Extracted from:

Quantum Computing

Program Next-Gen Computers for Hard, Real-World Applications

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Quantum Computing

Program Next-Gen Computers for Hard, Real-World Applications

Nihal Mehta, Ph.D.

edited by Brian MacDonald

Quantum Computing

Program Next-Gen Computers for Hard, Real-World Applications

Nihal Mehta, Ph.D.

The Pragmatic Bookshelf
Raleigh, North Carolina

Pragmatic Bookshelf

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Brian MacDonald
Copy Editor: L. Sakhi MacMillan
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-720-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—August 2020

Your First Quantum Program

The way to program quantum computers differs from what you would do for traditional computers. Although the quantum programming syntax borrows from today's computer languages, such as JavaScript, C#, or Python, the underlying concepts are widely divergent. Objects such as JavaScript `Promises` help us write organized and efficient code. But, under the hood, the machine language instructions are pretty much the same as those, say, for FORTRAN or BASIC. Quantum programming is different. The underlying hardware has fundamentally changed. So, to write code that works on these machines, we need to rethink the way we write computer programs.

Rather than introduce quantum computing on a "Hello World" program or some other contrived example, we'll jump right in and run a practical computational task on a quantum computer. This exercise will immediately show you that this technology is real.

A Scheduling Problem

We'll use a quantum computer to come up with a schedule for Las Vegas shows. This scheduling task is a simpler version of a problem discussed in Knuth's *The Art of Computer Programming [Knu11]*, Section 7.1.1. In our version, we deal with contemporary performers but have retained the timeless Las Vegas hotels.

Our job is to schedule three talk show hosts for a comedy festival over two days at three hotels. We have to slot the shows based on the hotels that each artist can perform at:

- Jimmy Kimmel performs only at Aladdin and Bellagio.
- Bill Maher performs only at Bellagio and Caesars.
- Trevor Noah performs only at Caesars and Aladdin.

For these kinds of problems, whether you write a program for a conventional computer or a quantum one, you must first express them with logical constraints. Only then can you write a program to hunt for a valid solution.

For a vast range of applications, including these types of scheduling problems, this form boils down to searching for a feasible solution to a system of Boolean logic expressions. This way of modeling applications is referred to as the *Boolean Satisfiability (SAT)*[16] problem in computer science. Following Knuth, we go through this analysis in the next section.

16. https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

Joe asks:

What Makes a Problem Hard?

Since the quantum equivalents of the binary bits and gates work differently in the quantum world, not every application is suitable for quantum computers. For example, quantum computers aren't used to verify whether an email address is correctly filled out in an HTML form or for transactional applications, such as putting information into a database or streaming video to a browser. Rather, quantum computers are ideally suited where a computer has to crunch through a large number of possible solutions of computational tasks.

Such a problem will allow you to see quantum effects in play and will drive home the point that quantum phenomena can fruitfully be put to use in common computational applications and not merely reserved for esoteric and highly idealized cases. The scheduling problem has several candidate solutions, as well as features that allow you to exercise many quantum principles, yet is simple enough to understand the solution space without getting overwhelmed with details.

Modeling Boolean Logic Expressions

Setting up the Boolean logic expressions for computational problems is more art than science. For many problems, there are several acceptable ways to model them. Knuth's book is a great resource to get an overall flavor on this way of modeling.

Writing a System of Boolean Logic Expressions

To write a quantum program for this scheduling problem, we model it using a system of Boolean logic expressions. Start by defining three Boolean variables k,m,n as follows:

- k means that Jimmy Kimmel does Bellagio on Day 1 and Aladdin on Day 2; k-bar or $\overline{k}$ means that Kimmel does them in opposite order: Aladdin on Day 1 and Bellagio on Day 2.
- m means that Bill Maher does Bellagio on Day 1 and Caesars on Day 2; m-bar or $\overline{m}$ means that Maher does them in opposite order: Caesars on Day 1 and Bellagio on Day 2.
- n means that Trevor Noah does Aladdin on Day 1 and Caesars on Day 2; n-bar or $\overline{n}$ means that Noah does them in opposite order: Caesars on Day 1 and Aladdin on Day 2.

Next, we set up Boolean logic expressions that ensure that no two artists are slated at the same hotel on the same day—the *conflict constraints*. For example, on Day 1 at Aladdin, Kimmel and Noah cannot perform at the same time. This restriction results in the following logic expression:

- Aladdin on Day 1: $\neg(\overline{k} \wedge n) = k \vee \overline{n}$

The symbol $\wedge$ stands for the *Logical AND*, $\vee$ for *Logical OR*, and $\neg$ is *Logical NOT*.

