CS211FZ: Data Structures and Algorithms II
12- NP-Problem
NP-Completeness

LECTURER: Chris Roadknight

Chris.Roadknight@mu.ie

Introduction

Introduction (1)

- Computational complexity studies the time and space resources needed to solve computational problems.

- Understanding complexity helps in algorithm design and predicting computational limits.

Introduction (2)

- Almost all the algorithms we have studied thus far have been ***polynomial-time algorithms***: on inputs of size n, their worst-case running time is $O(n^k)$ for some constant k.

- You might wonder whether all problems can be solved in polynomial time?

- The answer is no.

Introduction (3)

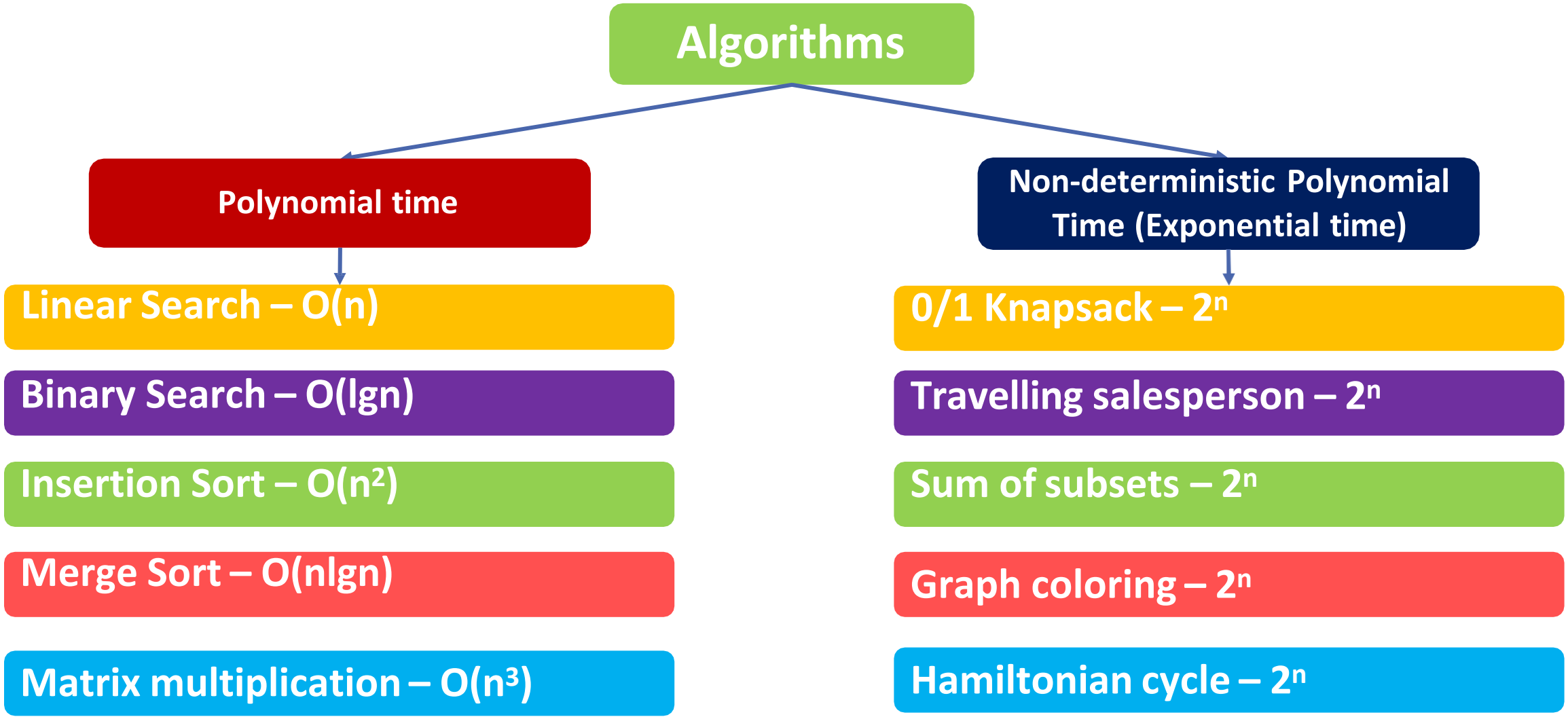
- For example, there are problems, such as Turing's famous "**Halting Problem**"*, that cannot be solved by any computer, no matter how long you're willing to wait for an answer.
- The question is simply whether the given program will ever halt on a particular input.

