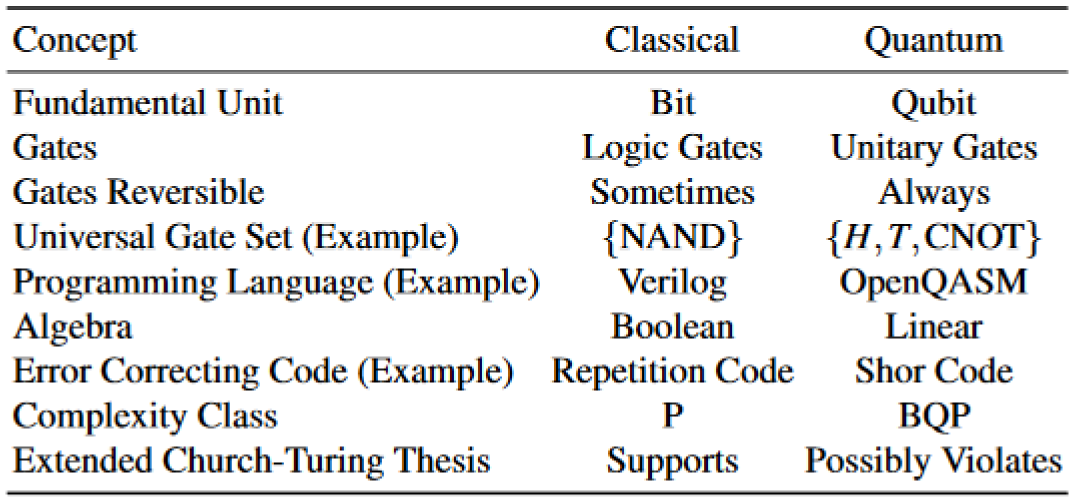
Classical Information and Computation

In this chapter, we learn about the classical concepts shown in the table below:



Concept	Classical	Quantum
Fundamental Unit	Bit	Qubit
Gates	Logic Gates	Unitary Gates
Gates Reversible	Sometimes	Always
Universal Gate Set (Example)	$\{\text{NAND}\}$	$\{H, T, \text{CNOT}\}$
Programming Language (Example)	Verilog	OpenQASM
Algebra	Boolean	Linear
Error Correcting Code (Example)	Repetition Code	Shor Code
Complexity Class	P	BQP
Extended Church-Turing Thesis	Supports	Possibly Violates

THOMAS_WONG_CONCEPTS_TABLE.PNG

Bits

Coins

- With n coins, there are 2^n possible states.

Dice

- With n die, there are 6^n possible states.

Encoding Information

- Systems with two states carry the smallest amount of information possible.

Physical Bits

- Many physical systems have only two state, e.g. coins, light switch, CDs, etc., and others with more than two possible states can be treated as only having two if we ignore the rest of the states, e.g. electronic circuit (consider only two values (0 and 5 Volts) out of many).
- Regardless of the physical system, we conventionally denote the two possible states as 0 and 1, the binary digits.
- Bit is an abbreviation for binary digits, which can be either 0 or 1. Bits are used to represent the states of any physical system with two states. It is the smallest unit of classical information ^2d378f.

Binary

- Binary number, as the name suggest, only consist of 0s and 1s, e.g. 1101 or 101_2 where the subscript of 2 can be included to denote that it is a base-2 number. The leftmost bit is called the the *most significant bit* and the rightmost bit is called the *least significant bit*.

ASCII

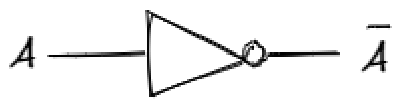
- ASCII uses 7 bits, i.e. $2^7 = 128$ possible states, to encode letters, numbers, symbols, and special commands (like carriage return or newline). 95 out of 128 states are used to encode printable characters, while the remaining 33 states encode non-printable characters.
- Nowadays, we need many more characters to encode (other languages, emojis, symbols).
- The most common modern standard to encode characters is UTF-8 which uses up to 32 bits ($2^{32} = 4,294,967,296$ states) to encode information. The first 128 bit strings in UTF-8 are the ASCII characters.

Logic Gates

- Logic gates are used to manipulate bits. They take one or more bits as input, and, depending on the input, output one or more bits.

Single-Bit Gates

- They take one bit as input and then output one bit.
- There are four possible single-bit gates:
  - The Identity Gate
    - The identity gate does nothing to the bit, i.e. 0 remains 0 and 1 remains 1. The identity gate is sometimes depicted by a triangle in a circuit diagram, but often, we omit the triangle and just draw a longer wire.
  - The NOT Gate:
    - The NOT gate flips a bit from 0 to 1, or 1 to 0. Its circuit diagram is a triangle with a small circle.

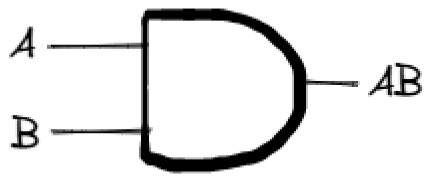


WONGINTRODUCTIONCLASSICALQUANTUM_NOT_GATE.EXCALIDRAW.PNG

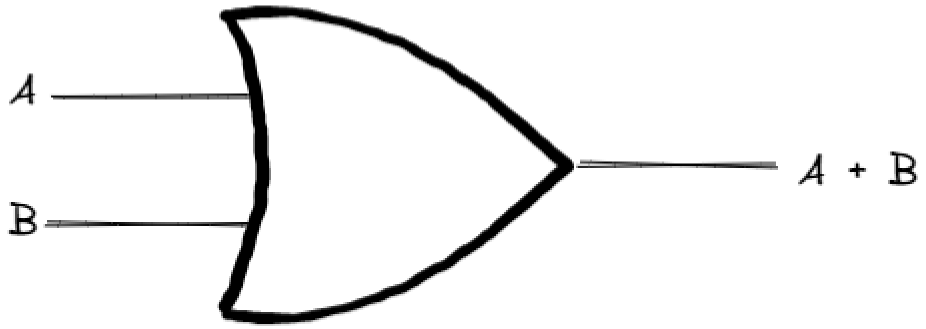
  - The *always 0 gate* always outputs 0 regardless of the input. It is not commonly used.
  - The *always 1 gate* always outputs 1 regardless of the input. It's not commonly used.