Thus, the left-hand side of the first relation, for example, states that Kimmel performing at Aladdin on Day 1 (k), *and* Noah at Aladdin on Day 1 (n), expressed as ($k \wedge n$), cannot be true at the same time: $\neg(k \wedge n)$. That is, both can't perform at Aladdin on the same day. The right-hand side is its simplification via De Morgan's rule (see Boolean Logic Expressions, on page ?).

Similarly, we can define the logic expressions for the other slots:

- Aladdin on Day 2: $\neg(k \wedge \overline{n}) = \overline{k} \vee n$
- Bellagio on Day 1: $\neg(k \wedge m) = \overline{k} \vee \overline{m}$
- Bellagio on Day 2: $\neg(\overline{k} \wedge \overline{m}) = k \vee m$
- Caesars on Day 1: $\neg(\overline{m} \wedge \overline{n}) = m \vee n$
- Caesars on Day 2: $\neg(m \wedge n) = \overline{m} \vee \overline{n}$

For a valid schedule, all these logic expressions must be true. That is,

$$(k \vee \overline{n}) \wedge (\overline{k} \vee n) \wedge (\overline{k} \vee \overline{m}) \wedge (k \vee m) \wedge (m \vee n) \wedge (\overline{m} \vee \overline{n}) = 1$$

In general, unless there are specialized techniques for a specific class of Boolean logic expressions, the only way to find a valid set of Boolean variables is go through each combination one at a time.

Single Letter Variables

Although you'll prefer using more suggestive variable names in your programs, I'll use single-letter variables so that they're easier to relate back to the Boolean logic expressions.

In subsequent chapters, you'll learn how quantum mechanical principles come together to get a feasible schedule for the performers at the Vegas hotels by only scanning a fraction of the combinations.

In the next section, you'll get a rapid-fire overview of how this search is done on a quantum computer. We'll explain in detail in subsequent chapters.

Running on a Quantum Computer

I selected the IBM Q Experience,[17] a cloud service to run quantum computers, for all the code examples in this book because it requires minimal setup, making it ideal to learn this new technology—there are no installation or connectivity battles to overcome before you can use a quantum computer. All you need is a web browser and an internet connection. You can also use the material in this book with Microsoft's or Amazon's quantum computer—it's much the same as what you'll see here. We're just using the IBM computer because we have to pick one.

To write and run your programs on the IBM Q Experience, which we'll also refer to as the IBM Quantum Computer, you'll need to first set up an account. You can sign up in one of two ways:

- Get a free IBMid account from https://quantum-computing.ibm.com/login.
- Use your Google, Twitter, LinkedIn, or GitHub account.

Later, in Chapter 11, Where to Go from Here, on page ?, you'll see how to invoke the IBM Quantum Computer from within your own applications using an API Token.

17. https://www.research.ibm.com/ibm-q/technology/experience

The examples for the IBM Q Experience are written in the *Open Quantum Assembly Language*.[18,19,20] These programs use a .qasm file extension. We'll learn this language along the way.

You'll Write Most Programs Using Drag-and-Drop

Although .qasm looks like assembly language, you'll learn quantum computing concepts by dragging and dropping quantum devices on a graphical interface. This visual form of your program translates into .qasm code. You can also upload the source code in this book, which then produces the visual representation.

Once you understand how to design algorithms with quantum effects, programming them in a conventional language becomes routine. In Programming with Qiskit, on page ?, you'll learn to program quantum concepts using conventional languages such as Python. With these languages, though, you don't get the immediate interactivity that drag-and-drop brings when learning about quantum phenomena.

