Blue ⇒ Write on board

Introduction

Hello, my name is Peri Kay. I am a senior math major and I will be discussing the second half of chapter 1 regarding sections 1.6 through 1.10. I want to highlight what the goal of our book is: “To help an eager reader understand what quantum information science is all about, and for them to realize which facets of it they would like to study in more detail”. So what I will do today is elaborate more on the introduction that Liz gave, so we can have an even better understanding and a better foundation of what we will be learning about throughout the semester.

Here are the topics for each section which I’ll check off as we go:
1.6: Qubits, Gates, and Circuits
1.7: Quantum Decoherence
1.8: Types of Computation
1.9: Computational Complexity
1.10: Outlook

1.6 Qubits, Gates, and Circuits

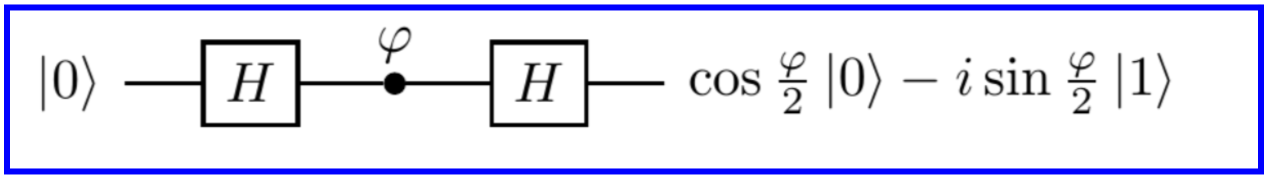
Def (Quantum Entities): microscopic particles or systems that exhibit quantum behavior, governed by the principles of quantum mechanics.

To break that down even further:

- really tiny building blocks–quantum entities – follow quantum mechanics
- can be in multiple states at once– like a magical coin spinning in the air instead of just showing heads or tails – superposition

Many different quantum entities, such as **atoms, trapped ions, molecules,** and **nuclear spins,** exhibit quantum interference by leveraging two pre-selected basis states, denoted as **$|0\rangle$ and $|1\rangle$**. These entities, collectively referred to as **quantum bits** or **=qubits**, serve as the building blocks for implementing basic quantum interference. Note: delving into the specifics of each technology, such as understanding nuclear spin, is not necessary for grasping the fundamentals of quantum theory.

To simplify our 1.6 discussion, we'll focus on representing the path qubits take graphically through **circuit diagrams**.


$|0\rangle$
H
φ
H
$\cos\frac{\varphi}{2}|0\rangle - i\sin\frac{\varphi}{2}|1\rangle$

These diagrams are read from left to right, where the horizontal line symbolizes a **quantum wire** taking a **qubit** through **quantum operations**.

The boxes or circles along the wire represent elementary quantum operations known as quantum **gates**.

So far we kind of understand the structure of the diagram, but now we can look at it more closely and understand what each piece really means. We know the qubit is a tiny quantum entity, which then glides along the quantum wire and passes through this H in a box, which represents the first gate in the diagram. The H gates are called **Hadamard Gates**, which we can think of as resonant interaction. – Resonant interaction in physics: pushing swing example

The Hadamard Gates represent a matrix H s.t.

$$\boxed{H = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}}$$

When a qubit passes through a Hadamard gate, it undergoes a linear transformation that puts it into a superposition of its basis states. In quantum mechanics, a qubit in superposition can exist in a combination of its basis states. For example, if a qubit is in the |0⟩ state (like a coin in heads) and undergoes a transformation, it can be in a superposition of |0⟩ and |1⟩ (like heads and tails) simultaneously.

The other gate that we see is "phi" above a circle, which is called a **phase gate**, which we can think of as a dispersive interaction – changing the timing or alignment of different aspects of a quantum system. For example, a phase gate in quantum computing introduces a dispersive interaction by altering the phase of the qubit state.

The representation of this phase gate is:

$$\boxed{P_\varphi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}}$$

where *phi* is a phase angle. This matrix introduces a phase shift to the quantum state.