Two-Bit Gates

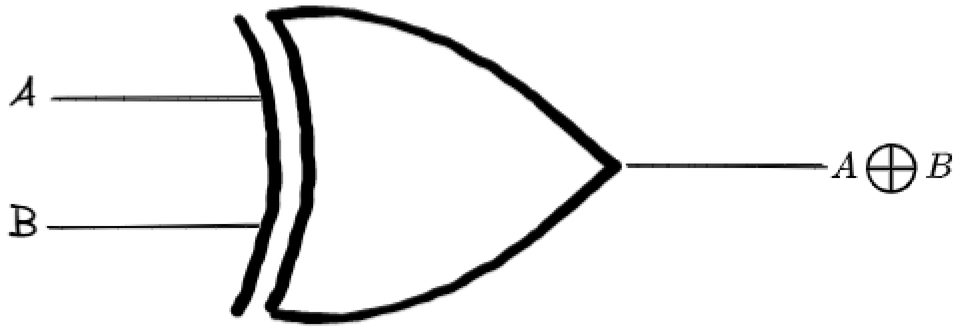
- They take two bits as input, and output one or more bits.
- Five of the most important two-bit gates are:
    - The AND Gate:
        - The AND gate outputs 1 only when both of the input bits are 1. Symbolically, it is represented as AB.

          WONGINTRODUCTIONCLASSICALQUANTUM_AND_GATE.PNG
    - The OR Gate:
        - The OR gate outputs 1 if either or both inputs are 1. Symbolically, it is represented as $A + B$.

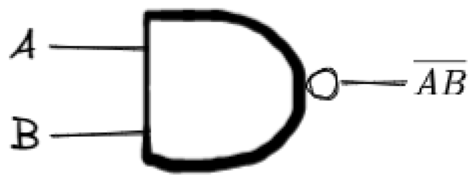
          WONGINTRODUCTIONCLASSICALQUANTUM_OR_GATE.PNG
    - The XOR Gate:
        - The Exclusive OR (XOR) gate outputs 1 when only one of the inputs is 1, but not both.

          WONGINTRODUCTIONCLASSICALQUANTUM_XOR_GATE.PNG

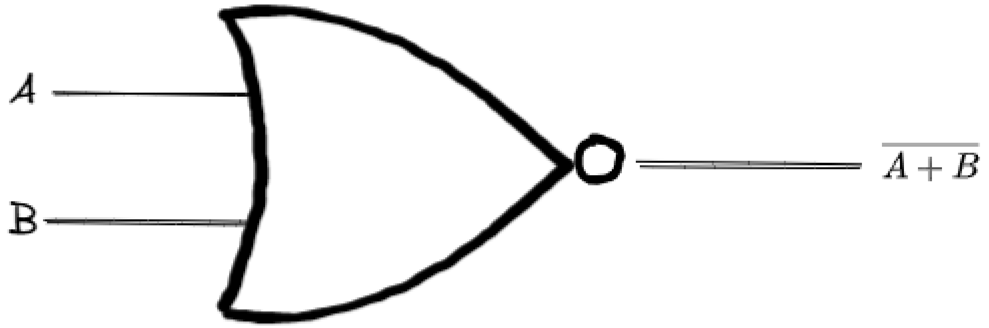
          We write the XOR of A and B as $A \bigoplus B$, where mathematically $\bigoplus$ is addition modulo 2, meaning we take the reminder after dividing by 2.
    - The NAND Gate:

    - The $NAND$ gate is simply NOT of AND. Symbolically, it is represented as $\overline{AB}$. It is a Universal Gate.



wongIntroductionClassicalQuantum_NAND_Gate.png

- The NOR Gate:
    - The NOR gate simply outputs the NOT of the OR Gate. Symbolically, it is represented as $\overline{A+B}$. It is a Universal Gate



wongIntroductionClassicalQuantum_NOR_Gate.png

Logic Gates as Physical Circuits

- Electrical circuits can be used to create logic gates by connecting switches in various ways.
- Nowadays, computers use transistors as the switches, typically made of silicon, where a single computer processor can have tens of billions of transistors.