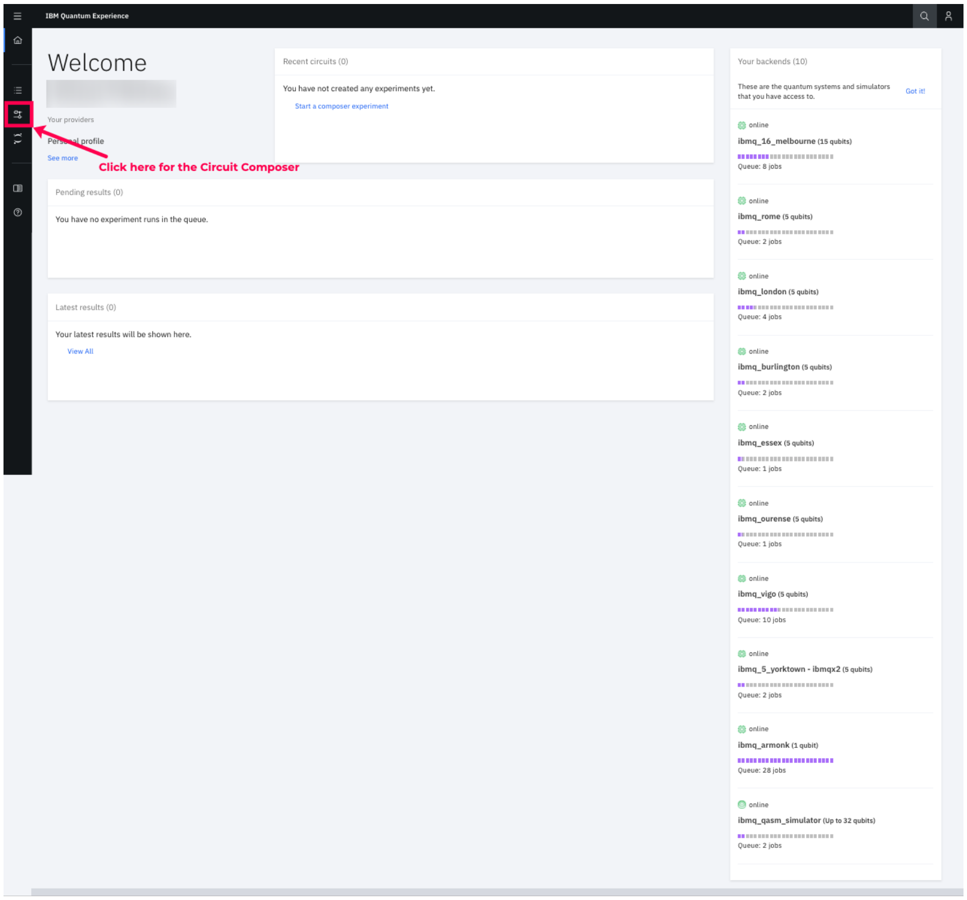
In addition to IBM's Qiskit, you can use the principles and techniques in this book to program quantum computers in languages from other vendors, such as Amazon's Braket, on page ?, Google's Cirq, on page ?, and Microsoft's Q#, on page ?.

Although we'll work with the IBM Quantum Computer, our examples are universal and easily modified to run on other quantum circuit computers.

Now that we've settled on a quantum computer and signed into our account, let's take a walk-through of the interface before we run the quantum program to find a feasible schedule for the talk show hosts.

On quantum circuit computers, a quantum program is also called a *circuit*, which is a visual representation of the sequence of quantum instructions. Thus, to start writing a quantum program, log in to the IBM Quantum Computer and click the Circuit Composer icon on the left margin, as shown in the figure on page 6.

18. https://arxiv.org/abs/1707.03429
19. https://github.com/Qiskit/openqasm
20. https://github.com/Qiskit/openqasm/blob/master/spec/qasm2.rst



IBM Quantum Experience

Welcome

[redacted]

Your providers

Personal profile

See more

Click here for the Circuit Composer

Recent circuits (0)

You have not created any experiments yet.

Start a composer experiment

Pending results (0)

You have no experiment runs in the queue.

Latest results (0)

Your latest results will be shown here.

View All

Your backends (10)

These are the quantum systems and simulators that you have access to. Got it!