Where the horizontal line (the quantum wire) stops is where we can identify the output qubits of the diagram. When we multiply the three matrices H, P of phi, and H again in that left to right order, our result will be an output matrix:

$$\boxed{HP_\varphi H = \begin{bmatrix} \cos\frac{\varphi}{2} & -i\sin\frac{\varphi}{2} \\ -i\sin\frac{\varphi}{2} & \cos\frac{\varphi}{2} \end{bmatrix}}$$

which mathematically describes the entire action of the whole circuit, depending on the basis state of the input qubit.

I want to explain it in a way that might make more sense.

-Cooking recipe example: Quantum recipe represented by a circuit diagram. On the left side of our quantum recipe, we have the input qubits. Think of them like the ingredients you start with.
On the right side, we get the output qubits. This is like the final dish after following the recipe. In the middle, there are three special matrices (mathematical tools) denoted as H and P of phi. These matrices perform certain actions on the qubits as they pass through the quantum recipe.
The combined action of these matrices, denoted as H Pofphi H, tells us how our ingredients, or the input state vectors, transform into the final dish, or the output state vectors.

The first row of our final matrix tells us what the output would look like if we started with 0 as our input state. The second row tells us what the output would look like if we started with 1 as our input state.

So the output matrix describes the action of the whole circuit, telling us that it maps input state vectors to output state vectors as follows:

$$\boxed{\begin{aligned} |0\rangle &\longmapsto \cos\tfrac{\varphi}{2}|0\rangle - i\sin\tfrac{\varphi}{2}|1\rangle, \\ |1\rangle &\longmapsto -i\sin\tfrac{\varphi}{2}|0\rangle + \cos\tfrac{\varphi}{2}|1\rangle. \end{aligned}}$$

1.7 Quantum Decoherence

As you might have noticed from 1.6, quantum theory is closely related to physics and physical phenomena. But believe it or not, we don’t actually need quantum theory to understand all physical phenomena, and instead we can use plain old probability to describe a lot of it.

Quantum theory works well when the object in question is completely isolated from its surroundings.
Example: consider a single atom in deep space, far away from any external influences – as soon as the atom interacts with its surroundings, the quantum behavior may be affected. This is where simpler models like classical probability might be more suitable to describe the system's behavior, as the isolation necessary for using quantum theory disappears.

This is where **quantum decoherence** comes into play.

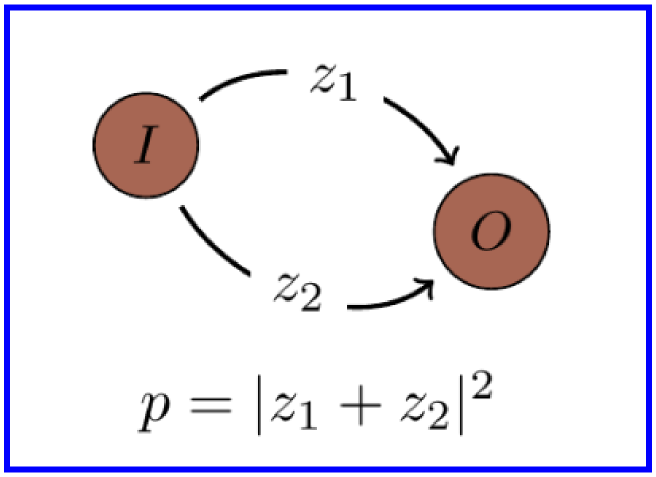
Def: Quantum decoherence is a phenomenon in quantum mechanics where the **coherence and superposition of quantum states break down due to interactions with the external environment.**

– the process by which a quantum system loses its special quantum properties and starts behaving more classically when it interacts with its surroundings.

Our reference gives an example of an isolated system composed of a quantum computer and its environment. Suppose the computer is prepared in some input state I and produces an output O.

