Lecture 1

Course Overview

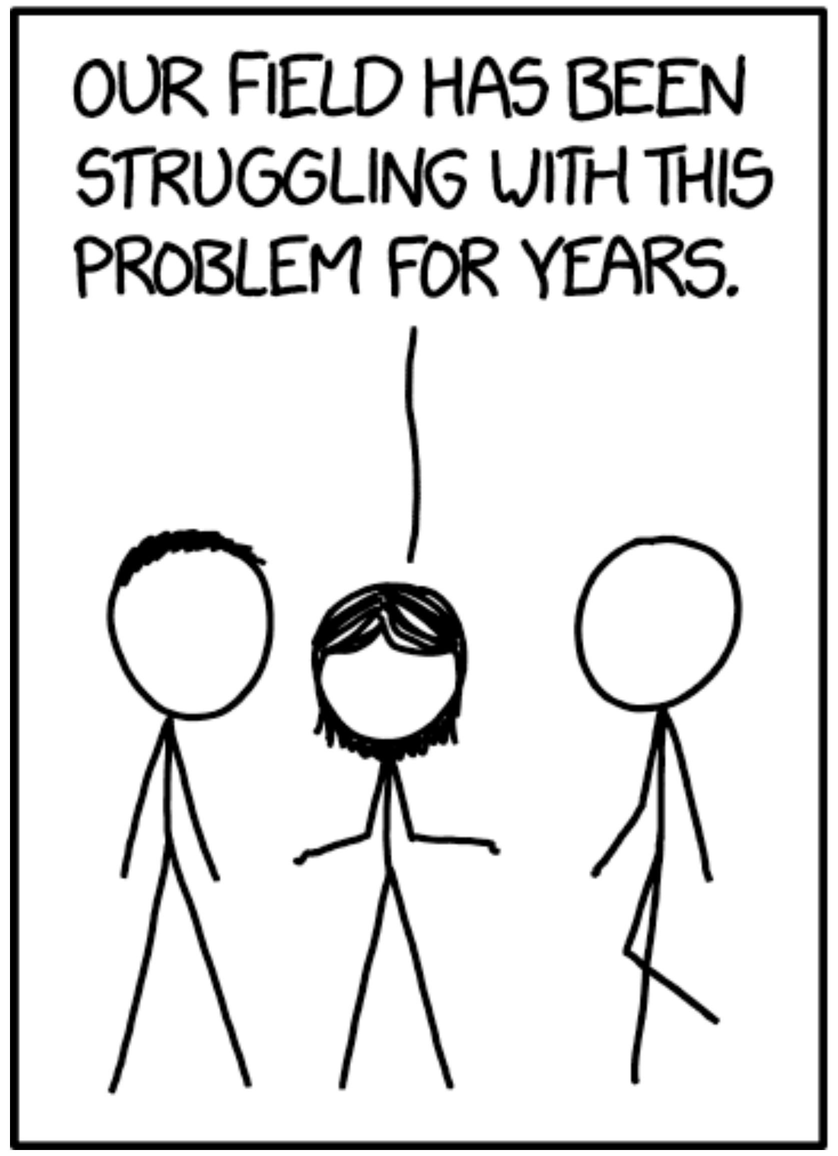
Overview of Complexity Theory

Overview of Complexity Theory

Algorithms vs Complexity Theory:

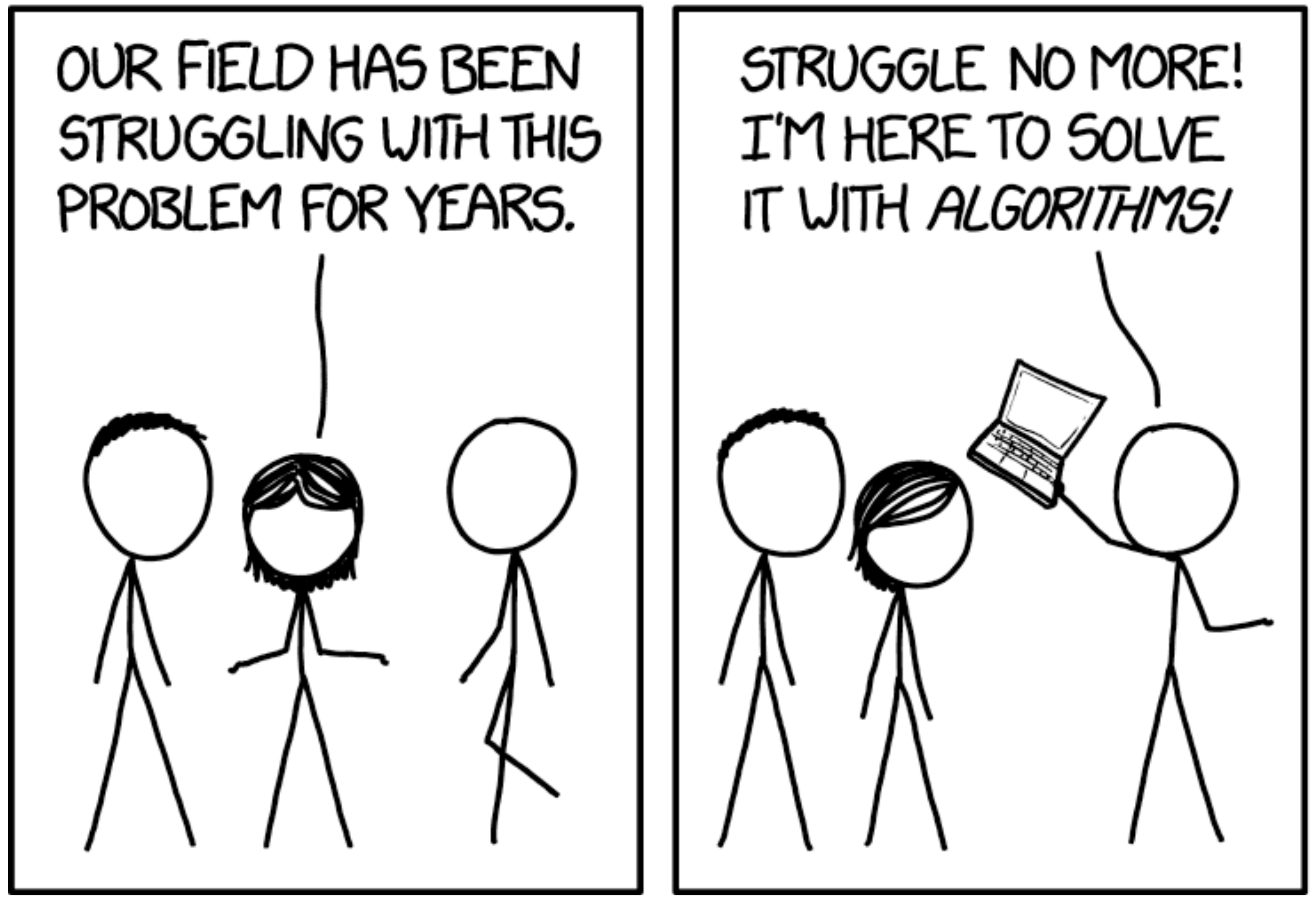
Overview of Complexity Theory

Algorithms vs Complexity Theory:



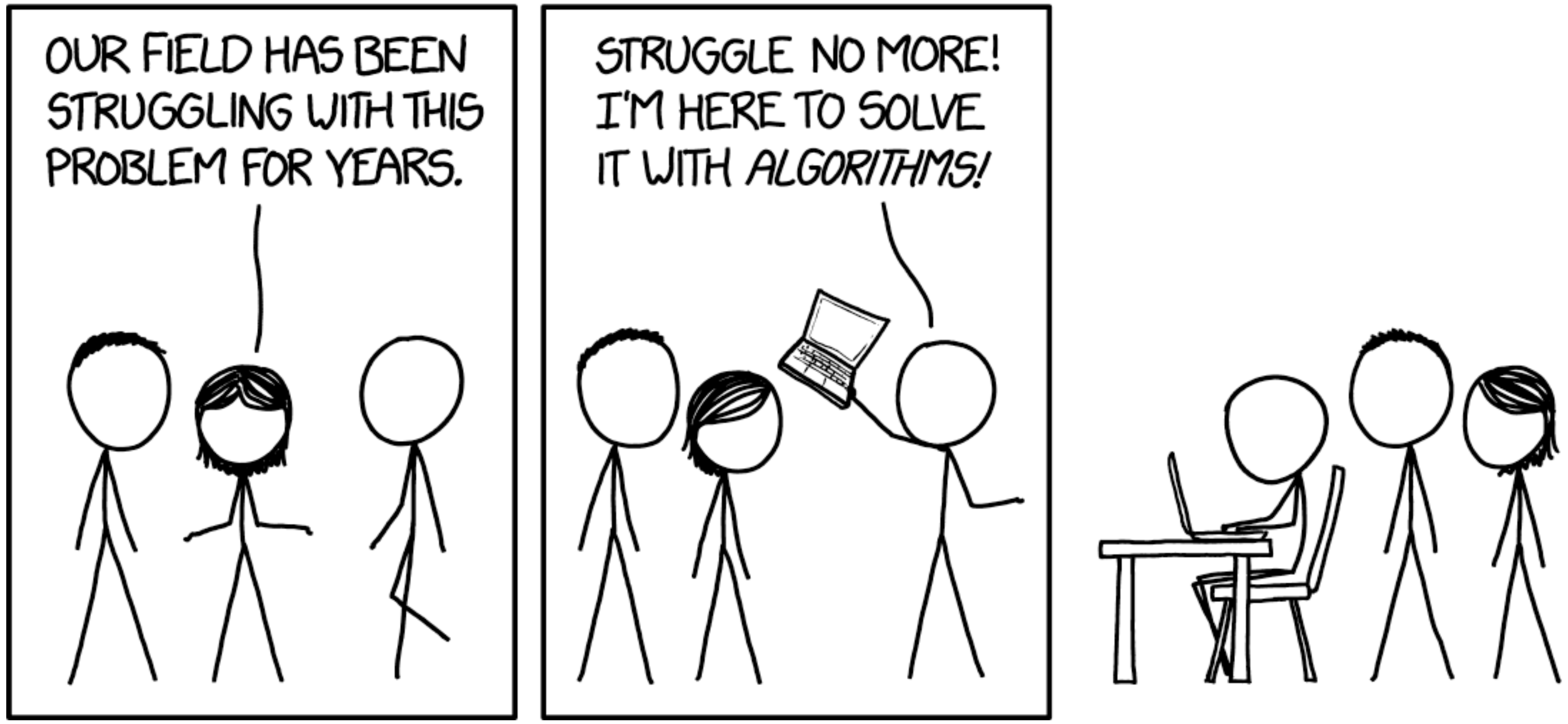
Overview of Complexity Theory

Algorithms vs Complexity Theory:



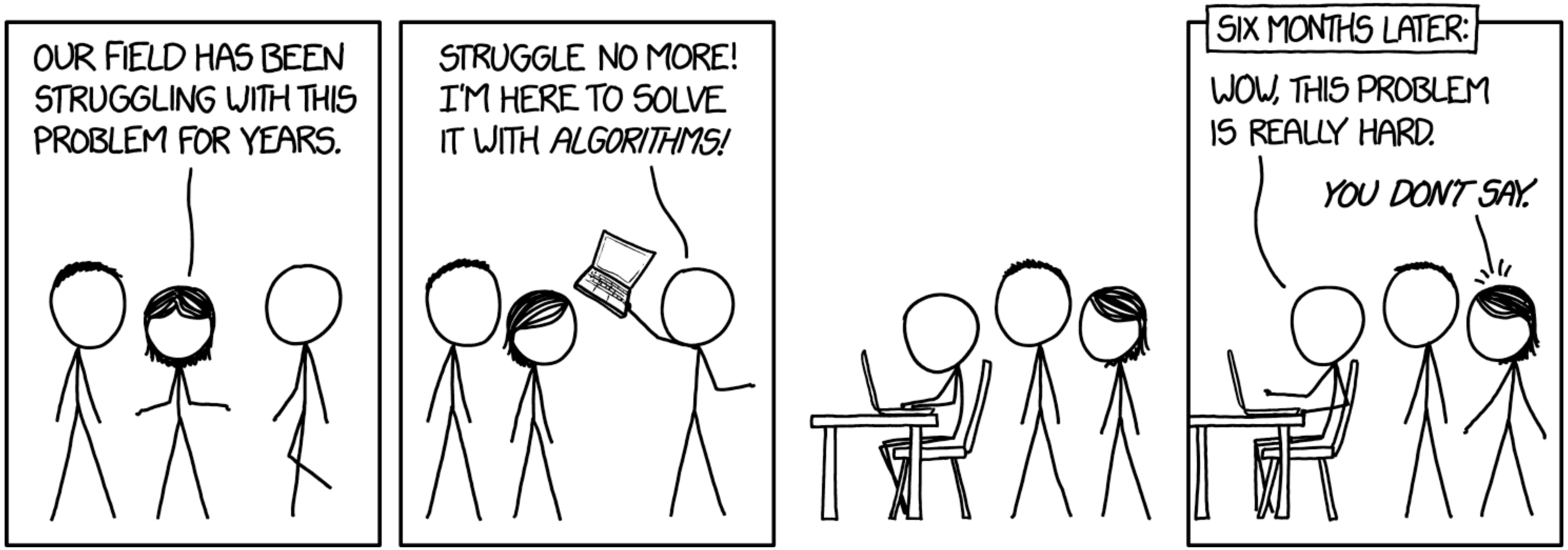
Overview of Complexity Theory

Algorithms vs Complexity Theory:



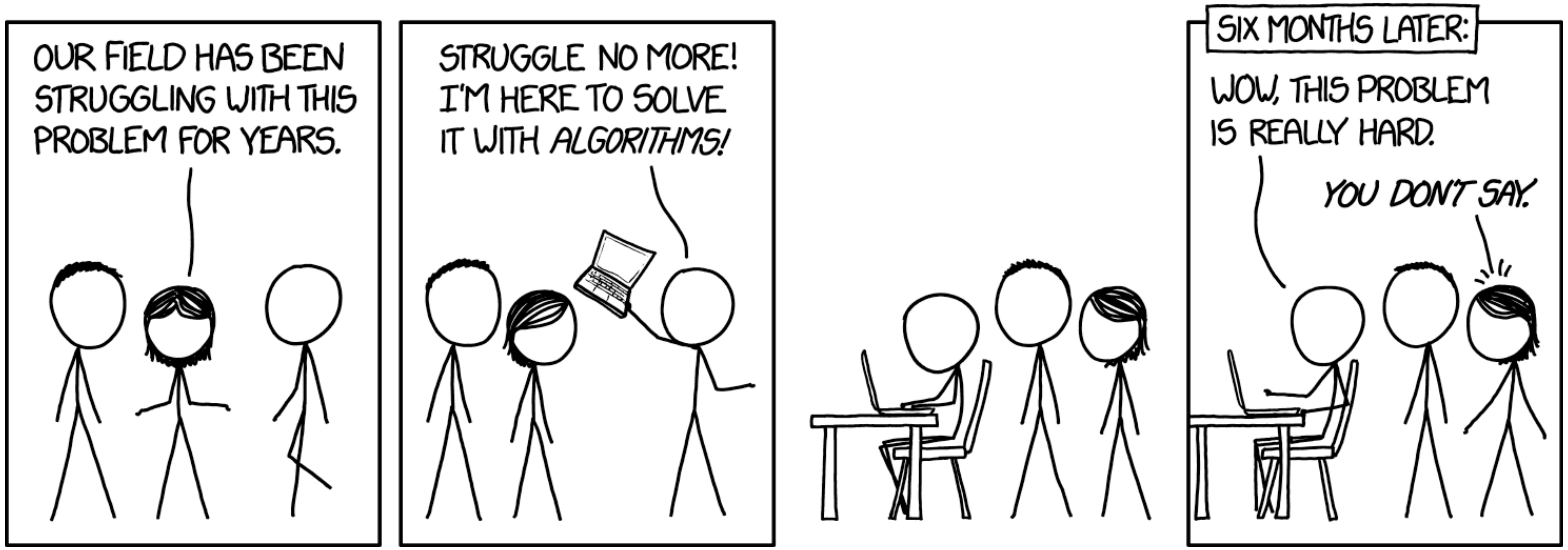
Overview of Complexity Theory

Algorithms vs Complexity Theory:



Overview of Complexity Theory

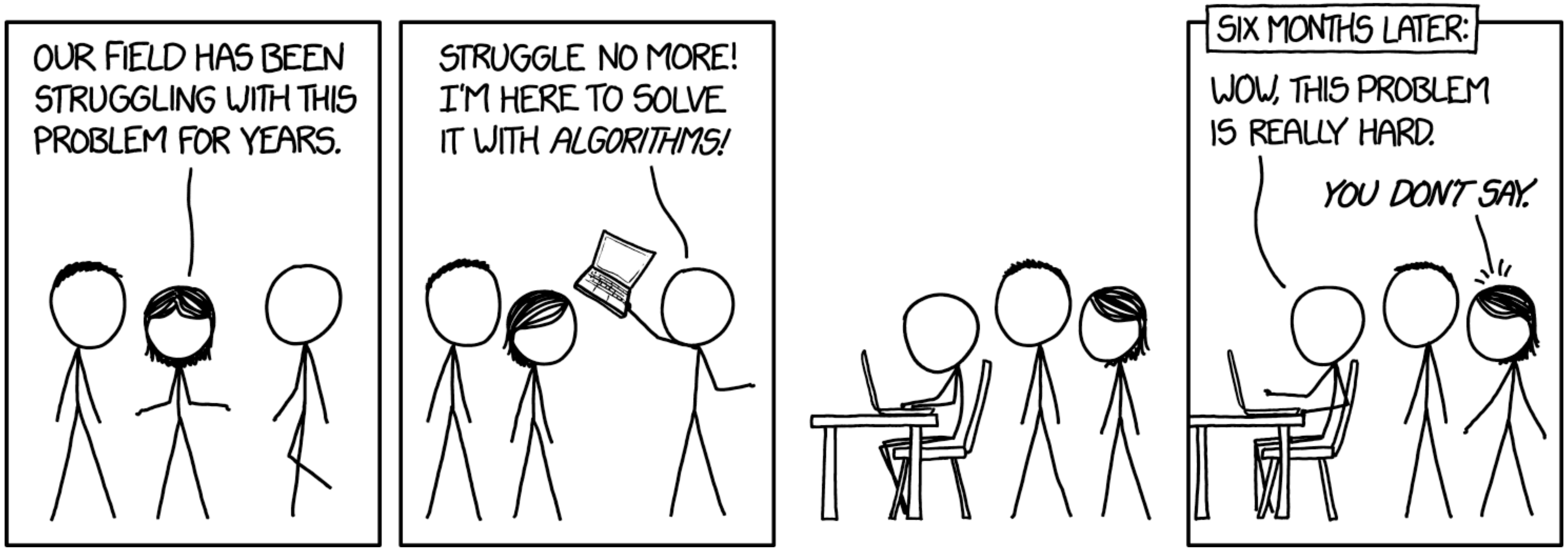
Algorithms vs Complexity Theory:



Algorithms focus on solving computation problems efficiently,

Overview of Complexity Theory

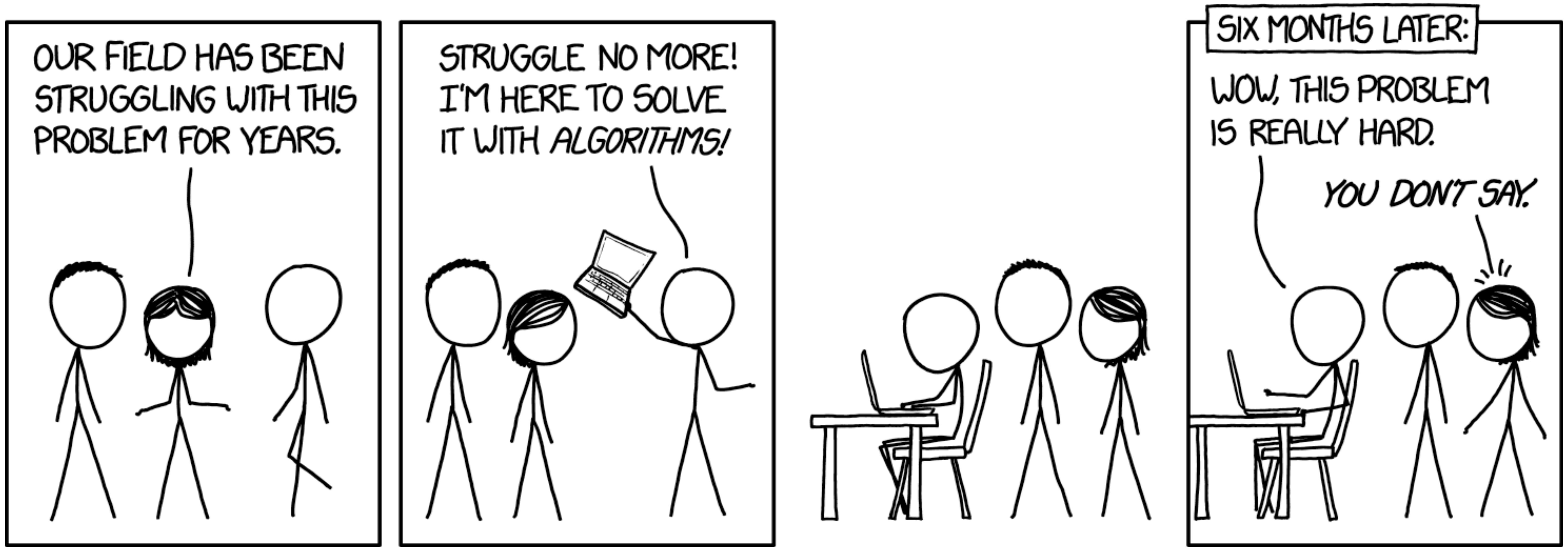
Algorithms vs Complexity Theory:



Algorithms focus on solving computation problems efficiently, while **Complexity theory**

Overview of Complexity Theory

Algorithms vs Complexity Theory:



Algorithms focus on solving computation problems efficiently, while **Complexity theory** studies inherent difficulty of problems.

Overview of Complexity Theory

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time.

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time. ⟵ Algorithms

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time. ⟵ Algorithms
- Prove that no algorithm exists that runs in less than T time.

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time. ⟵ Algorithms
- Prove that no algorithm exists that runs in less than T time. ⟵ Complexity Theory

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time. ⟵ Algorithms
- Prove that no algorithm exists that runs in less than T time. ⟵ Complexity Theory

Central Goal of Complexity Theory: Proving non-existence of efficient algorithms for

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time. ⟵ Algorithms
- Prove that no algorithm exists that runs in less than T time. ⟵ Complexity Theory

Central Goal of Complexity Theory: Proving non-existence of efficient algorithms for computational problems

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time. $\longleftarrow$ Algorithms
- Prove that no algorithm exists that runs in less than T time. $\longleftarrow$ Complexity Theory

Central Goal of Complexity Theory: Proving non-existence of efficient algorithms for computational problems w.r.t resources such as time,

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time. ⟵ Algorithms
- Prove that no algorithm exists that runs in less than T time. ⟵ Complexity Theory

Central Goal of Complexity Theory: Proving non-existence of efficient algorithms for computational problems w.r.t resources such as time, space,

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time. ⟵ Algorithms
- Prove that no algorithm exists that runs in less than T time. ⟵ Complexity Theory

Central Goal of Complexity Theory: Proving non-existence of efficient algorithms for computational problems w.r.t resources such as time, space, interactions,

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \,.\, y$.

- Design an algorithm to compute $x \,.\, y$ that runs in T time. ⟵ Algorithms
- Prove that no algorithm exists that runs in less than T time. ⟵ Complexity Theory

Central Goal of Complexity Theory: Proving non-existence of efficient algorithms for computational problems w.r.t resources such as time, space, interactions, randomness, etc.