Universal Gates

- A set of gates that can perform all possible logic operations is called a Universal Gate Set.
- Following are some of the examples of universal gate sets:
  - {NOT, AND, OR}
  - {NOT, AND}
  - {NAND}
  - {NOT, OR}
  - {NOR}

Adders and Verilog

Adding Binary Numbers by Hand

- In general, to add two 4-bit numbers $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$, we need to carry four numbers C_4, C_3, C_2, C_1, and we get a five-bit sum $S_4S_3S_2S_1S_0$:

$$\begin{array}{rr} \text{(carry)} & C_4\,C_3\,C_2\,C_1 \\ & A_3A_2A_1A_0 \\ \text{“+”} & B_3B_2B_1B_0 \\ \hline \text{(sum)} & S_4S_3\,S_2\,S_1\,S_0 \end{array}$$

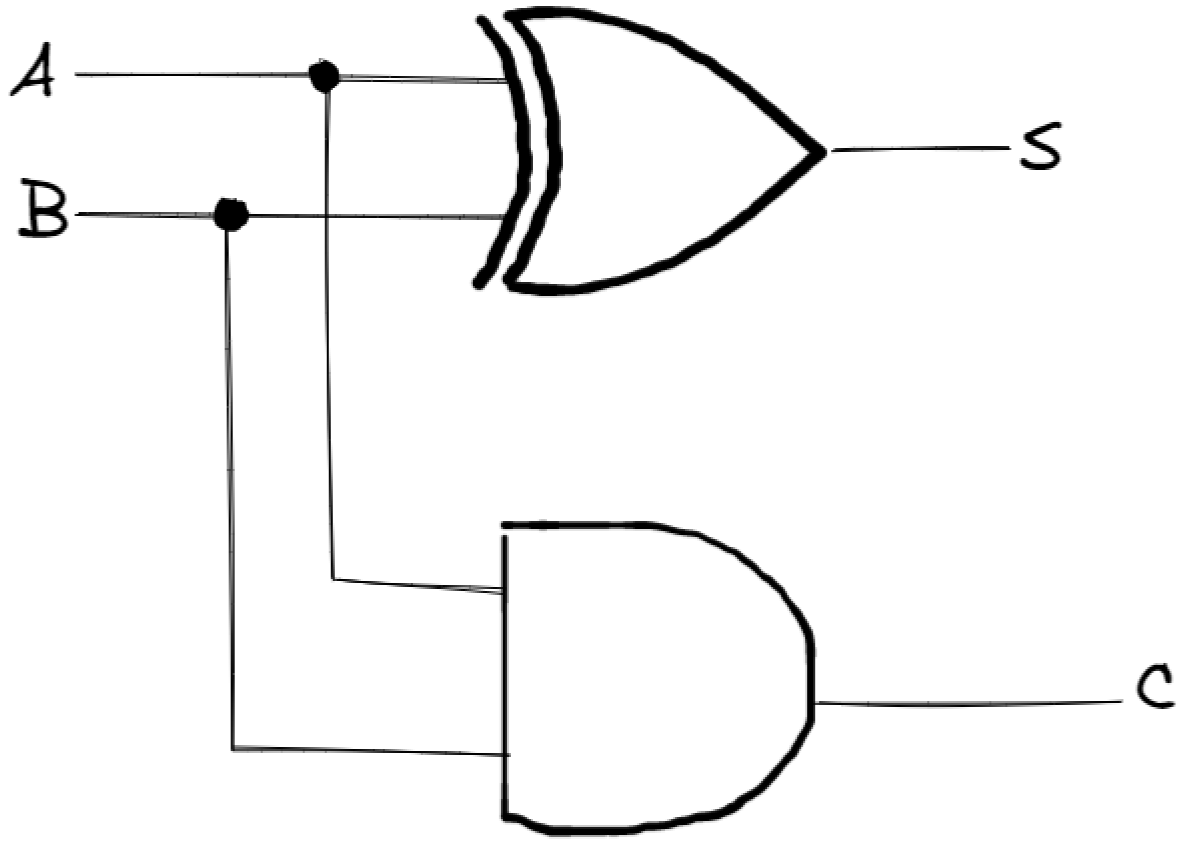
GENERAL_BINARY_ADDITION.PNG

Since the leftmost carry is the leftmost digit of the sum, $S_4 = C_4$.

Half Adder

- A circuit that adds two bits and outputs a carry and a bit of the sum is called *half adder*.
- Sum is the XOR of A and B, and the carry is the AND of A and B, i.e.

$$S = A \oplus B, \quad C = AB.$$



wongIntroductionClassicalQuantum_Half_adder_circuit.png

- Code in Verilog:

```verilog
module halfadd(C,S,A,B);
        input A, B;
        output C, S;

        xor xor1(S,A,B);
        and and1(C,A,B);
endmodule

module main;
        reg A,B;
        wire C,S;

        halfadd half1(C,S,A,B);

        initial begin
                A=0;
                B=1;
                #5; // Wait 5 time units.
                $display("Carry = ",C);
                $display("Sum = ",S);
        end
endmodule
```