These are our two given scenarios:

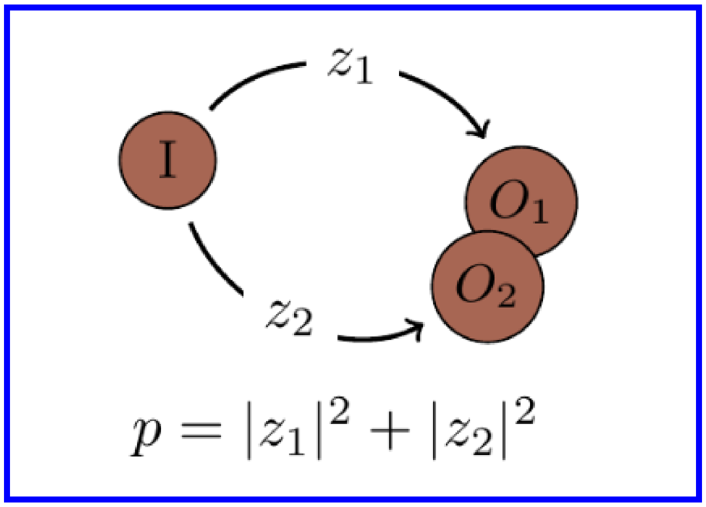
1. **The computer is isolated and quantum info does not affect the environment.**



In this case, the computer is working independently of the environment so the environment doesn't hold any record of how the computer reached its output. So, we take the sum of the amplitudes for each of the two different paths the computer could take to get from its input *I* to its output *O*. So p = the absolute value of z1 + z2 squared. The squared magnitude of the sum is taken to obtain the probability (*p*). This is a common practice in quantum mechanics to ensure that the probability is non-negative. The probability *p* represents the likelihood of observing a specific outcome corresponding to the quantum system.

Our second scenario is

2. **Quantum computation affects the environment.**



Now, the environment remembers and keeps record of how the computer got to its output *O*, creating two final states for the whole system , which we call O_1 and O_2. To figure out the overall likelihood, we just add up the probabilities for each of these two different paths the computation could take.

The sum of the squared magnitudes gives the total probability (*p*). This is a fundamental principle in quantum mechanics known as the **Born Rule** discovered by Max Born. It states that the probability of a system being in a particular state is the sum of the probabilities associated with each possible path or state.

In quantum computation, when the environment is involved, there are two possible outcomes: O_1 and O_2. O_1 is when the computer shows output O, and the environment knows path z_1 was taken; O_2 is when the computer shows output O, and the environment knows path 2 was taken. No interference occurs, so the probabilities for O_1 and O_2 are given by $p_1 = |z_1|^2$ and $p_2 = |z_2|^2$. The overall probability of the computer showing output O is the sum of these probabilities: $p = p_1 + p_2$, which is the decoherence formula when decoherence is not present.

When it is present, it changes to

$$\boxed{p = p_1 + p_2 + 2v\sqrt{p_1 p_2}\cos(\varphi_2 - \varphi_1)}$$

Here, ***v* = visibility** is a parameter ranging from 0 to 1.

***v* = *0* ⇒ total decoherence**

meaning that the environment distinguishes the two paths, resulting in no interference

***v* = *1* ⇒ no decoherence**

meaning that the environment cannot distinguish between paths, leading to full interference.

And any values in between represent varying degrees of partial decoherence.

1.8 Types of Computation

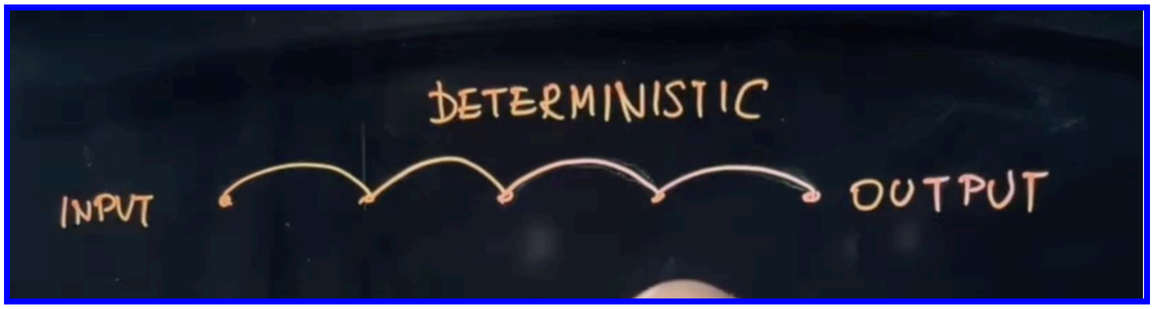
We already know that a qubit (or a quantum bit) has two possible logical states, 0 and 1.
1 qubit ⇒ 2^1 = 2
If we're looking at 2 qubits, then the system of those 2 qubits has four possible states 00, 01, 10, or 11.
2 qubits ⇒ 2^2 = 4
In general, if we have n qubits, the system will have 2^n possible states.
n qubits ⇒ 2^n possible states
When we guide and control how qubits interact with each other, we're essentially carrying out computation.

