Deterministic and Non-Deterministic

Dr. Animesh Chaturvedi

Assistant Professor: **IIIT Dharwad**

Young Researcher: **Heidelberg Laureate Forum**

and **Pingala Interaction in Computing**

Young Scientist: **Lindau Nobel Laureate Meetings**

Postdoc: **King's College London** & **The Alan Turing Institute**

PhD: **IIT Indore** MTech: **IIITDM Jabalpur**

Computational complexity theory

- Fields in
  - Theoretical Computer Science
  - Analysis of Algorithms
- An algorithm solves a computation problem by mathematical steps.
- A computational problem (such as an algorithm) is a task solved by a computer.
- Focuses on classifying computational problems according to the resource usage
- Resource usage: amount of resources needed to solve computational problem,
- Resources: such as time and storage.

https://en.wikipedia.org/wiki/Computational_complexity_theory

Single Source Shortest-Paths Implementation

algorithm	restriction	typical case	worst case	extra space
topological sort	no directed cycles	$E + V$	$E + V$	V
Dijkstra (binary heap)	no negative weights	$E \log V$	$E \log V$	V
Bellman–Ford	no negative cycles	$E\,V$	$E\,V$	V
Bellman–Ford (queue–based)		$E + V$	$E\,V$	V

Easy

Medium

Hard

Remark 1. Directed cycles make the problem harder.

Remark 2. Negative weights make the problem harder.

Remark 3. Negative cycles makes the problem intractable.

Single Source Shortest-Paths Implementation

algorithm	restriction	typical case	worst case	extra space
topological sort	no directed cycles	$E + V$	$E + V$	V
Dijkstra (binary heap)	no negative weights	$E \log V$	$E \log V$	V
Bellman–Ford	no negative cycles	$E\,V$	$E\,V$	V
Bellman–Ford (queue-based)		$E + V$	$E\,V$	V

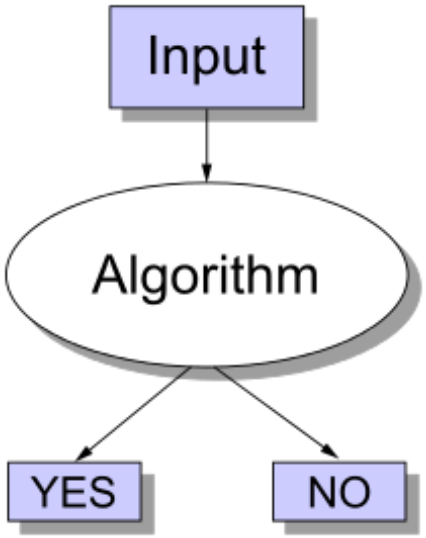
Remark 1. Directed cycles make the problem harder.

Remark 2. Negative weights make the problem harder.

Remark 3. Negative cycles makes the problem intractable.

Deterministic algorithm

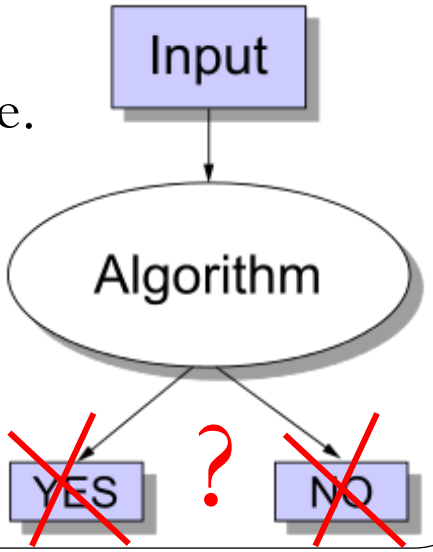
- Given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states.
- State machine: a state describes what a machine is doing at a particular instant in time.
- State machines pass in a discrete manner from one state to another.
- Enter the input, initial state or start state.
- Current state determines what will be next state, the set of states is predetermined.



