$(x + y) \cdot (u + v) - x \cdot u - y \cdot v = x \cdot v + y \cdot u$$

An asymptotically superior algorithm

A neat observation

$$(x + y) \cdot (u + v) - x \cdot u - y \cdot v = x \cdot v + y \cdot u$$

New algorithm

An asymptotically superior algorithm

A neat observation

$$(x + y) \cdot (u + v) - x \cdot u - y \cdot v = x \cdot v + y \cdot u$$

New algorithm

1. Compute $x \cdot u$, $y \cdot v$ and $(x + y) \cdot (u + v)$.

An asymptotically superior algorithm

A neat observation

$$(x + y) \cdot (u + v) - x \cdot u - y \cdot v = x \cdot v + y \cdot u$$

New algorithm

1. Compute $x \cdot u$, $y \cdot v$ and $(x + y) \cdot (u + v)$.
2. Now compute $(x \cdot v + y \cdot u)$,

An asymptotically superior algorithm

A neat observation

$$(x + y) \cdot (u + v) - x \cdot u - y \cdot v = x \cdot v + y \cdot u$$

New algorithm

1. Compute $x \cdot u$, $y \cdot v$ and $(x + y) \cdot (u + v)$.
2. Now compute $(x \cdot v + y \cdot u)$, using the above observation.

An asymptotically superior algorithm

A neat observation

$$(x + y) \cdot (u + v) - x \cdot u - y \cdot v = x \cdot v + y \cdot u$$

New algorithm

1. Compute $x \cdot u$, $y \cdot v$ and $(x + y) \cdot (u + v)$.
2. Now compute $(x \cdot v + y \cdot u)$, using the above observation.
3. Finally, compute $a \cdot b =$

An asymptotically superior algorithm

A neat observation

$$(x+y)\cdot(u+v) - x\cdot u - y\cdot v = x\cdot v + y\cdot u$$

New algorithm

1. Compute $x \cdot u$, $y \cdot v$ and $(x+y)\cdot(u+v)$.
2. Now compute $(x\cdot v + y\cdot u)$, using the above observation.
3. Finally, compute $a \cdot b = 10^n \cdot x \cdot u$

An asymptotically superior algorithm

A neat observation

$$(x + y) \cdot (u + v) - x \cdot u - y \cdot v = x \cdot v + y \cdot u$$

New algorithm

1. Compute $x \cdot u$, $y \cdot v$ and $(x + y) \cdot (u + v)$.
2. Now compute $(x \cdot v + y \cdot u)$, using the above observation.
3. Finally, compute $a \cdot b = 10^n \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u)$

An asymptotically superior algorithm

A neat observation

$$(x+y)\cdot(u+v) - x\cdot u - y\cdot v = x\cdot v + y\cdot u$$

New algorithm

1. Compute $x\cdot u$, $y\cdot v$ and $(x+y)\cdot(u+v)$.
2. Now compute $(x\cdot v + y\cdot u)$, using the above observation.
3. Finally, compute $a\cdot b = 10^n\cdot x\cdot u + 10^{\frac{n}{2}}\cdot(x\cdot v + y\cdot u) + y\cdot v$.

An asymptotically superior algorithm

A neat observation

$$(x+y)\cdot(u+v) - x\cdot u - y\cdot v = x\cdot v + y\cdot u$$

New algorithm

1. Compute $x\cdot u$, $y\cdot v$ and $(x+y)\cdot(u+v)$.
2. Now compute $(x\cdot v + y\cdot u)$, using the above observation.
3. Finally, compute $a\cdot b = 10^n\cdot x\cdot u + 10^{\frac{n}{2}}\cdot(x\cdot v + y\cdot u) + y\cdot v$.

Analysis

$$T(n) =$$

An asymptotically superior algorithm

A neat observation

$$(x+y)\cdot(u+v)-x\cdot u-y\cdot v=x\cdot v+y\cdot u$$

New algorithm

1. Compute $x \cdot u$, $y \cdot v$ and $(x+y)\cdot(u+v)$.
2. Now compute $(x \cdot v + y \cdot u)$, using the above observation.
3. Finally, compute $a \cdot b = 10^n \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u) + y \cdot v$.