This chapter discusses three types of computation, used for different problems or tasks; **Deterministic** Computation, **Probabilistic** Computation, and **Quantum** Computation.

Deterministic cooking example: same recipe ⇒ same cookies

Deterministic algorithm: the same input (like the ingredients for cookies), it will always produce the same output (like the batch of cookies). The outcome is entirely predictable, and there's no randomness involved.

In practical terms, think of a calculator as an example of deterministic computation. If you enter the same numbers and perform the same operations, you'll always get the same result. The calculator follows a set of rules or instructions in a predictable way.



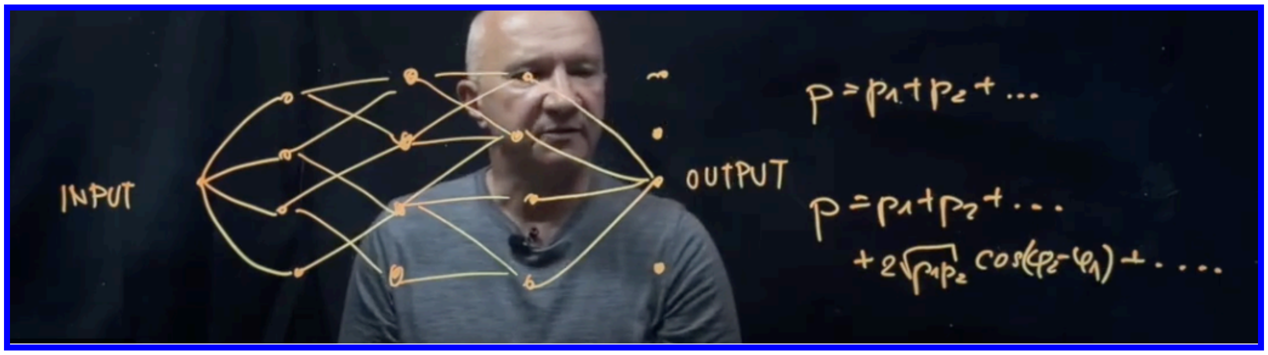
It's also the most common form of computation that computer scientists study.

Deterministic: same input ⇒ same output

The next type of computation is probabalistic. When we say a process or algorithm is probabilistic, it means that the outcome is not fixed or certain. It involves a bit of randomness or uncertainty, like the friend who makes the cookies trying out new things in the kitchen.

Another example: program that makes predictions about the weather. In a probabilistic model, it might not give you a definite "it will rain tomorrow" or "it won't rain." Instead, it might say, "There's a 70% chance of rain."

So, probabilistic computation is about dealing with uncertainties and expressing outcomes in terms of probabilities rather than absolute certainties.


INPUT
OUTPUT
$p = p_1 + p_2 + \dots$
$p = p_1 + p_2 + \dots + 2\sqrt{p_1 p_2}\cos(\varphi_2 - \varphi_1) + \dots$

Probabilistic: outcomes = probabilities

The last type of computation discussed is quantum computation, which involves amplifying the probability that we will see one particular output or outputs that correspond to the correct answer in a problem. So in this case,we add probability amplitudes.

Quantum computation puzzle example: each piece represents a different way a computer can process information.

