Complexity Classes and Performance Metrics

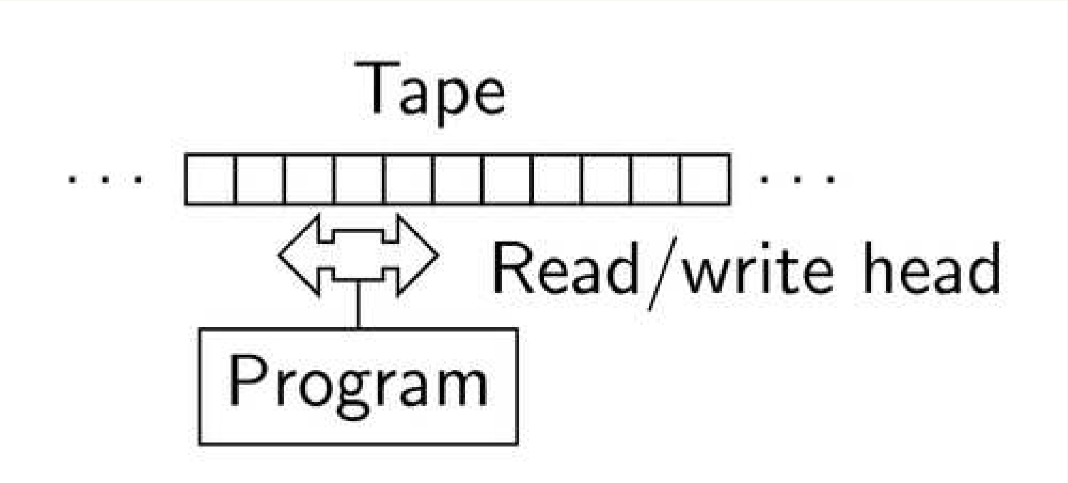
Quantum Capita Selecta

Amsterdam University of Applied Sciences

Bernardo Villalba Frías, PhD

b.r.villalba.frias@hva.nl

Turing Machine



- Mathematical model of computation
- Abstract machine that manipulates symbols
- Infinite memory tape divided into cells
- Church–Turing thesis
  - If a problem can be solved by an algorithm, there exists a Turing machine that solves it

Computability

Amsterdam University of Applied Sciences

- Theoretical limits of computer programming
- Understand which tasks:
  - Can be performed (computable)
    - Easy to establish
    - Write/test a program to solve the task
  - Cannot be performed (non computable)
    - Harder to prove
    - Rule out every possibility
    - Even with unlimited resources:
      - Memory
      - Time

Computational problems

Amsterdam University of Applied Sciences

- Decision problems:
  - Make decision about problems
    - Is p a prime number?
  - `yes/no` question of the input values
- Search problems:
  - Search for solutions to problems:
    - Find a prime factor of an integer N
- Optimization problems:
  - Find the best solution from all feasible solutions

Undecidable problem

- No program can always give the right answer
  - Sometimes gives wrong answers
  - Sometimes runs forever without any answer
- Examples:
  - The halting problem:

    Given a computer program and an input, will the program terminate or will it run forever?

  - Hilbert’s Entscheidungsproblem:

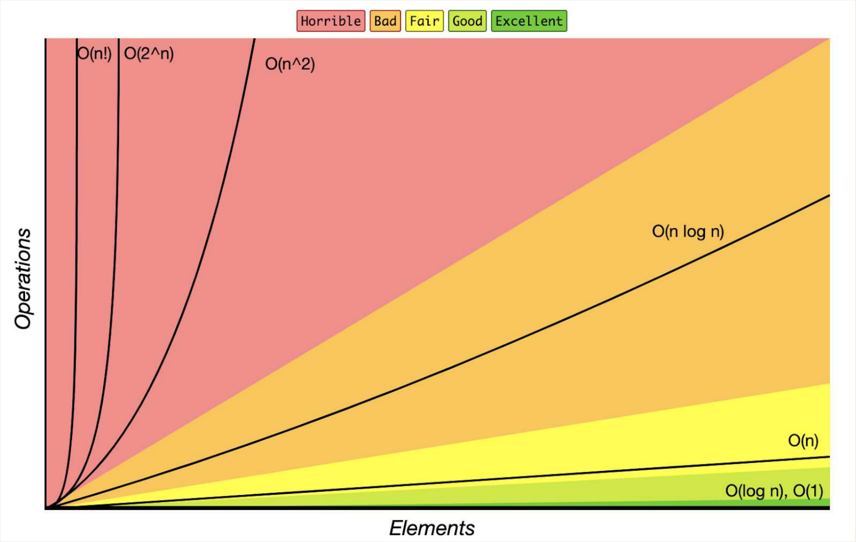
    Is there an algorithm that can decide in a finite amount of steps whether any given mathematical statement is provable given some fixed system of axioms?