Full Adder

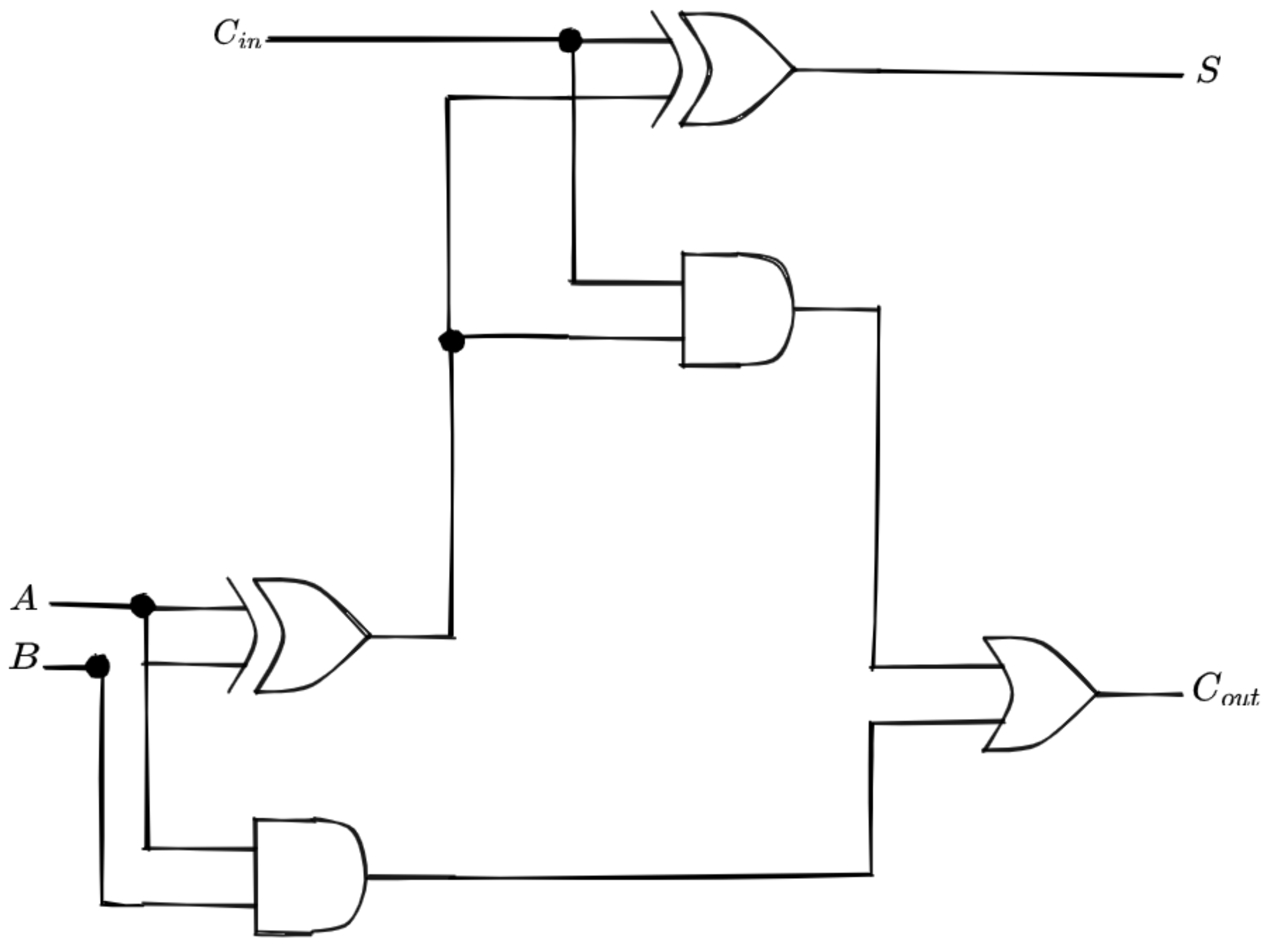
- A circuit that adds three bits (a carry into the sum and the two bits we want to add) and outputs a carry and a bit of sum is called a full adder.
- Carry out is 1 when A and B are both 1, or when $C_{in} = 1$ and $A \oplus B = 1$.:

$$C_{out} = AB + C_{in}(A \oplus B)$$

- The sum S is 1 whenever one of the inputs A, B, or C_{in} are 1, or when all three of them are 1. This is equivalent to the XOR of all three bits:

$$S = A \oplus B \oplus C.$$

- The logic circuit is:



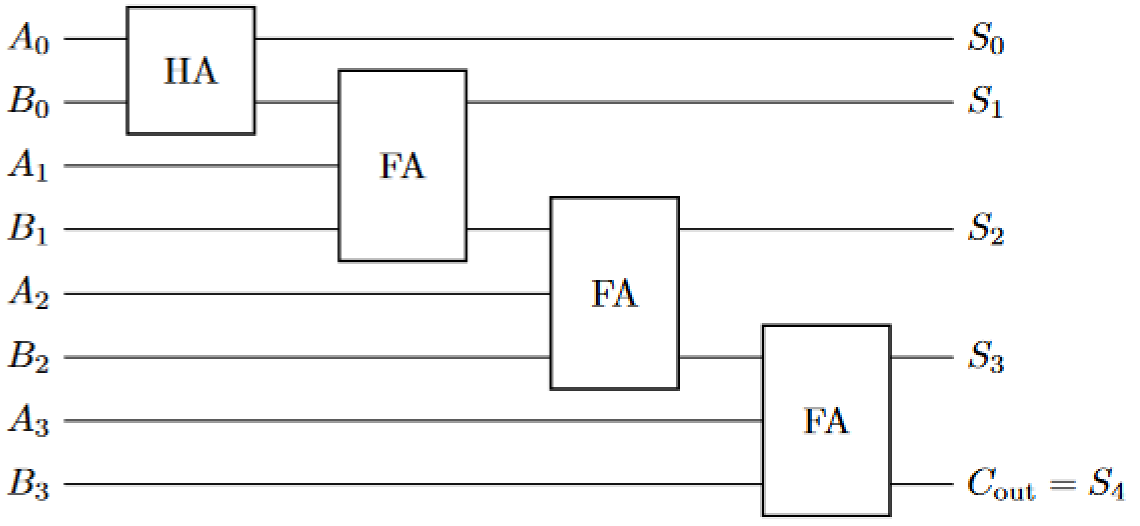
Media/ wongIntroductionClassicalQuantum_2022-01-18 18.35.36.excalidraw.png

- Code in Verilog:

```
module halfadd(C,S,A,B);                                    language-verilog
        input A, B;
        output C, S;

        xor xor1(S,A,B);
        and and1(C,A,B);
endmodule

module fulladd(Cout,S,A,B,Cin);
        input A, B, Cin;
        output Cout, S;
        wire w1, w2, w3;

        halfadd half1(w1,w2,A,B);
        halfadd half2(w3,S,w2,Cin);
        or or1(Cout,w1,w3);
endmodule

module main;
        reg A,B,Cin;
        wire Cout,S;
        fulladd full1(Cout,S,A,B,Cin);

        initial begin
                A=0;
                B=1;
                Cin=1;
                #5; // Wait 5 time units.
                $display("Carry = ",Cout);
                $display("Sum = ",S);
        end
endmodule
```

Ripple-Carry Adder

- It stings together a half adder and several full adders.