Analysis

$$T(n) = 3 \cdot T(\frac{n}{2}) + O(n)$$

It follows that $T(n) =$

An asymptotically superior algorithm

A neat observation

$$(x + y) \cdot (u + v) - x \cdot u - y \cdot v = x \cdot v + y \cdot u$$

New algorithm

1. Compute $x \cdot u$, $y \cdot v$ and $(x + y) \cdot (u + v)$.
2. Now compute $(x \cdot v + y \cdot u)$, using the above observation.
3. Finally, compute $a \cdot b = 10^n \cdot x \cdot u + 10^{\frac{n}{2}} \cdot (x \cdot v + y \cdot u) + y \cdot v$.

Analysis

$$T(n) = 3 \cdot T(\frac{n}{2}) + O(n)$$

It follows that $T(n) = O(n^{\log_2 3})$.

Polynomial versus Exponential

Polynomial versus Exponential

When is computation infeasible?

Polynomial versus Exponential

When is computation infeasible?

1. Difference between $\Theta(n)$ and $\Theta(2^n)$ when n doubles.

Polynomial versus Exponential

When is computation infeasible?

1. Difference between $\Theta(n)$ and $\Theta(2^n)$ when n doubles.
2. $n!$ when $n = 20$ is larger than the date of the universe.

Polynomial versus Exponential

When is computation infeasible?

1. Difference between $\Theta(n)$ and $\Theta(2^n)$ when n doubles.
2. $n!$ when $n = 20$ is larger than the date of the universe.
3. The Vishnu Sharma story.

Analyzing Euler's Algorithm

Analyzing Euler's Algorithm

Details

Analyzing Euler's Algorithm

Details

1. How fast is Euler's algorithm?

Analyzing Euler's Algorithm

Details

1. How fast is Euler's algorithm? Details!

Analyzing Euler's Algorithm

Details

1. How fast is Euler's algorithm? Details!
2. How many times does the main **for** loop run?

Analyzing Euler's Algorithm

Details

1. How fast is Euler's algorithm? Details!
2. How many times does the main **for** loop run?
3. Measuring running time.

Analyzing Euler's Algorithm

Details

1. How fast is Euler's algorithm? Details!
2. How many times does the main **for** loop run?
3. Measuring running time. What is an elementary step?

Analyzing Euler's Algorithm

Details

1. How fast is Euler's algorithm? Details!
2. How many times does the main **for** loop run?
3. Measuring running time. What is an elementary step?
4. How much time to measure the degree of a vertex?

Analyzing Euler's Algorithm

Details

1. How fast is Euler's algorithm? Details!
2. How many times does the main **for** loop run?
3. Measuring running time. What is an elementary step?
4. How much time to measure the degree of a vertex? Representation details.

Analyzing Euler's Algorithm

Details

1. How fast is Euler's algorithm? Details!
2. How many times does the main **for** loop run?
3. Measuring running time. What is an elementary step?
4. How much time to measure the degree of a vertex? Representation details.
5. Tape storage versus RAM storage.

Analyzing Euler's Algorithm

Details

1. How fast is Euler's algorithm? Details!
2. How many times does the main **for** loop run?
3. Measuring running time. What is an elementary step?
4. How much time to measure the degree of a vertex? Representation details.
5. Tape storage versus RAM storage.
6. Size of instance.

The class P

The class P

Definition

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

Note

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

Note

1. **P** *is a complexity class.*

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

Note

1. **P** *is a complexity class.*
2. *A problem is not in* **P***, if . . ...*

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

Note

1. **P** *is a complexity class.*
2. *A problem is not in* **P***, if . . . ..*
3. *We include both decision and search problems.*

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

Note

1. **P** *is a complexity class.*
2. *A problem is not in* **P**, *if . . ...*
3. *We include both decision and search problems.*
4. *In general,* $TIME(\mathbf{f}(\mathbf{n}))$ *is the class of problems for which an algorithm exists that solves problems of size n, in time* $TIME(\mathbf{f}(\mathbf{n}))$.