> For example, in pseudocode, the program
>
> ```
> while (true) continue
> ```
>
> does not halt; rather, it goes on forever in an infinite loop. On the other hand, the program
>
> ```
> print "Hello, world!"
> ```
>
> does halt.

*https://en.wikipedia.org/wiki/Halting_problem

Algorithm Classes

Polynomial Time vs Non-deterministic Polynomial Time (NP)



Polynomial Time vs Non-deterministic Polynomial Time (NP)

Deterministic Algorithms

- Deterministic algorithms are those where the behavior is completely predictable.
- Given a particular input, the algorithm will always produce the same output, following the same sequence of steps.
- Linear Search and Binary Search are deterministic algorithms.

```
1 def linear_search(array, target):
2     for i in range(len(array)):
3         if array[i] == target:
4             return i  # target found at index i
5     return -1  # target not found
```

Polynomial Time vs Non-deterministic Polynomial Time (NP)

Non-Deterministic Algorithms

- Non-deterministic algorithms, on the other hand, can have multiple possible paths of execution. Involve some level of randomness or "guessing.
- An example of Non-deterministic algorithm.

```
1  A=[9,8,7,6,4,11];
2  Irfan Search(A, n, key)
3  {
4    i = choice(1, n); // non-deterministic choice
5    if (A[i] == key)
6    {
7      print(i); // Success
8    }
9    else
10   {
11     print(0); // Failure
12   }
13 }
```

Polynomial Time vs Non-deterministic Polynomial Time (NP)

Non-Deterministic Algorithm Example

- Non-deterministic Choice: i = choice(1, n) implies that the algorithm can magically choose the correct index.

- In reality, such a non-deterministic choice is not practically feasible but is used to understand theoretical properties of problems and algorithms.

- Polynomial Time: Once the correct choice is made, checking A[i] against the key is O(1), implying constant time. If all choices are guessed correctly in polynomial time, the problem is in class NP.

Classes of Algorithms

- Commonly used algorithm classes:
  - P Class
  - NP Class
  - NP-Complete (NPC)

P Class

- The class P consists of those problems that are solvable in polynomial time.
- More specifically, they are problems that can be solved in $O(n^k)$ time for some constant k, where n is the size of the input to the problem.
- P is often a class of computational problems that are solvable and tractable.
- Tractable means that the problems can be solved in theory as well as in practice.

P Class - Example

- For example, most of the problems we studies in this course belongs to P class.

Linear Search – O(n)

Binary Search – O(lgn)

Insertion Sort – $O(n^2)$

Merge Sort – O(nlgn)

NP Class

- The name “NP” stands for “nondeterministic polynomial time”.
- The class NP consists of those problems that cannot be solved in polynomial time but can be “**verified**” in polynomial time.
- What do we mean by a problem being verified?
- The solutions of the NP class are hard to find. However, If we were somehow given a “certificate” of a solution, we can quickly (in polynomial time) test whether a solution is correct (without worrying about how hard it might be to find the solution).

NP Class - Example

- For example, the following problems take exponential time instead of polynomial time to solve, but once we know the solution, we can verify in polynomial time.

0/1 Knapsack – 2^n

Travelling salesperson – 2^n

Sum of subsets – 2^n

Graph coloring – 2^n

Hamiltonian cycle – 2^n

NP Class - Example

- Consider an example of typical sudoku. To find the solution, the time increases exponentially. However, once the solution is known, we can verify in polynomial time.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A typical Sudoku puzzle

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

The solution to the puzzle

NPC Class

- A problem belongs to the class NP-complete - and we call it NPC- if it belongs to NP and is as ”hard” as any problem in NP.