Tractability

Amsterdam University of Applied Sciences

- Establish a basic classification for decidable problems:
  - Tractable:
    - Easy, efficient
    - Can be solved in polynomial time
  - Intractable:
    - Hard, inefficient
    - Can be solved in non–polynomial (exponential or worst) time

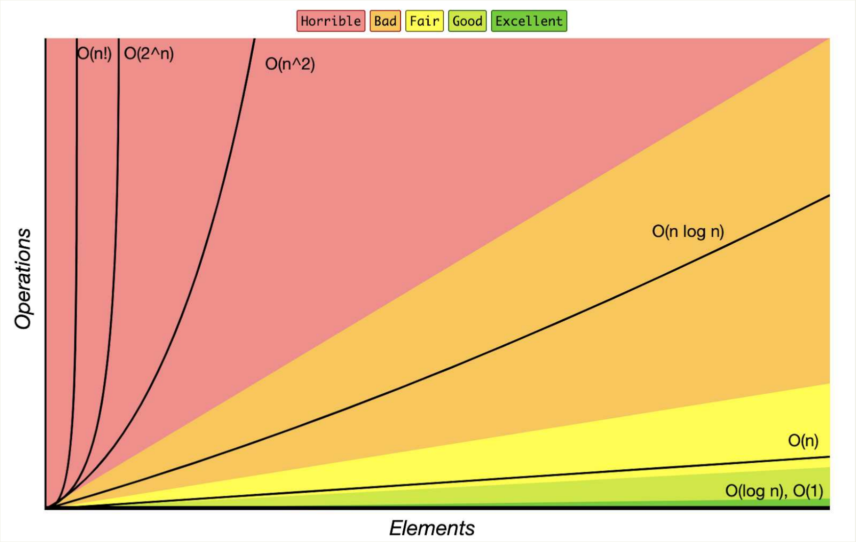
Computational complexity



- Problems vs Algorithms:
  - “Given a set of n numbers, sort the numbers in increasing numerical order”
    - Insertion sort algorithm
    - Merge sort algorithm

Computational complexity

Amsterdam University of Applied Sciences



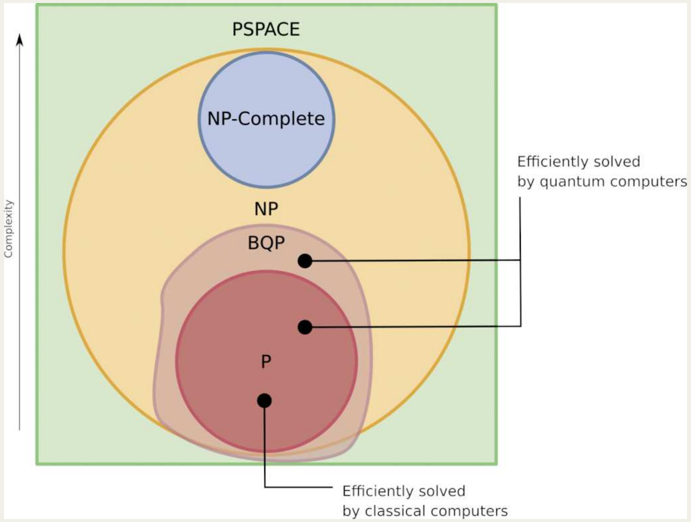
- Analysis of algorithms
  - Complexity of explicitly given algorithms
  - Big–O notation
  - Expressed as a function of the input size