https://en.wikipedia.org/wiki/Deterministic_algorithm

Non-deterministic Algorithms

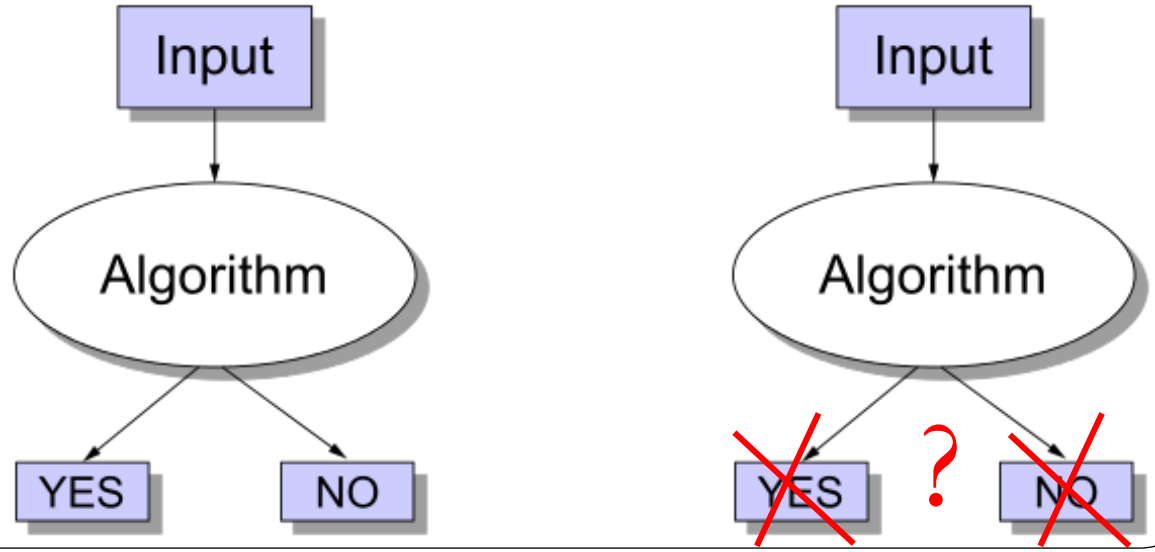
- If it uses external state other than the input, such as
  - user input,
  - a global variable,
  - a hardware timer value,
  - a random value, or
  - stored disk data.
- If it is timing-sensitive,
  - e.g. if it has multiple processors writing to the same data at the same time.
- If a hardware error causes its state to change in an unexpected way.
- The order each processor writes data will affect the result.



https://en.wikipedia.org/wiki/Deterministic_algorithm

Deterministic and Non-deterministic Algorithms

- Disadvantages of Determinism
  - predictable future by players or predictable security by hacker
  - e.g. predictable card shuffling program or security key
- Pseudorandom number generator is often not sufficient,
  - thus cryptographically secure pseudo-random number generator,
  - hardware random number generator.



https://en.wikipedia.org/wiki/Deterministic_algorithm

P (Polynomial) Time Problems

- Contains all decision problems that can be solved deterministically using a polynomial time (polynomial amount of computation time).
- A problem is in P-complete, if it is in P
- P is the class of computational problems that are "efficiently solvable" or "tractable".
- Class P, typically take all the "tractable" problems for a sequential algorithm,
- But, in reality, some problems not known to be solved in polynomial P time.

https://en.wikipedia.org/wiki/P_(complexity)
https://en.wikipedia.org/wiki/P-complete