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

Note

1. **P** *is a complexity class.*
2. *A problem is not in* **P**, *if . . ...*
3. *We include both decision and search problems.*
4. *In general,* $TIME(\mathbf{f}(\mathbf{n}))$ *is the class of problems for which an algorithm exists that solves problems of size n, in time* $TIME(\mathbf{f}(\mathbf{n}))$.
5. $\mathbf{P} = \cup_{c>o} TIME(n^c)$.

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

Note

1. **P** *is a complexity class.*
2. *A problem is not in* **P***, if . . ...*
3. *We include both decision and search problems.*
4. *In general,* $TIME(\mathbf{f}(\mathbf{n}))$ *is the class of problems for which an algorithm exists that solves problems of size n, in time* $TIME(\mathbf{f}(\mathbf{n}))$*.*
5. $\mathbf{P} = \cup_{c>o} TIME(n^c)$.
6. *Is* $TIME(n^{\log n}) \subseteq \mathbf{P}$*?*

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

Note

1. **P** *is a complexity class.*
2. *A problem is not in* **P***, if . . ...*
3. *We include both decision and search problems.*
4. *In general,* $TIME(\mathbf{f}(\mathbf{n}))$ *is the class of problems for which an algorithm exists that solves problems of size n, in time* $TIME(\mathbf{f}(\mathbf{n}))$*.*
5. $\mathbf{P} = \cup_{c>o} TIME(n^c)$*.*
6. *Is* $TIME(n^{\log n}) \subseteq \mathbf{P}$*?*
7. *Defining* $\mathbf{TIME}(f(n))$*.*

The class P

Definition

P is the class of problems for which an algorithm exists that solves problems of size n, in time $O(n^c)$ for some constant c.

Note

1. **P** *is a complexity class.*
2. *A problem is not in* **P***, if . . ...*
3. *We include both decision and search problems.*
4. *In general,* $TIME(\mathbf{f}(\mathbf{n}))$ *is the class of problems for which an algorithm exists that solves problems of size n, in time* $TIME(\mathbf{f}(\mathbf{n}))$*.*
5. $\mathbf{P} = \cup_{c>o} TIME(n^c)$*.*
6. *Is* $TIME(n^{\log n}) \subseteq \mathbf{P}$*?*
7. *Defining* $\mathbf{TIME}(f(n))$*.*
8. *The class* **EXP** *is defined as* $TIME(2^{n^c})$*.*

Robustness of P

Robustness of P

Why P is such a robust class

Robustness of P

Why P is such a robust class

1. Elementary steps can change within reason.

Robustness of P

Why P is such a robust class

1. Elementary steps can change within reason.
2. Architecture can change within reason.

Robustness of P

Why P is such a robust class

1. Elementary steps can change within reason.
2. Architecture can change within reason.
3. Input formats can change within reason.Binary/unary encoding. Buffing up the input.

Robustness of P

Why P is such a robust class

1. Elementary steps can change within reason.
2. Architecture can change within reason.
3. Input formats can change within reason.Binary/unary encoding. Buffing up the input.
4. Implementation details can change within reason.

Robustness of P

Why P is such a robust class

1. Elementary steps can change within reason.
2. Architecture can change within reason.
3. Input formats can change within reason.Binary/unary encoding. Buffing up the input.
4. Implementation details can change within reason.

Note

Being in **P** *is a fundamental property of a problem and* not dependent upon how somebody goes about solving it.

Tractability and mathematical insight

Tractability and mathematical insight

What is tractable

Tractability and mathematical insight

What is tractable

1. Definition of tractability.

Tractability and mathematical insight

What is tractable

1. Definition of tractability.
2. Does tractability coincide with **P**?

Tractability and mathematical insight

What is tractable

1. Definition of tractability.
2. Does tractability coincide with **P**?
3. How practical is $O(n^{10})$?

Tractability and mathematical insight

What is tractable

1. Definition of tractability.
2. Does tractability coincide with **P**?
3. How practical is $O(n^{10})$?
4. Sublinear algorithms.

Tractability and mathematical insight

What is tractable

1. Definition of tractability.
2. Does tractability coincide with **P**?
3. How practical is $O(n^{10})$?
4. Sublinear algorithms.
5. Going from $\mathbf{P}^c$ to **P** gives fundamental insight into the nature of problems.