Computational complexity

Big O Notation Summary

Notation	Type	Examples	Description
O(1)	Constant	Hash table access	Remains constant regardless of the size of the data set
O(log n)	Logarithmic	Binary search of a sorted table	Increases by a constant. If n doubles, the time to perform increases by a constant, smaller than n amount
O(<n)	Sublinear	Search using parallel processing	Performs at less than linear and more than logarithmic levels
O(n)	Linear	Finding an item in an unsorted list	Increases in proportion to n. If n doubles, the time to perform doubles
O(n log(n))	n log(n)	Quicksort, Merge Sort	Increases at a multiple of a constant
$O(n^2)$	Quadratic	Bubble sort	Increases in proportion to the product of n*n
$O(c^n)$	Exponential	Travelling salesman problem solved using dynamic programming	Increases based on the exponent n of a constant c
O(n!)	Factorial	Travelling salesman problem solved using brute force	Increases in proportion to the product of all numbers included (e.g., 1*2*3*4...)

Computational complexity



- What about the complexity of a problem?
- Computational Complexity Theory:
  - Classify problems according to resource usage
    - Time
    - Memory
  - ... and relate these classes to each other

Complexity classes

- Depending on the analyzed resource:
  - **TIME**$(f(n))$:
    - Set of all decision problems, solved by a Turing machine in $O(f(n))$ time
    - Examples: P, NP, EXPTIME, NEXPTIME, etc.
  - **SPACE**$(f(n))$:
    - Set of all decision problems, solved by a Turing machine with $O(f(n))$ memory
    - Examples: L, NL, PSPACE, NPSPACE, etc.

Complexity classes

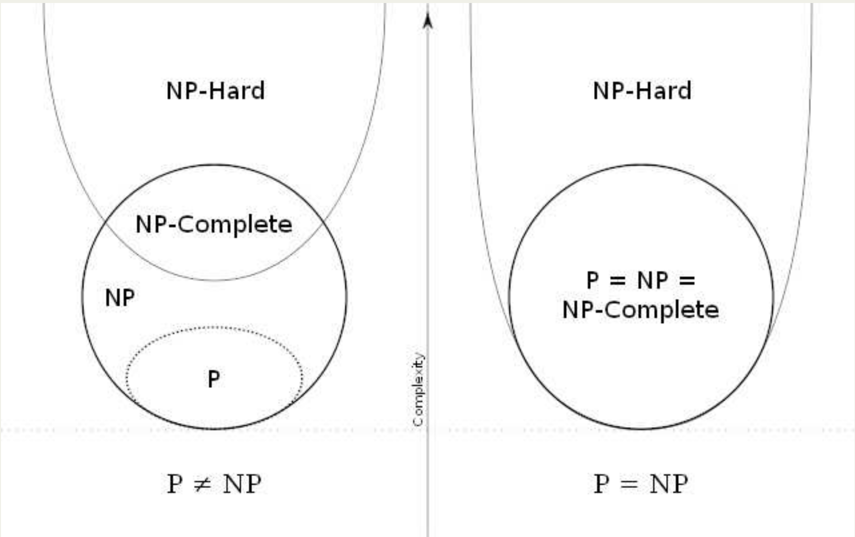
- P: Solved by a deterministic Turing machine in polynomial time
- NP: Solved by a non–deterministic Turing machine in polynomial time
    - Solutions are verifiable by a deterministic Turing machine in polynomial time
- EXPTIME: Solved by a deterministic Turing machine in exponential time
- NEXPTIME: Solved by a non–deterministic Turing machine in exponential time