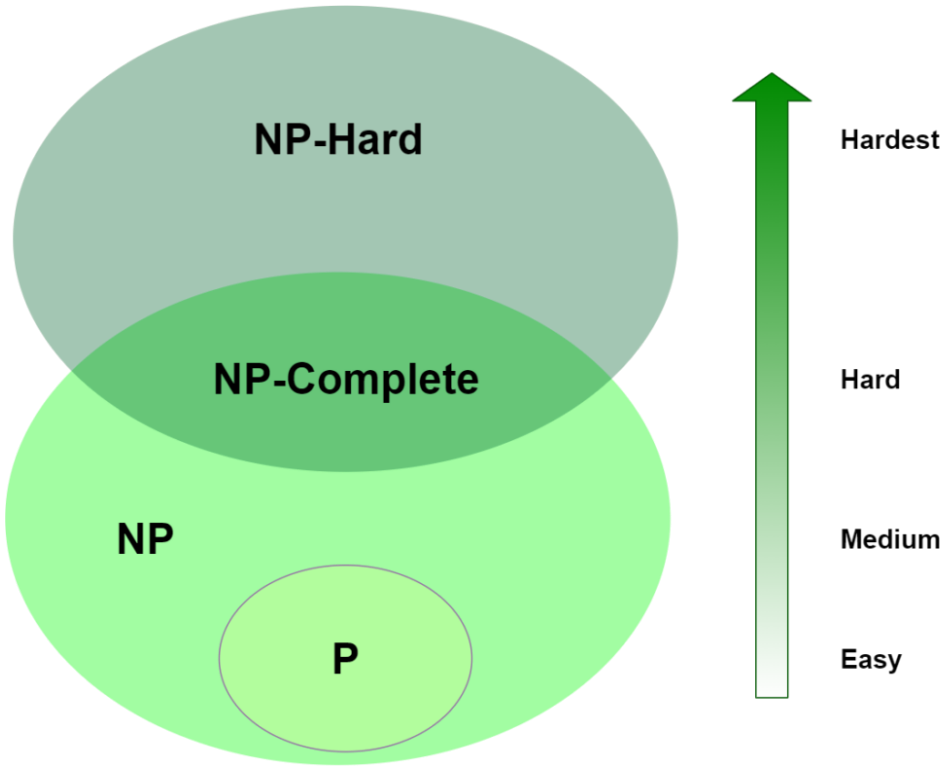
P (Polynomial) Time Problems

- Programmable Function (or method) is polynomial-time
  - if completes in constant-time or polynomial time,
  - then the entire algorithm takes polynomial time.
- Polynomial-time algorithms:
  - Minimum Spanning Tree: Kruskal's $\mathbf{O(}E \lg V\mathbf{)}$ and Prim's $\mathbf{O(}E + V \lg V\mathbf{)}$ algorithm
  - Shortest Path Algorithms: Djikstra's **O(E lgV)** and Bellman-Ford's $\mathbf{O(}EV\mathbf{)}$ algorithm

https://en.wikipedia.org/wiki/P_(complexity)
https://en.wikipedia.org/wiki/P-complete

NP - Naming convention

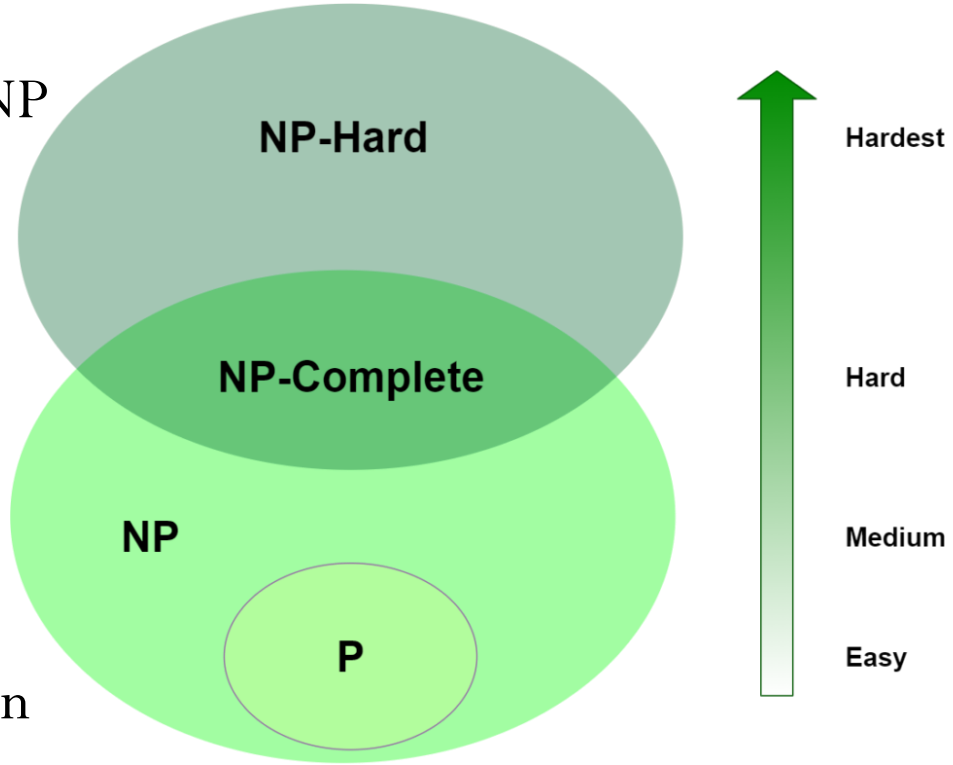
- Classification
  - Hardest → NP-Hard
  - Hard → NP-Complete
  - Medium → NP
  - Easy →P
- Order of N inputs
  - O(1) – constant-time
  - $O(\log_2(n))$ – logarithmic-time
  - O(n) – linear-time
  - $O(n^2)$ – quadratic-time
  - $O(n^k)$ – polynomial-time
  - $O(k^n)$ – exponential-time
  - O(n!) – factorial-time