[[wongIntroductionClassicalQuantum]]

WONGINTRODUCTIONCLASSICALQUANTUM_RIPPLE_CARRY_ADDER.PNG

This is called a ripple-carry adder, since the carry from one addition ripples to the next addition.

Ripple-Carry with Full Adders

- We can replace the first half adder with a full adder with a carry-in of `0`.

Circuit Complexity

- If the ripple-carry adder consists entirely of full adders, then adding two n-bit numbers requires n full adders. Each full adder uses five logic gates, for a total of $5n$ logic gates.
- If the first adder is half-adder, then it reduces the number of gates to $4n + 2$.

Circuit Simplification and Boolean Algebra

Order of Operations'

- Order of operations matters, just like in regular math.
- AND is done first, then OR.

Association, Commutivity, and Distribution

- AND and OR are associative:
  - $ABC = (AB)C = A(BC)$
  - $A + B + C = (A + B) + C = A + (B + C)$
- AND and OR are commutative:
  - $AB = BA$
  - $A + B = B + A$
- AND and OR are distributive:
  - $A(B + C) = AB + AC$
  - $A + (BC) = (A + B)(A + C)$

Identities Involving Zero and One

- $A0 = 0$
- $A1 = A$
- $A + 0 = A$
- $A + 1 = 1$

Single-Variable Identities

- $A = \overline{\overline{A}}$
- $AA = A$
- $A\overline{A} = 0$
- $A + A = A$
- $A + \overline{A} = 1$

Two-Variable Identities and De Morgan's Laws

- $A + AB = A$
- $A + \overline{A}B = A + B$
- De Morgan's Laws states that the NOT of an AND is the OR of the NOTs, and the NOT of an OR is the AND of the NOTs.
  - $\overline{AB} = \overline{A} + \overline{B}$
  - $\overline{A + B} = \overline{A}\,\overline{B}$

Circuit Simplification

- We can use the above defined identities to simply a circuit.

Reversible Logic Gates

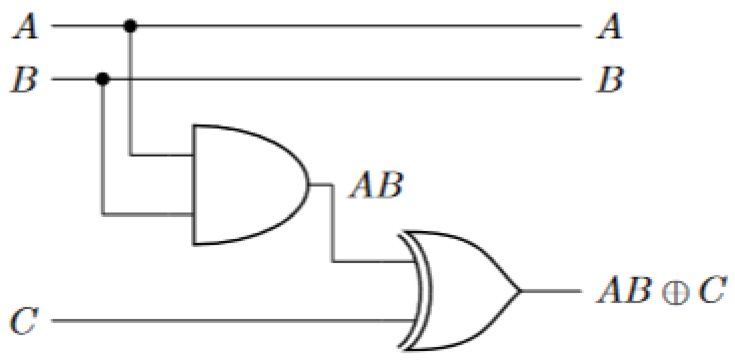
Reversible Gates

- A reversible gate, is a logic gate where, given the output of the gate, we can always determine what the input was. For example, a NOT Gate.
- For a logic gate to be reversible, it must have the same number of input bits and output bits.
- No information is lost in reversible gates as we can always recover the inputs from the outputs.

Irreversible Gates

- An irreversible gate is the opposite of a reversible gate. Given the output of the gate, we generally cannot determine what the input was. For example, the AND Gate.
- A gate will be irreversible if there are fewer possibilities for the outputs than the inputs, and also if there are fewer input bits than output bits.
- Information is lost in irreversible gates, as given an output, we generally do not know what the inputs were.

Toffoli Gate: A Reversible AND Gate

- The Toffoli gate has three inputs A, B, and C. To be reversible, it needs to have three outputs, and they are A, B, and $AB \oplus C$.
- The Toffoli gate can be constructed using an AND gate and an XOR gate. We take the AND of A and B, and we take the XOR of AB with C to get $AB \oplus C$.
- 

  [[WongIntroductionClassicalQuantum]]

  wongIntroductionClassicalQuantum_Toffoli_gate_circuit_digram.png

- Toffoli gate is a Universal Gate.
- The Toffoli gate is also called the Controlled-Controlled-NOT gate or CCNOT gate. Whether the third bit is flipped is controlled by whether the first two bits are 1. That is, if A = B = 1, then the Toffoli gate flips C, and otherwise it does nothing.

Making Irreversible Gates Reversible

- We can use an (some) additional XOR Gate and apply it to one or more additional inputs to make an irreversible gate reversible.

Error Correction

Errors in Physical Devices

- A single event upset is when high energy particles (like cosmic rays, radiation etc.) strike computers, causing bits to flip from 0 to 1 to 0.
- Transmitting data through the internet may also corrupt some bits.