Complexity classes

- Example of P problem:
  - Given a list of n integers and an integer k, is there an integer in the list greater than k?
    - Solved in $O(n)$
    - Iterate the list and check against k
- Example of NP problem:
  - Given a list of n integers and an integer k, is there a set of integers, within the list, that when summed equal k?
    - Verifiable in polynomial time, but ...
    - Solved in exponential time

Completeness and hardness

Amsterdam University of Applied Sciences



- Almost all of the “hard” NP problems are the same “hard” problem in different guises
- A problem is said to be NP–hard if any NP problem can be efficiently reduced to it
- A problem is said to be NP–complete if it is NP–hard and in NP

Completeness and hardness

- Example of NP–complete problem:
  - Given a Boolean circuit, is there an assignment of its inputs that makes the output `True`?
- Example of NP–hard problem:
  - Given a list of n cities, the distance between each pair of cities and some cost k. Is there a route with length l such that $l < k$ that visits every city exactly once?

Randomness

- What if we allow randomized algorithms?
- Bounded–error Probabilistic Polynomial time (BPP)
  - Solved by a probabilistic Turing machine in polynomial time
  - Bounded error rate of $1/3$
  - Basically:
    - Algorithms are allowed to make a coin toss
    - Guaranteed to run in polynomial time
    - Gives the correct answer at least $2/3$ of the time