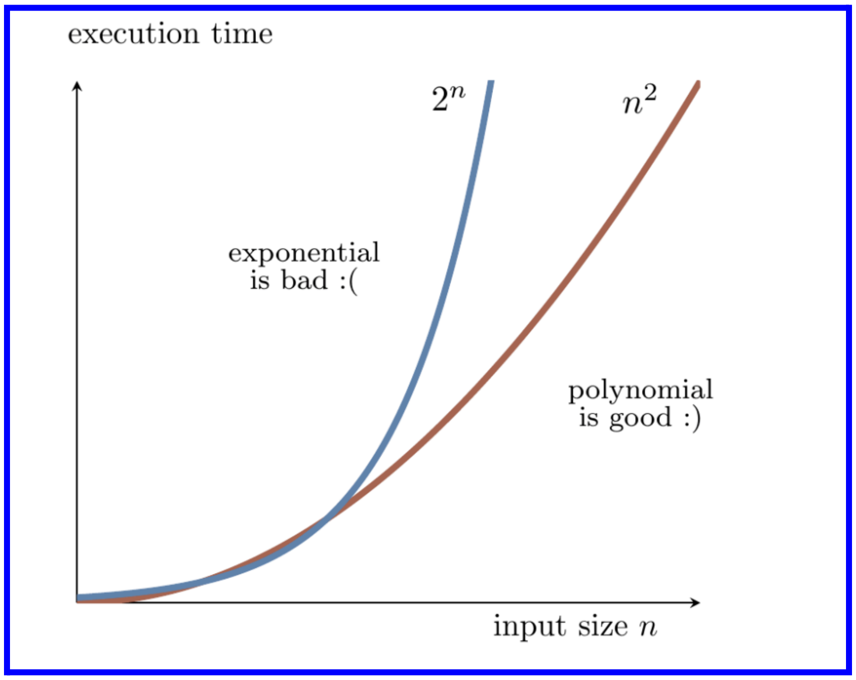
When dealing with quantum computation, the likelihood of reaching a specific final state is determined by adding up the amplitudes along all distinct paths that connect the initial state to that particular outcome. This is represented by the equation here. (point to cos equation)

Understanding different types of computation is important because they each cater to various needs, challenges, and applications. Each type of computation has its strengths and weaknesses, which makes them suitable for different tasks.

1.9 Computational Complexity

Now we will discuss the big question that will hopefully motivate us to stay focused throughout the rest of the semester: **Why is quantum info processing interesting?** In other words, as quoted by our reference, “Is there a compelling reason why we should care about quantum computation?” In general, for physicists they can play with quantum phenomena, build gadgets, experiment with quantum teleportation and quantum cryptography, and at some point, will hopefully be able to build quantum computers.

Computer scientists find quantum computation interesting because it redefines computational complexity classes. To go deeper into this, computer scientists classify problems as either easy or hard, like we all naturally do. But what computer scientists specifically do in this process of classification to quantify problems into the two classes is that they look at an algorithm for a specific problem. We've all heard the word algorithm before, I assume, but in this case, to address a specific problem, computers, whether classical or quantum, adhere to a defined set of instructions known as an **algorithm**. Computer scientists apply an algorithm for whatever problem they're working on and try to solve it by increasing the size of the input.