Error Detection

- The simplest way to detect errors is to repeat each bit multiple times so that multiple physical bits encode a single logical bit. This is called the repetition code.
- So if we want to send the letter "Q" in ASCII, which is represented by the seven-bit string 1010001, we would actually send

  $$11001100000011.$$

  If one of the physical bits is flipped, say due to a single event upset or transmission error, then instead of receiving 00 or 11, the recipient would receive 01 or 10, indicating that an error is occurred.
- The binary strings $00, 01, 10$, and 11 are called *codewords*.
- The parity of a bit string is whether the bit string has an even or odd number of 1's.
- The codewords 00 and 11 have an even parity, while the codewords 01 an 10 have an odd parity. This can be computed using the XOR Gate, since $0 \oplus 0 = 1 \oplus 1 = 0$ (even parity) and $1 \oplus 0 = 0 \oplus 1 = 1$ (odd parity).
- If we use the repetition code with two physical bits per logical bit, only one-bit errors can be detected.

Error Correction

- If we use three bits for the repetition code, we can correct the error by taking the majority vote:

  $$\begin{aligned} 001, 010, 100 &\to 000, \\ 110, 101, 011 &\to 111. \end{aligned}$$

  This is an example of an error-correcting code.
- We can use parity checks to correct the error, without every needing to know the actual codeword. This will be helpful in Quantum Computing_⚛ because it won't collapse the qubit's state.

- In case of a logical bit represented with three bits, it can detect and correct the error if a single bit flips. But if two or three bits flip, then it can't. This is called an *uncorrectable error*.
- Let p denote the probability of a single bit flipping. The probability of two specific bits getting flipped is p^2, and since there are three combinations for two of the three bits to be flipped (the two bits could be the first two bits, the last two bits, or the first and last bits), the probability of any two bits getting flipped is $3p^2$. An uncorrectable error also occurs when all three bits gets flipped, and the probability of that occurring is p^3. The coefficient of this is simply 1 because there is only 1 way to choose 3 bits out of 3. So, the probability of an uncorrectable error occurring is

  $$3p^2 + p^3.$$

  As long as this is less than p, which is the probability that an error occurs without error correction, then it is favorable to do error correction

  $$3p^2 + p^3 < p.$$

  Solving this inequality, we see that the 3-bit repetition code is better than no error correction when

  $$0 < p < 0.303.$$

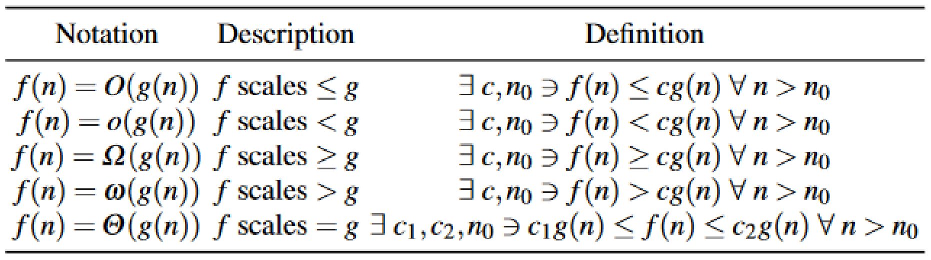
Computational Complexity

Asymptotic Notation

- Big-O is the most commonly used asymptotic notation, and it is useful for specifying the worst-case behavior of an algorithm. Mathematically, $f(n) = O(g(n))$ means there exists constants c and n_0 such that $f(n) \leq cg(n)$ for all values of n greater than n_0.
- Little-o notation is used when we want the inequality to be strictly "less than". So we can write $5n - 3 \neq o(n^2)$. Mathematically, $f(n) = o(g(n))$ means there exists constants c and n_0 such that $f(n) < cg(n)$ for all values of n_0.
- A lower bound on the asymptotic behavior of a function is denoted using big-Omega notation. Mathematically, $f(n) = \Omega(g(n))$ means there exists constants c and n_0 such that $f(n) \leq cg(n)$ for all values of n greater than n_0. We could also write $5n - 3 = \Omega(1)$ to mean that $5n - 3$ is lower bounded by a constant.
- If we want the inequality to be strictly “greater than,” we use a lowercase symbol, or little-omega notation. Mathematically, $f(n) = o(g(n))$ means there exists constants c and n_0 such that $f(n) < cg(n)$ for all values of n greater than n_0.
- To specify that $5n - 3$ scales linearly with n, we use big-Theta notation. We write this as $5n - 3 = \Theta(n)$, and it means that $5n - 3$ is both upper bounded and lower bounded by n, asymptotically. That is, $5n - 3 = O(n)$ and $5n - 3 = \Omega(n)$. Combining the mathematical definitions of each, $f(n) = \Theta(g(n))$ means there

exists constants c_1, c_2, and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all values of n greater than n_0.

- Table 1.2: Summary of asymptotic notations. The mathematical symbol $\exists$ means "there exists," $\ni$ means "such that," and $\forall$ means "for all."