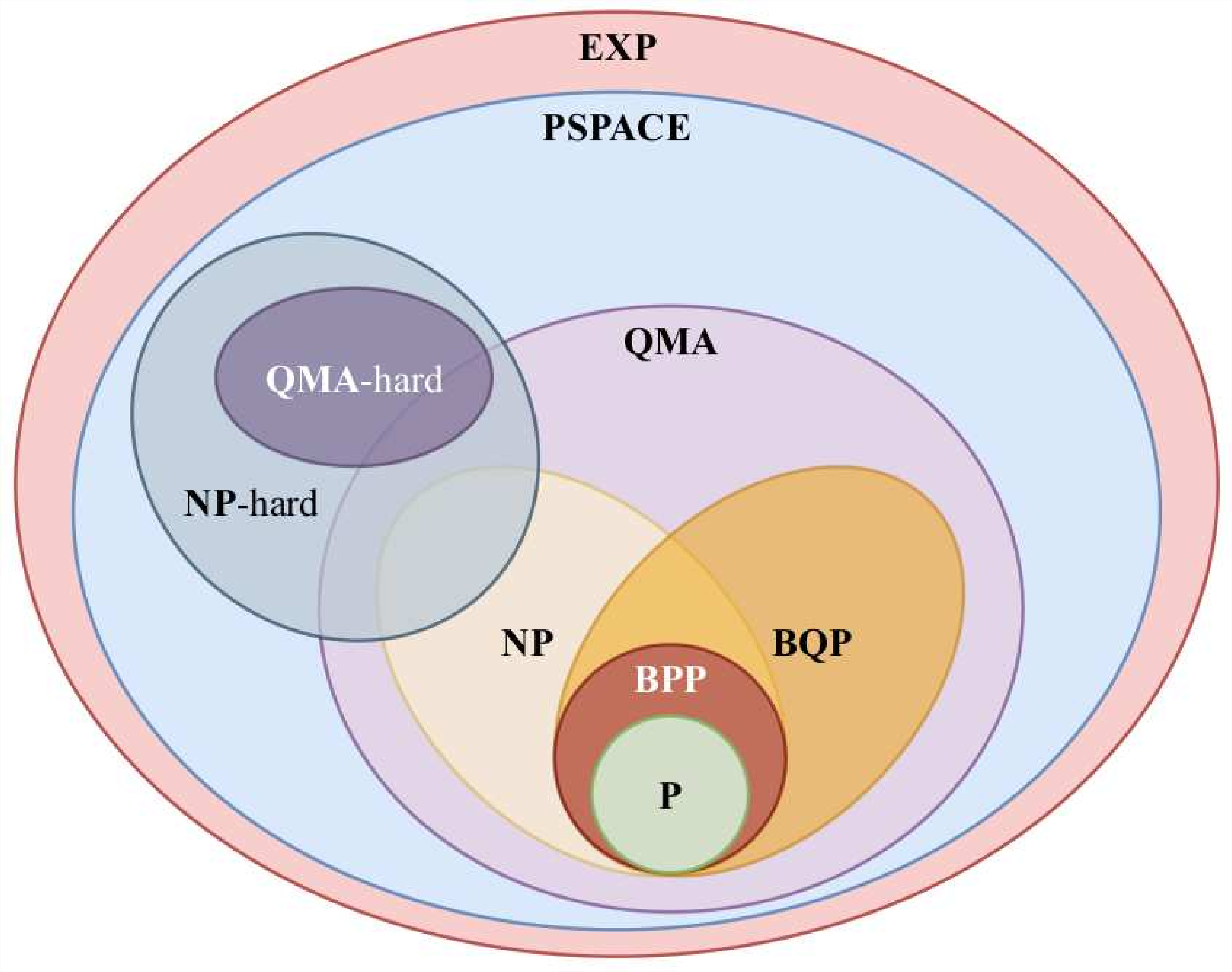
Quantum Complexity Classes

- Bounded–error Quantum Polynomial time (BQP)
  - Set of all decision problems solvable by a quantum computer in polynomial time with an error probability of, at most, $1/3$ for all instances
  - Problem in BQP:
    - There is a quantum algorithm that solves it
    - With high probability ($p > 2/3$)
    - Guaranteed to run in polynomial time
  - NP problems in BQP:
    - Integer factorization

Quantum Complexity Classes

- Quantum Merlin–Arthur (QMA)
  - Set of all decision problems verifiable by a quantum computer in polynomial time with an error probability of, at most, $1/3$ for all instances
  - A prover sends a message to a verifier
  - The message is verified in polynomial time
  - If the answer is $\texttt{YES}$:
    - Verified with $p > 2/3$
    - Using a proof that runs in polynomial time
    - In a quantum computer
  - QMA relates to BQP as NP relates to P
  - Same for QMA–hard and QMA–complete

General Overview



Church–Turing thesis

Amsterdam University of Applied Sciences

- The original thesis states:
  - Any algorithmic process can be simulated using a Universal Turing Machine
- The strong Church–Turing thesis:
  - Any algorithmic process can be simulated *efficiently* using a Universal Turing Machine
- The extended Church–Turing thesis:
  - Any algorithmic process can be simulated efficiently using a *Probabilistic* Turing Machine
- The quantum extended Church–Turing thesis:
  - Any realistic physical computing device can be efficiently simulated by a fault–tolerant quantum computer

Quantum Computing Performance

Amsterdam University of Applied Sciences

- Amount of useful work accomplished by a quantum computer per unit of time
- The QPU is supported by a classical runtime system
- Performance metrics must consider the whole system
- Narrowed down to 3 key factors:
  - Scale: size of the problems to be encoded
  - Quality: size of the quantum circuit to be executed
  - Speed: number of primitive circuits per second

Scale

Amsterdam University of Applied Sciences

- Given by the number of qubits
- Determines the amount of information that can be encoded
  - Size of solvable problems
- Highly dependent on:
  - Available material
  - Fabrication technologies
- Main challenge:
  - Increase scale while maintaining quantum coherence

Quality

- How faithfully a quantum circuit can be implemented
- Quantum Volume:
  - Single–number metric
  - Width of the largest random square circuit that the quantum computer can successfully run
  - Sensitive to:
    - Coherence
    - Gate fidelity
    - Measurement fidelity
    - Connectivity
    - Compilers

Speed

Amsterdam University of Applied Sciences

- How many circuits a QPU can execute per second
- Execution time includes:
  - Updating parameters to the circuit
  - Submitting the job to the QPU
  - Executing on the QPU
  - Sending results back
- QPU speed $\rightarrow$ Circuit Layer Operations Per Second (CLOPS)
- Can be increased by:
  - High–fidelity and fast gates and readout
  - Advanced control electronics
  - Reducing latencies


Questions?