https://www.baeldung.com/cs/p-np-np-complete-np-hard

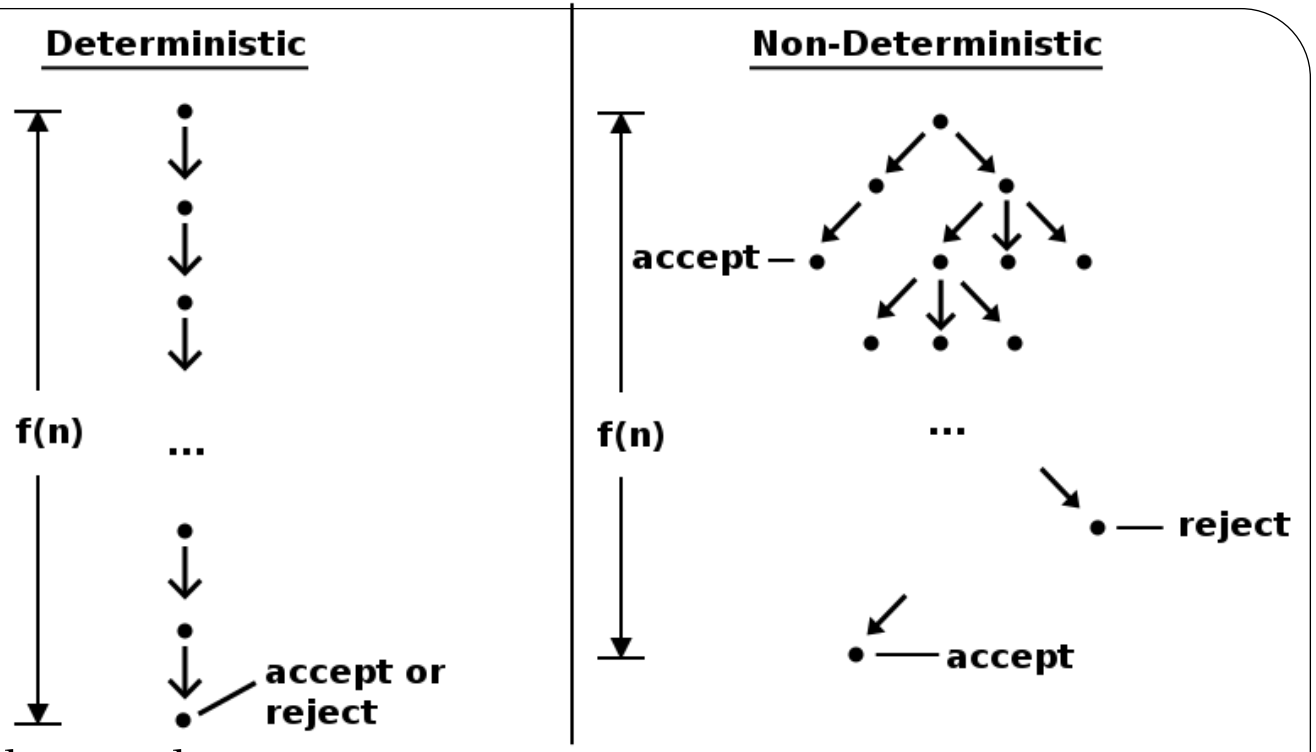
NP - Naming convention

- NP-hard: Class of problems are at least as hard as the hardest problems in NP.
- NP-hard problems do not have to be in NP; means NP hard problem may not even be decidable.
- NP-complete: Class of decision problems which contains the hardest problems in NP. Each NP-complete problem has to be in NP.
- NP: Class of computational decision problems for which any given yes-solution can be verified as a solution in polynomial time
- NP-easy: At most as hard as NP, but not necessarily in NP.



https://en.wikipedia.org/wiki/NP-hardness
https://www.baeldung.com/cs/p-np-np-complete-np-hard