Overview of Complexity Theory

Example: Given two numbers x and y, compute $x \cdot y$.

- Design an algorithm to compute $x \cdot y$ that runs in T time. ⟵ Algorithms
- Prove that no algorithm exists that runs in less than T time. ⟵ Complexity Theory

Central Goal of Complexity Theory: Proving non-existence of efficient algorithms for computational problems w.r.t resources such as time, space, interactions, randomness, etc.

Haven't been very successful

Overview of Complexity Theory

Overview of Complexity Theory

What we actually do in Complexity Theory:

Overview of Complexity Theory

What we actually do in Complexity Theory:

- Prove non-existence of efficient algorithms.

Overview of Complexity Theory

What we actually do in Complexity Theory:

- Prove non-existence of efficient algorithms. (E.g. $\mathit{GeneralisedChess} \notin \mathsf{P}$)

Overview of Complexity Theory

What we actually do in Complexity Theory:

- Prove non-existence of efficient algorithms. (E.g. *GeneralisedChess* $\notin$ P)
- Interrelate different complexity questions. For instance,

Overview of Complexity Theory

What we actually do in Complexity Theory:

- Prove non-existence of efficient algorithms. (E.g. *GeneralisedChess* $\notin$ P)
- Interrelate different complexity questions. For instance,

  Question: Are problems P_1 and P_2 not solvable in polynomial time?

Overview of Complexity Theory

What we actually do in Complexity Theory:

- Prove non-existence of efficient algorithms. (E.g. *GeneralisedChess* $\notin$ P)
- Interrelate different complexity questions. For instance,

  Question: Are problems P_1 and P_2 not solvable in polynomial time?

  Interrelation: P_1 is not solvable in poly. time

Overview of Complexity Theory

What we actually do in Complexity Theory:

- Prove non-existence of efficient algorithms. (E.g. *GeneralisedChess* $\notin$ P)
- Interrelate different complexity questions. For instance,

  Question: Are problems P_1 and P_2 not solvable in polynomial time?

  Interrelation: P_1 is not solvable in poly. time $\iff$ P_2 is not solvable in poly. time

Overview of Complexity Theory

What we actually do in Complexity Theory:

- Prove non-existence of efficient algorithms. (E.g. *GeneralisedChess* $\notin$ P)
- Interrelate different complexity questions. For instance,

  Question: Are problems P_1 and P_2 not solvable in polynomial time?

  Interrelation: P_1 is not solvable in poly. time $\iff$ P_2 is not solvable in poly. time
- Classify problems based on the amount of resources required to solve them and compare those classes.

Overview of Complexity Theory

What we actually do in Complexity Theory:

- Prove non-existence of efficient algorithms. (E.g. *GeneralisedChess* $\notin$ P)
- Interrelate different complexity questions. For instance,

  Question: Are problems P_1 and P_2 not solvable in polynomial time?

  Interrelation: P_1 is not solvable in poly. time $\iff$ P_2 is not solvable in poly. time
- Classify problems based on the amount of resources required to solve them and compare those classes.

  For instance, let X, Y, and Z be the set of problems solvable in logspace, polynomial time, and polynomial space, respectively.

Overview of Complexity Theory

What we actually do in Complexity Theory:

- Prove non-existence of efficient algorithms. (E.g. *GeneralisedChess* $\notin$ P)
- Interrelate different complexity questions. For instance,

  Question: Are problems P_1 and P_2 not solvable in polynomial time?

  Interrelation: P_1 is not solvable in poly. time $\iff$ P_2 is not solvable in poly. time
- Classify problems based on the amount of resources required to solve them and compare those classes.

  For instance, let X, Y, and Z be the set of problems solvable in logspace, polynomial time, and polynomial space, respectively. Then, $X \subseteq Y \subseteq Z$.

Glimpses of this Course

Glimpses of this Course

We’ll learn about the following and more in this course:

Glimpses of this Course

We'll learn about the following and more in this course:

- **P** vs **NP**:

Glimpses of this Course

We'll learn about the following and more in this course:

- **P** vs **NP**:

  P = Set of problems that are polynomial-time solvable.

Glimpses of this Course

We'll learn about the following and more in this course:

- **P** vs **NP**:

  P = Set of problems that are polynomial-time solvable.

  NP = Set of problems whose solutions are polynomial-time verifiable.

Glimpses of this Course

We'll learn about the following and more in this course:

- **P** vs **NP**:

  P = Set of problems that are polynomial-time solvable.

  NP = Set of problems whose solutions are polynomial-time verifiable.

  For instance,

Glimpses of this Course

We'll learn about the following and more in this course:

- **P** vs **NP**:

  P = Set of problems that are polynomial-time solvable.

  NP = Set of problems whose solutions are polynomial-time verifiable.

  For instance,

  PATH: Given a graph G and $u, v \in G$, decide if there is a path from u to v.

Glimpses of this Course

We'll learn about the following and more in this course:

- **P** vs **NP**:

  P = Set of problems that are polynomial-time solvable.