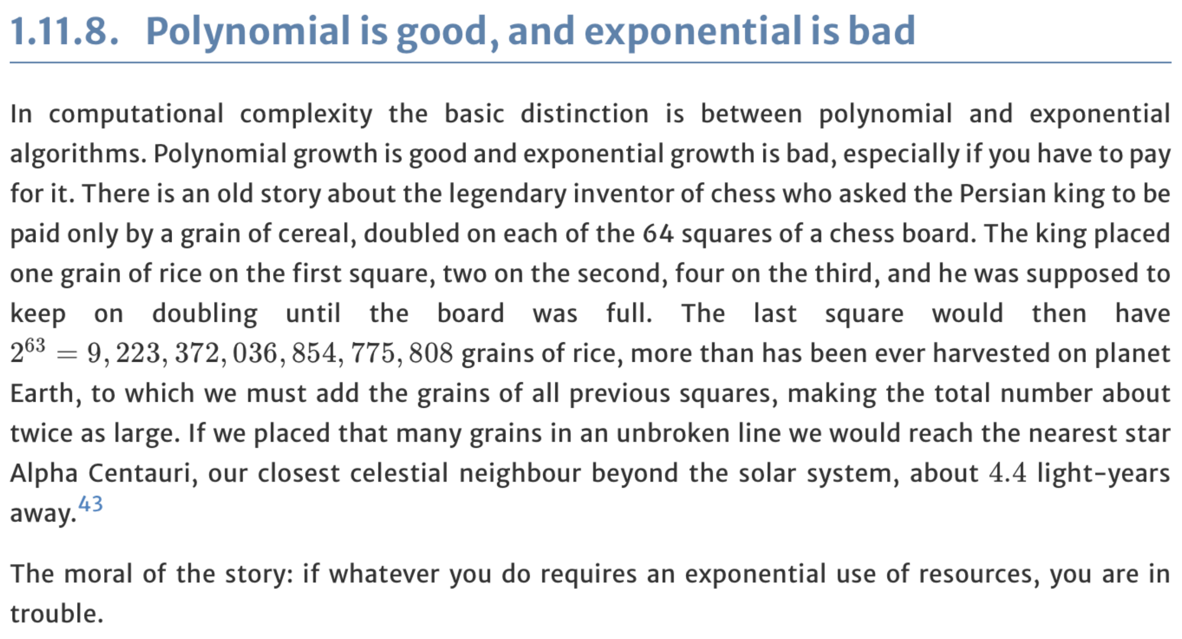


They evaluate the efficiency of an algorithm based on how quickly its execution time increases with larger inputs.

An algorithm is considered efficient if the number of elementary operations required for its execution doesn't grow faster than a polynomial function of the input size. The input size is measured by the total number of binary digits (bits) needed to represent the input. For instance, when using a basic multiplication algorithm taught in elementary school, the time to multiply two n-digit numbers increases at a rate proportional to the square of the number of digits (n^2). On the other hand, the fastest-known method for the reverse operation, factoring an n-digit integer into prime numbers, takes an exponentially growing amount of time, roughly 2^n, which is considered inefficient, as we can see by the graph.

Good ⇒ Execution time doesn’t grow too fast
Bad ⇒ Exponential function of the size of the input …Computer scientists are not happy in this case because they know how to solve the problem, but it's not an efficient way to solve it

A remark from section 1.11 reads, “polynomial is good and exponential is bad”. It reads:



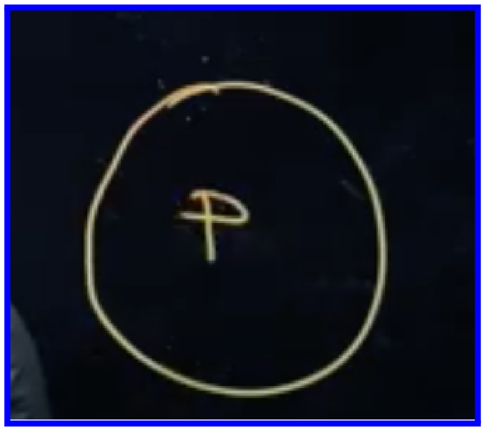
1.11.8. Polynomial is good, and exponential is bad

In computational complexity the basic distinction is between polynomial and exponential algorithms. Polynomial growth is good and exponential growth is bad, especially if you have to pay for it. There is an old story about the legendary inventor of chess who asked the Persian king to be paid only by a grain of cereal, doubled on each of the 64 squares of a chess board. The king placed one grain of rice on the first square, two on the second, four on the third, and he was supposed to keep on doubling until the board was full. The last square would then have $2^{63} = 9,223,372,036,854,775,808$ grains of rice, more than has been ever harvested on planet Earth, to which we must add the grains of all previous squares, making the total number about twice as large. If we placed that many grains in an unbroken line we would reach the nearest star Alpha Centauri, our closest celestial neighbour beyond the solar system, about 4.4 light-years away.[43]

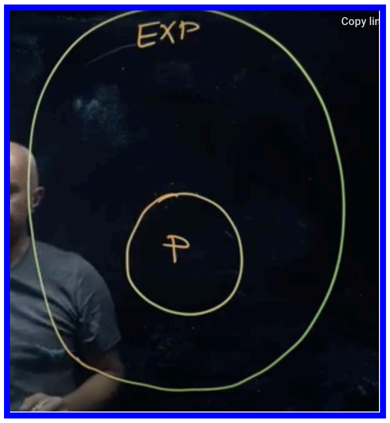
The moral of the story: if whatever you do requires an exponential use of resources, you are in trouble.