P and NP Problems



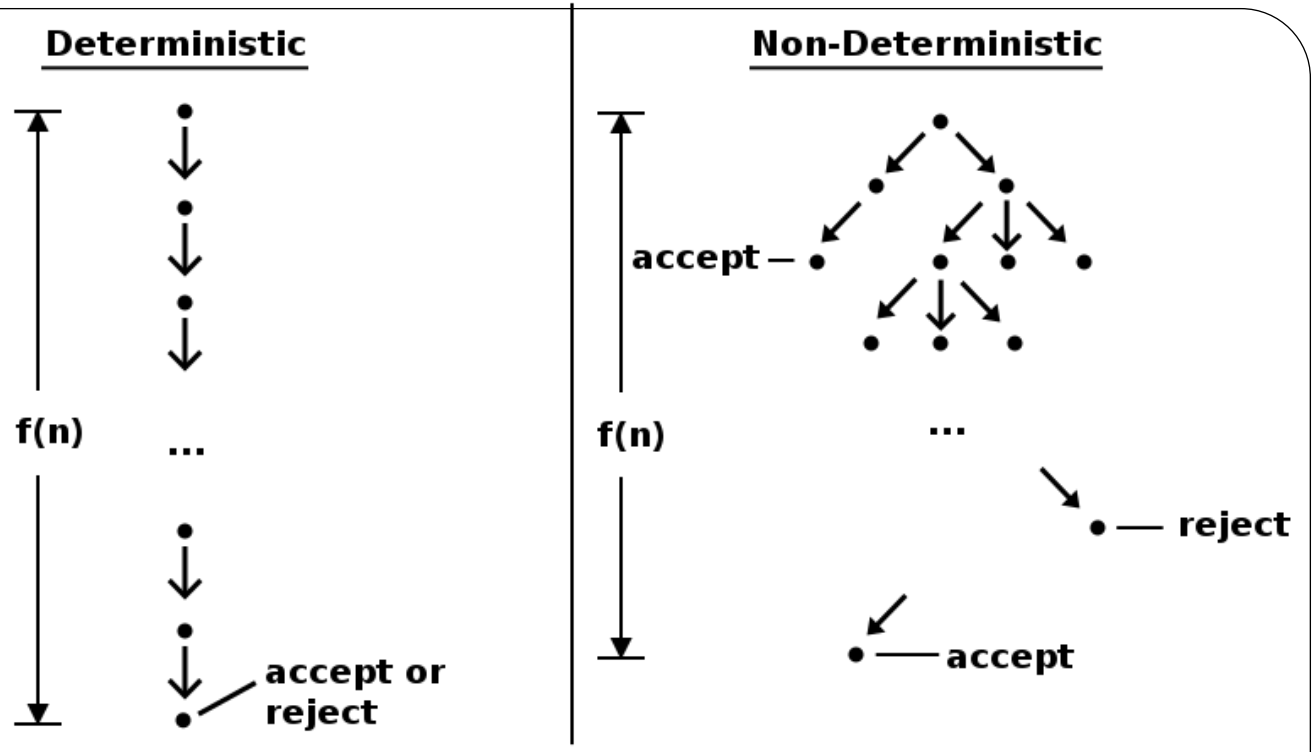
- Nondeterministic Polynomial-time
- “Nondeterministic” refers to
  - “luckiest possible guesser”
- "Complete" refers to
  - “in the same complexity class”
- **P** versus **NP** determine
  - whether a problem can be verified in polynomial time
  - whether the problem can also be solved in polynomial time.
- If it turned out that **P** ≠ **NP**, (widely accepted/believed),
- There are problems in **NP** that are harder to compute than to verify:
- NP problems could not be solved in polynomial time, but the answer could be verified in polynomial time.