  NP = Set of problems whose solutions are polynomial-time verifiable.

  For instance,

  PATH: Given a graph G and $u, v \in G$, decide if there is a path from u to v.

  HAMPATH: Given a graph G, decide if G has a path that visits all the vertices of G.

Glimpses of this Course

We'll learn about the following and more in this course:

- **P** vs **NP**:

  P = Set of problems that are polynomial-time solvable.

  NP = Set of problems whose solutions are polynomial-time verifiable.

  For instance,

  PATH: Given a graph G and $u, v \in G$, decide if there is a path from u to v.

  HAMPATH: Given a graph G, decide if G has a path that visits all the vertices of G.

  PATH $\in$ **P** and *HAMPATH* $\in$ **NP**.

Glimpses of this Course

We'll learn about the following and more in this course:

- **P** vs **NP**:

  P = Set of problems that are polynomial-time solvable.

  NP = Set of problems whose solutions are polynomial-time verifiable.

  For instance,

  PATH: Given a graph G and $u, v \in G$, decide if there is a path from u to v.

  HAMPATH: Given a graph G, decide if G has a path that visits all the vertices of G.

  PATH $\in$ **P** and *HAMPATH* $\in$ **NP**.

- Are there problems solvable in $O(n^3)$ time that are not solvable in $O(n)$ time?

Glimpses of this Course

Glimpses of this Course

- Given a directed graph G and $u, v \in G$, can we find whether $u \rightsquigarrow v$ in logspace?

Glimpses of this Course

- Given a directed graph G and $u, v \in G$, can we find whether $u \rightsquigarrow v$ in logspace?

  Or is $\mathsf{L} = \mathsf{NL}$?

Glimpses of this Course

- Given a directed graph G and $u, v \in G$, can we find whether $u \rightsquigarrow v$ in logspace?
  Or is $\mathbf{L} = \mathbf{NL}$?

- Problems beyond **NP**. For instance,

Glimpses of this Course

- Given a directed graph G and $u, v \in G$, can we find whether $u \rightsquigarrow v$ in logspace?

  Or is $\mathbf{L} = \mathbf{NL}$?

- Problems beyond **NP**. For instance,

  INDSET: Given a graph G and $k \in \mathbb{Z}^+$, decide if G has an independent set of size k.

Glimpses of this Course

- Given a directed graph G and $u, v \in G$, can we find whether $u \rightsquigarrow v$ in logspace?

  Or is **L** = **NL**?

- Problems beyond **NP**. For instance,

  INDSET: Given a graph G and $k \in \mathbb{Z}^+$, decide if G has an independent set of size k.

  Easily verifiable solutions to *INDSET* exist. (*INDSET* ∈ **NP**)

Glimpses of this Course

- Given a directed graph G and $u, v \in G$, can we find whether $u \rightsquigarrow v$ in logspace?

  Or is **L** = **NL**?

- Problems beyond **NP**. For instance,

  INDSET: Given a graph G and $k \in \mathbb{Z}^+$, decide if G has an independent set of size k.

  Easily verifiable solutions to *INDSET* exist. (*INDSET* $\in$ **NP**)

  EXACT-INDSET: Given a graph G and $k \in \mathbb{Z}^+$, decide if the size of the largest independent set of G is k.

Glimpses of this Course

- Given a directed graph G and $u, v \in G$, can we find whether $u \rightsquigarrow v$ in logspace?

  Or is $\mathbf{L} = \mathbf{NL}$?

- Problems beyond **NP**. For instance,

  INDSET: Given a graph G and $k \in \mathbb{Z}^+$, decide if G has an independent set of size k.

  Easily verifiable solutions to *INDSET* exist. (*INDSET* $\in$ **NP**)

  EXACT-INDSET: Given a graph G and $k \in \mathbb{Z}^+$, decide if the size of the largest independent set of G is k.

  Easily verifiable solutions to *EXACT-INDSET* seem to not exist. (*EXACT-INDSET* $\in \Sigma_2^p$)

Glimpses of this Course

Glimpses of this Course

- Can we use randomness to speed up the computation?

Glimpses of this Course

- Can we use randomness to speed up the computation?

  P = Set of problems that are polytime solvable by deterministic algorithm.

Glimpses of this Course

- Can we use randomness to speed up the computation?

  P = Set of problems that are polytime solvable by deterministic algorithm.

  BPP = Set of problems that are polytime solvable by probabilistic algorithm.

Glimpses of this Course

- Can we use randomness to speed up the computation?

  P = Set of problems that are polytime solvable by deterministic algorithm.

  BPP = Set of problems that are polytime solvable by probabilistic algorithm.

  For instance,

Glimpses of this Course

- Can we use randomness to speed up the computation?

  P = Set of problems that are polytime solvable by deterministic algorithm.

  BPP = Set of problems that are polytime solvable by probabilistic algorithm.

For instance,

PRIMES: Is x prime?