- **We state that if any NP-complete problem can be solved in polynomial time, then every problem in NP has a polynomial-time algorithm.**

- No polynomial-time algorithm has yet been discovered for an NP-complete problem, nor has anyone yet been able to prove that no polynomial-time algorithm can exist for any one of them.

NPC Class

- For example, all the following NP problems are NPC. If any of these problem can be solved in polynomial time, then every problem in NP can be solved with a polynomial-time algorithm.

0/1 Knapsack – 2^n

Travelling salesperson – 2^n

Sum of subsets – 2^n

Graph coloring – 2^n

Hamiltonian cycle – 2^n

NPC Class

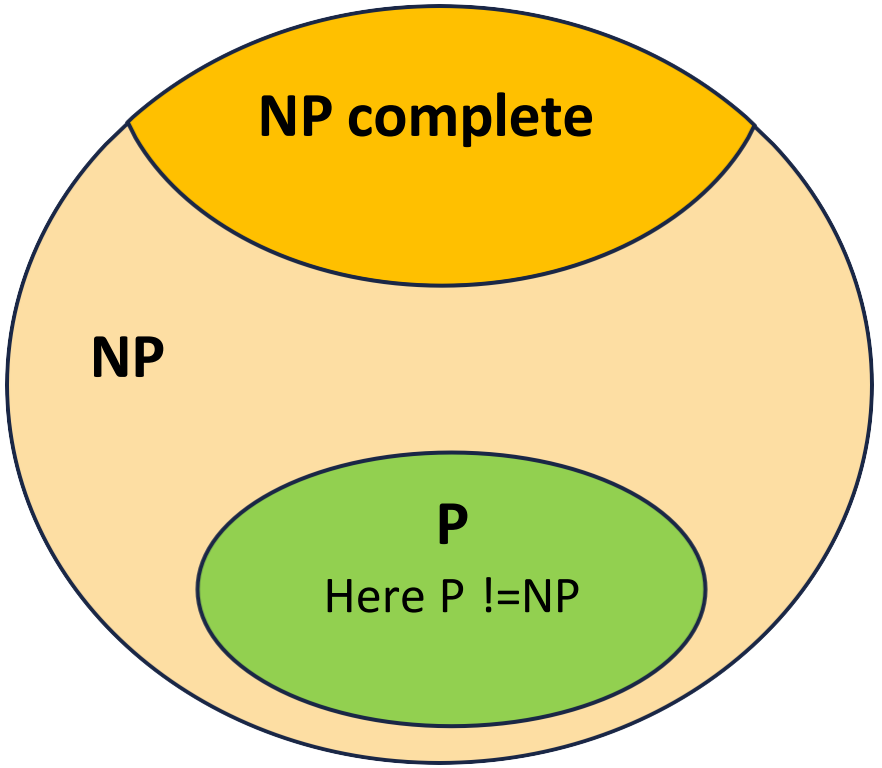
- Most theoretical computer scientists believe that the NP-complete problems are intractable.

- The problems that can be solved in theory but not in practice are known as intractable.

- Since given the wide range of NP-complete problems that have been studied to date - without anyone having discovered a polynomial-time solution to any of them - it would be truly astounding if all of them could be solved in polynomial time.

P, NP, and NPC Class

- P problems are solvable in polynomial time, NP problems are verifiable in polynomial time, and NP-complete problems are the hardest of NP problems.



Why study of the algorithms' classes important

- To become a good algorithm designer, you must understand the basics of the theory of NP-completeness. If you can establish a problem as NP-complete, you provide good evidence for its intractability.
- As an engineer, you would then do better to spend your time developing an approximation algorithm or solving a tractable special case, rather than searching for a fast algorithm that solves the problem exactly.
- Moreover, many natural and interesting problems that on the surface seem no harder than sorting, graph searching, or network flow are in fact NP-complete. Therefore, you should become familiar with this remarkable class of problems.