https://en.wikipedia.org/wiki/P_versus_NP_problem

NP Complete

- Nondeterministic Polynomial-time

Complete

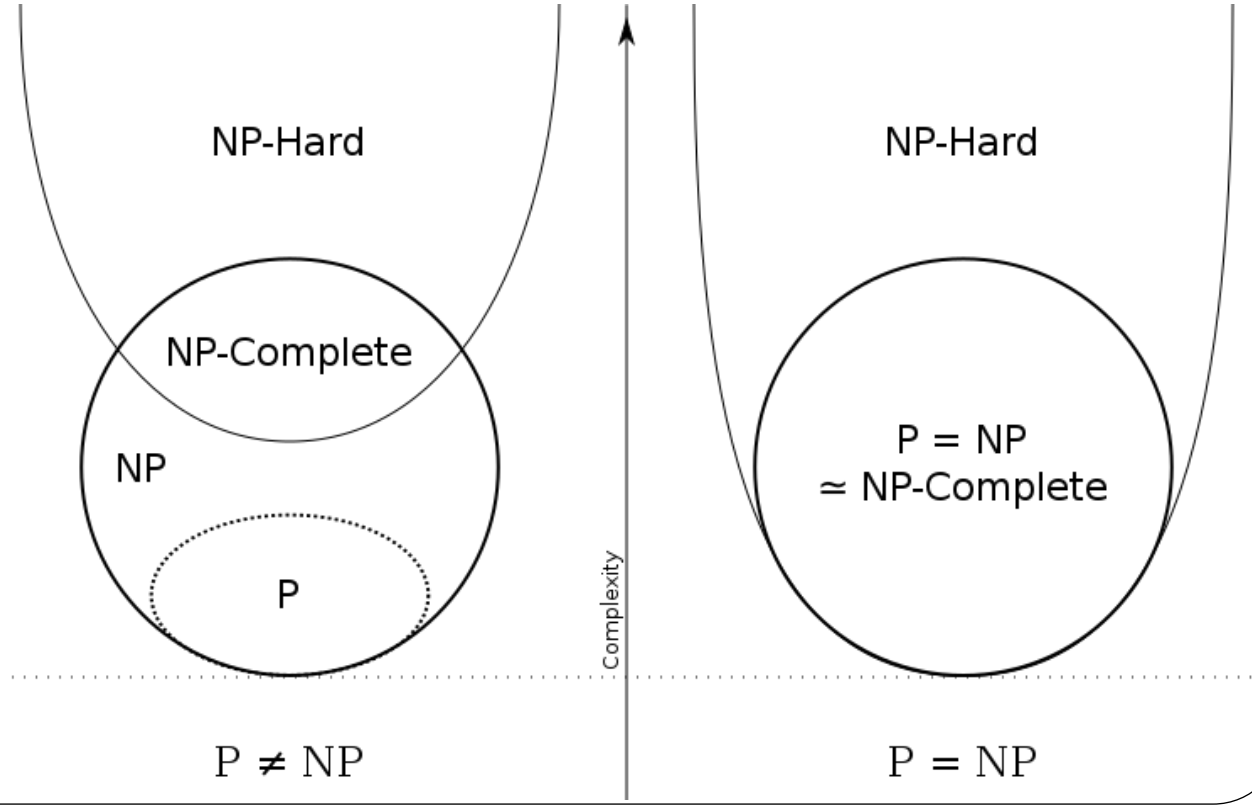


- A problem is NP-complete when:
  - a brute-force search algorithm can solve it, and the correctness of each solution can be verified quickly, and
  - the problem can be used to simulate any other problem with similar solvability.
- NP-complete problem can be *verified* "quickly",
- There is no known way to *find* a solution quickly.

https://en.wikipedia.org/wiki/NP-completeness

NP - Hard Problems

- Non-Deterministic Polynomial-time hardness
- At least as hard as the hardest problems in NP
- The might be some polynomial-time algorithms for NP-hard problems but might not been discovered yet
- NP-hard but not NP-complete
  - halting problem: "given a program and its input, will it run forever?"
  - traveling salesman problem



https://en.wikipedia.org/wiki/NP-hardness

References

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (Vol. 3, pp. 624-642). Cambridge: MIT press.
- Stuart Russel, and Peter Norvig. "Artificial intelligence: A modern approach. Third edit." Upper Saddle River, New Jersey 7458 (2015). http://aima.cs.berkeley.edu/
- Wikipedia pages https://www.wikipedia.org/
- Images are from several sources e.g. movies, TV serials, internet, miscellaneous links, slides, blogs, etc.

תודה רבה
Hebrew

Ευχαριστώ
Greek

Спасибо
Russian

Danke
German

धन्यवादः
Sanskrit

Merci
French

ধন্যবাদ
Bangla

நன்றி
Tamil

شكراً
Arabic

ಧನ್ಯವಾದಗಳು
Kannada

Thank You
English

നന്ദി
Malayalam

多謝
Traditional Chinese

Grazie
Italian

ధన్యవాదాలు
Telugu

આભાર
Gujarati

ਧੰਨਵਾਦ
Punjabi

धन्यवाद
Hindi & Marathi

Gracias
Spanish

多谢
Simplified Chinese

https://sites.google.com/site/animeshchaturvedi07

Obrigado
Portuguese

ありがとうございました
Japanese

ขอบคุณ
Thai

감사합니다
Korean