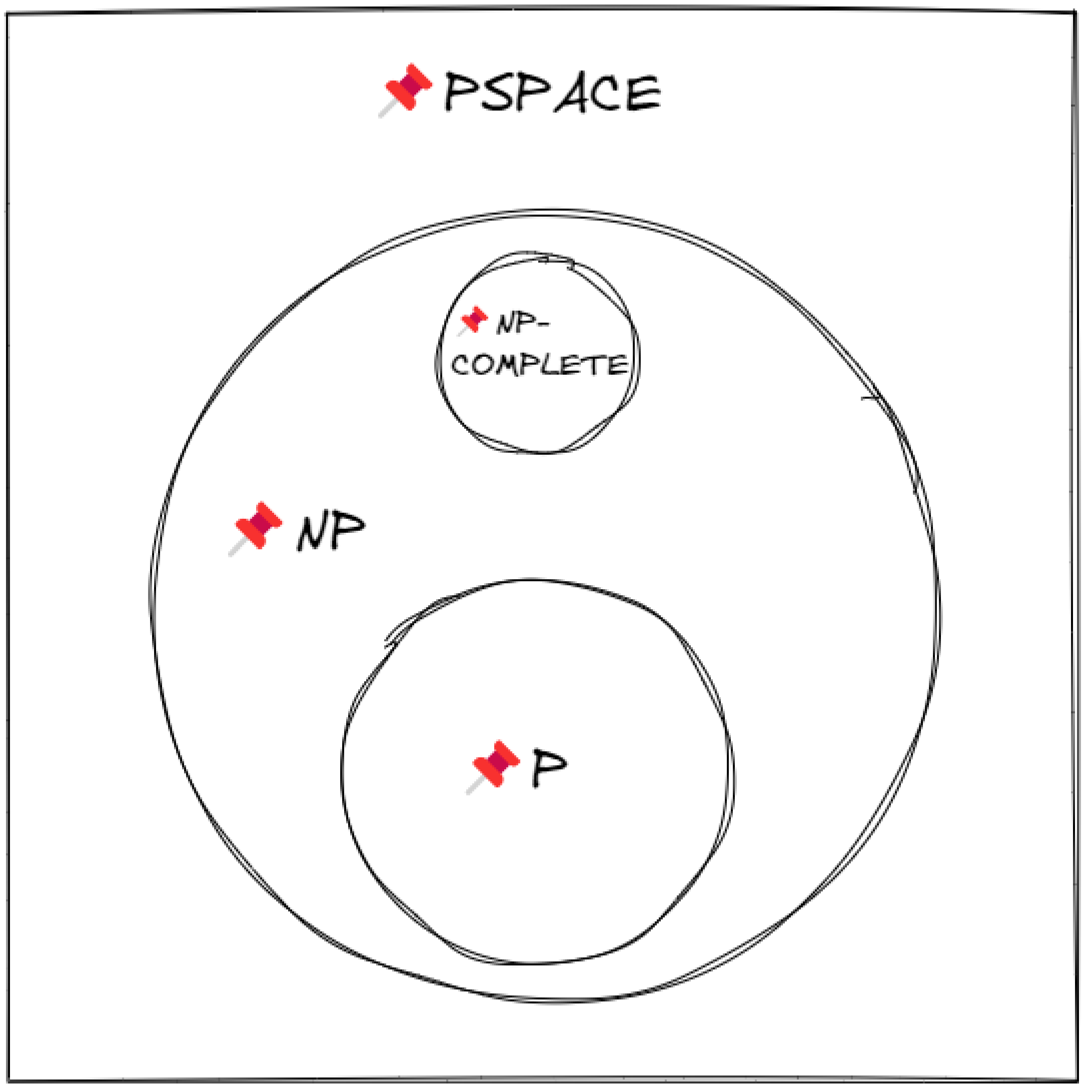


Notation	Description	Definition
$f(n) = O(g(n))$	f scales $\leq g$	$\exists\, c, n_0 \ni f(n) \leq cg(n)\ \forall\, n > n_0$
$f(n) = o(g(n))$	f scales $< g$	$\exists\, c, n_0 \ni f(n) < cg(n)\ \forall\, n > n_0$
$f(n) = \Omega(g(n))$	f scales $\geq g$	$\exists\, c, n_0 \ni f(n) \geq cg(n)\ \forall\, n > n_0$
$f(n) = \omega(g(n))$	f scales $> g$	$\exists\, c, n_0 \ni f(n) > cg(n)\ \forall\, n > n_0$
$f(n) = \Theta(g(n))$	f scales $= g$	$\exists\, c_1, c_2, n_0 \ni c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall\, n > n_0$

📍 [[wongIntroductionClassicalQuantum]]

Complexity Classes

- In computer science, an algorithm is called **efficient** if it takes polynomial time, e.g. n^{1000}. On the other hand, an algorithm is **inefficient** if it takes exponential time, e.g. 2^n.
- Complexity classes are used to classify different problems, depending on how easy or how hard a certain problem is.
- Class P, which stands for Polynomial-Time, contains those problems that can be solved efficiently (in polynomial time) by classical computer.
    - Some problems in Class P are:
      - Matchmaking in the "Stable Marriage problem".
      - Determining if a number is prime.
      - Linear programming.
- Class NP (the N stands for a non-deterministic Turing machine, and the P stands for polynomial) problems are those problems whose solution can be quickly verified by a computer in polynomial time.
    - It includes problems such as:
      - Factoring
      - Graph isomorphism
- Certain problems within NP Class have a special property called *completeness*, and we call these problems NP-COMPLETE. If we can find an efficient solution to any NP-COMPLETE, then we can use it to find an efficient solution to any NP problem.
    - Some NP-COMPLETE problems include:
      - Solving n × n Sudoku puzzles.
      - Traveling salesman problem.
      - Hamiltonian path problem.
      - Bin packing problem.

- A literal million dollar question is Whether P and NP are equal.
- Class PSPACE contains all the problems that can be solved by a computer using a polynomial amount of memory, without any limits on time. Generalizations of many games are in PSPACE.
- It is generally believed that $P \neq NP$ and $NP \neq PSPACE$ (and hence $PSPACE \neq P$), but none of these relations have been proved yet.
- 

WONGINTRODUCTIONCLASSICALQUANTUM_RELATIONSHIPS_BETWEEN_THE_COMPLEXITY_CLASSES.PNG

Turing Machines

Components

- A Turing machine consists of four parts:
  - A tape divided into cells, each with a symbol from some finite alphabet.
  - A head that can read or write to the tape.
  - A register that stores the state of the Turing machine.
  - A list of instructions or program.