Glimpses of this Course

- Can we use randomness to speed up the computation?

  P = Set of problems that are polytime solvable by deterministic algorithm.

  BPP = Set of problems that are polytime solvable by probabilistic algorithm.

  For instance,

  PRIMES: Is x prime? (Is in **BPP**. Was shown to be in **P** after a long effort. [AKS'02])

Glimpses of this Course

- Can we use randomness to speed up the computation?

  P = Set of problems that are polytime solvable by deterministic algorithm.

  BPP = Set of problems that are polytime solvable by probabilistic algorithm.

  For instance,

  PRIMES: Is x prime? (Is in **BPP**. Was shown to be in **P** after a long effort. [AKS'02])

  PIT (Polynomial Identity Testing): Given a multivariate polynomial with integer

Glimpses of this Course

- Can we use randomness to speed up the computation?

  P = Set of problems that are polytime solvable by deterministic algorithm.

  BPP = Set of problems that are polytime solvable by probabilistic algorithm.

  For instance,

  PRIMES: Is x prime? (Is in **BPP**. Was shown to be in **P** after a long effort. [AKS'02])

  PIT (Polynomial Identity Testing): Given a multivariate polynomial with integer coefficients, find whether there is an assignment of values to variables such that

Glimpses of this Course

- Can we use randomness to speed up the computation?

  P = Set of problems that are polytime solvable by deterministic algorithm.

  BPP = Set of problems that are polytime solvable by probabilistic algorithm.

  For instance,

  PRIMES: Is x prime? (Is in **BPP**. Was shown to be in **P** after a long effort. [AKS'02])

  PIT (Polynomial Identity Testing): Given a multivariate polynomial with integer coefficients, find whether there is an assignment of values to variables such that polynomial evaluates to non-zero.

Glimpses of this Course

- Can we use randomness to speed up the computation?

  P = Set of problems that are polytime solvable by deterministic algorithm.

  BPP = Set of problems that are polytime solvable by probabilistic algorithm.

  For instance,

  PRIMES: Is x prime? (Is in **BPP**. Was shown to be in **P** after a long effort. [AKS'02])

  PIT (Polynomial Identity Testing): Given a multivariate polynomial with integer coefficients, find whether there is an assignment of values to variables such that polynomial evaluates to non-zero. (Is in **BPP**, but not known to be in **P**.)

Administrative Details

Administrative Details

Grading:

Administrative Details

Grading:

- $20\,\%$ - Project (a presentation on a paper/topic in groups of two)

Administrative Details

Grading:

- 20% - Project (a presentation on a paper/topic in groups of two)
- 20% - Best 2 out of 3 quizzes (Mostly MCQs and T/F)

Administrative Details

Grading:

- $20\,\%$ - Project (a presentation on a paper/topic in groups of two)
- $20\,\%$ - Best 2 out of 3 quizzes (Mostly MCQs and T/F)
- $30\,\%$ - Minors (15% each)

Administrative Details

Grading:

- $20\,\%$ - Project (a presentation on a paper/topic in groups of two)
- $20\,\%$ - Best 2 out of 3 quizzes (Mostly MCQs and T/F)
- $30\,\%$ - Minors (15% each)
- $30\,\%$ - Major

Administrative Details

Grading:

- $20\,\%$ - Project (a presentation on a paper/topic in groups of two)
- $20\,\%$ - Best 2 out of 3 quizzes (Mostly MCQs and T/F)
- $30\,\%$ - Minors (15% each)
- $30\,\%$ - Major
- $0\,\%$ - Problem sets with solutions or solution links

Administrative Details

Grading:

- $20\,\%$ - Project (a presentation on a paper/topic in groups of two)
- $20\,\%$ - Best 2 out of 3 quizzes (Mostly MCQs and T/F)
- $30\,\%$ - Minors (15% each)
- $30\,\%$ - Major
- $0\,\%$ - Problem sets with solutions or solution links

Book: Computational Complexity: A Modern Approach by Arora and Barak

Administrative Details

Grading:

- $20\,\%$ - Project (a presentation on a paper/topic in groups of two)
- $20\,\%$ - Best 2 out of 3 quizzes (Mostly MCQs and T/F)
- $30\,\%$ - Minors (15% each)
- $30\,\%$ - Major
- $0\,\%$ - Problem sets with solutions or solution links

Book: Computational Complexity: A Modern Approach by Arora and Barak

Office Hours: Mail me to fix an appointment

Administrative Details

Grading:

- $20\,\%$ - Project (a presentation on a paper/topic in groups of two)
- $20\,\%$ - Best 2 out of 3 quizzes (Mostly MCQs and T/F)
- $30\,\%$ - Minors (15% each)
- $30\,\%$ - Major
- $0\,\%$ - Problem sets with solutions or solution links

Book: Computational Complexity: A Modern Approach by Arora and Barak

Office Hours: Mail me to fix an appointment

Course Site: http://home.iitj.ac.in/~vimalraj/courses/ct/csl7140.html