To further separate easy and difficult, computer scientists draw complexity classes like this

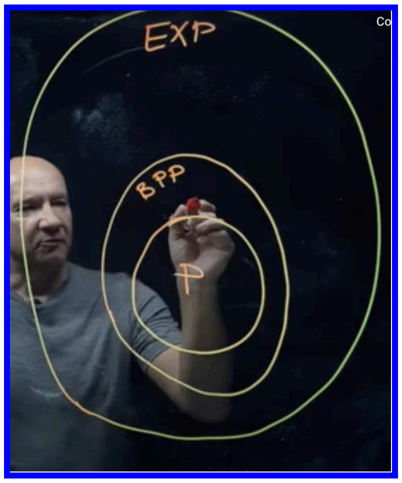
All problems that we can solve on a deterministic classical computer in polynomial time is a class called “p”



That class is a subset of a big class of problems that require exponential time to solve them.



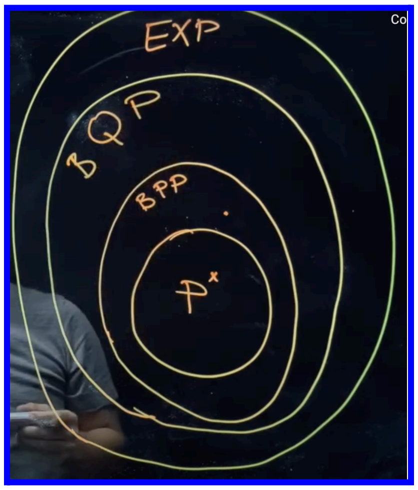
And we talked about probabilistic computation in the previous section, so what if we want to use that? When we use randomness as an extra resource for computation, computer scientists believe that maybe the class of problems that can be solved in polynomial time on a probabilistic computer is slightly larger,



Which is not proved yet but for the sake of this lesson, we can ignore that and this class is called BPP, which stands for **bounded-error probabilistic polynomial** time.

BPP includes P because deterministic computation (P) is a type of probabilistic computation where we don't use randomness. In probabilistic (or randomized) computation, if we run the same task many times, we might not get the exact answer every time. However, it's considered useful as long as there's a high chance of getting the correct answer.

What we're more specifically interested in is a larger class of problems that quantum computers can solve in polynomial time, which we call BQP



Which stands for **bounded-error quantum polynomial** time.

Because a quantum computer can effortlessly create random bits and imitate a probabilistic classical computer, it's clear that the category BQP includes the class BPP (which, in turn, includes the class P). There are problems in BQP such as factoring large integers or decomposing a large number into prime factors, which we saw was difficult for any classical computations, but this is why computer scientists find quantum computing so interesting is because in BQP, with a quantum computer, computer scientists can do it. There are problems that are known to be in BQP that are not in BPP or P. Building a faster and faster computer doesn’t make a difficult problem easier because whatever problem progresses exponentially in terms of difficulty will still progress exponentially no matter how fast the computer is.

In order to make a difficult problem easy, the only way is to be creative and to think of new forms of algorithms or ways to use external tools to solve the problem. That’s what computer science is all about. So in conclusion of this chapter, quantum information is very technological and looks very technological but actually requires a lot of creativity.

1.10 Outlook

1960s, when scientists began exploring how computers work, they were worried that the strange rules of quantum mechanics, some of which I just went over, might limit how accurately computers could represent abstract ideas like logical operations. BUT quantum mechanics didn't cause as many problems as they feared, and it even helped solve some issues from classical physics.

Quantum mechanics has a lot of interesting possibilities that can lead to new technologies and knowledge. Our reference only touches the basics of quantum computing and focuses on important concepts, not getting into too many technical details. Although we're not sure when or how we'll have fully working quantum computers, they're definitely a serious option for the future.

In our current society, quantum theory in computing is already more important than the older classical ideas. If you're interested in understanding the core principles of physics, computing, or logic, you should do more research in the field beyond the text and consider the new perspectives brought by quantum theory. Quantum theory can be used to explain many questions we have about the universe.