- Turing machines can compute everything that a circuit-based computer can compute.

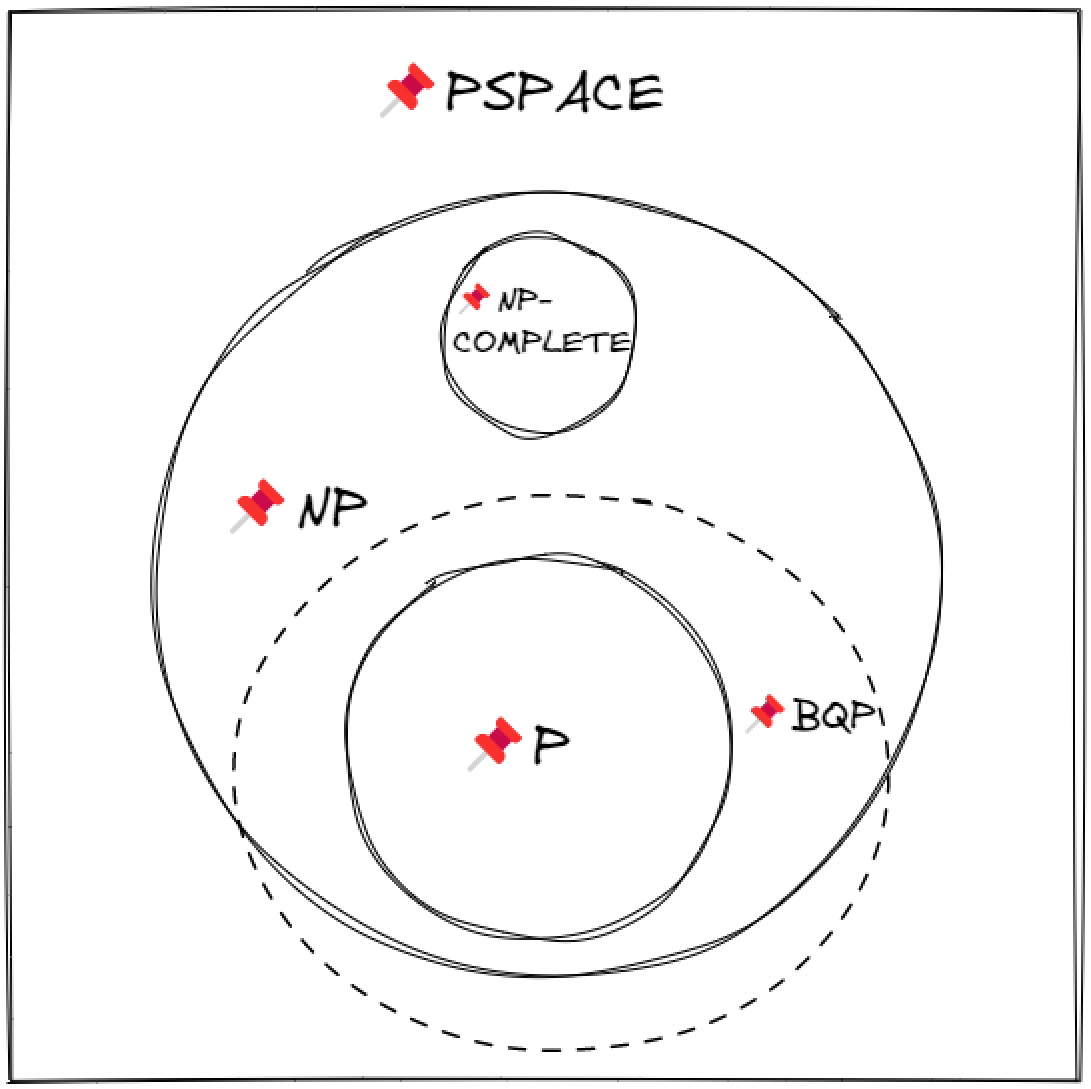
Incrementing Binary Numbers

- Example of a Turing machine being used to add 1 (increment) to a binary number can be seen in the textbook.

Church-Turing Thesis

- The Church-Turing Thesis states that everything that is computable can be computed with a Turing machine, although it could take a long time (e.g., exponential time).
- The Strong Church-Turing Thesis says that any model of computation, be it the circuit model or something else, can be simulated by a probabilistic Turing machine with at most polynomial overhead.
- Quantum computers would not violate the regular Church-Turing Thesis. That is, what is impossible to compute remains impossible. The hope, however, is that quantum computers can violate the Strong Church-Turing Thesis, that they will efficiently solve problems that are inefficient on classical computers.
- While there is no proof of this hope, there is strong evidence for Quantum Computers to violate Strong Church-Turning Thesis.
- The complexity class of problems efficiently solved by a quantum computer is called Class BQP.

-



WONGINTRODUCTIONCLASSICALQUANTUM_RELATIONSHIPS_BW_COMPLEXITY_CLASSES.PNG

Summary

- The smallest unit of classical information is the bit, which can be used to encode information.
- Bits are operated on by logic gates. Together, these gates can be used to perform any computation, and sets of these gates are also universal.
- The mathematics that describes logic gates is called boolean algebra.
- Logic gates can be made reversible.
- In physical systems, errors sometimes occur, but as long as the error rates are sufficiently low, they can be corrected.
- Classical computers can efficiently solve some problems, while other problems take an exponential amount of time.
- It is believed that quantum computers can efficiently solve some of the problems that are hard for classical computers.