online
ibmq_16_melbourne (15 qubits)
Queue: 8 jobs

online
ibmq_rome (5 qubits)
Queue: 2 jobs

online
ibmq_london (5 qubits)
Queue: 4 jobs

online
ibmq_burlington (5 qubits)
Queue: 2 jobs

online
ibmq_essex (5 qubits)
Queue: 1 jobs

online
ibmq_ourense (5 qubits)
Queue: 1 jobs

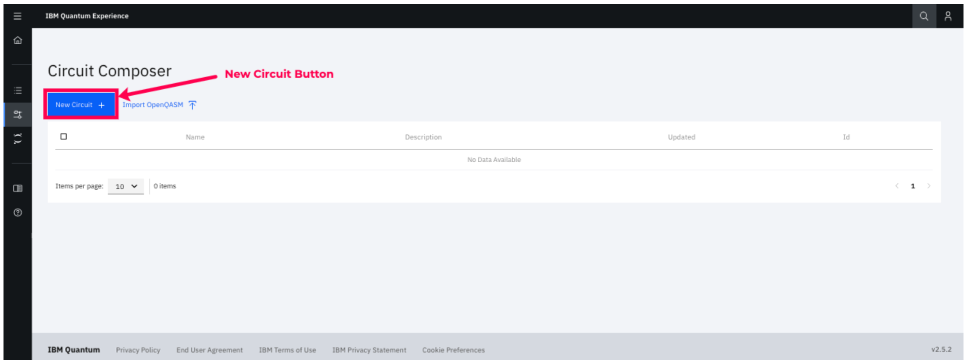
online
ibmq_vigo (5 qubits)
Queue: 10 jobs

online
ibmq_5_yorktown - ibmqx2 (5 qubits)
Queue: 2 jobs

online
ibmq_armonk (1 qubit)
Queue: 28 jobs

online
ibmq_qasm_simulator (Up to 32 qubits)
Queue: 2 jobs

This takes you to a page that lists your quantum programs. To create a new program, click the New Circuit button, shown in the following figure:



IBM Quantum Experience

Circuit Composer

New Circuit Button

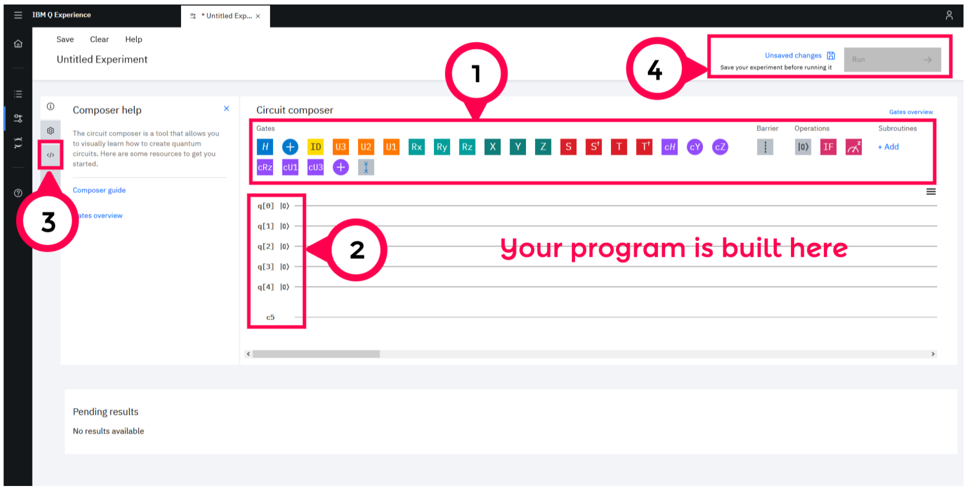
New Circuit +   Import OpenQASM

☐	Name	Description	Updated	Id
		No Data Available		

Items per page: 10   0 items   1

IBM Quantum   Privacy Policy   End User Agreement   IBM Terms of Use   IBM Privacy Statement   Cookie Preferences   v2.5.2

This action will open up the following interface:



This interface is called the Composer. Its key elements are as follows:

1. In a quantum circuit computer, the quantum instructions are called *gates*. To write a program, you drag and drop these gates on the main area labeled Your Program Is Built Here.
2. The variables of your computational problem are stored in arrays that are the subatomic particles in the computer. The "quantum" stuff happens here.
3. To see the code corresponding to the visual drag-and-dropping of the gates, click this tab to open the Circuit editor.
4. To Save and Run, click the respective buttons in this area.

Over the subsequent chapters, you'll learn to write quantum programs from scratch. For now, though, you'll run a complete program and see for yourself how it solves the *Hotel Scheduling Problem*. You can get the program from the book's official page.[21]

Simple Scheduling Problem

The quantum program you'll run is actually a simpler version of the Hotel Scheduling Problem formulated in Writing a System of Boolean Logic Expressions, on page 2, as its solution is easier to verify.

21. https://pragprog.com/book/nmquantum/quantum-computing

Simple Scheduling Problem

In the simpler version, we'll work with just one hotel, Bellagio, and the two hosts, Kimmel and Maher. The solution to this problem is then that each of them performs on a different day—we can't have both perform on the same day or have a day when neither performs.

The first few lines of the program are listed here:

Bellagio_Hotel_Scheduling_Problem_Final.qasm

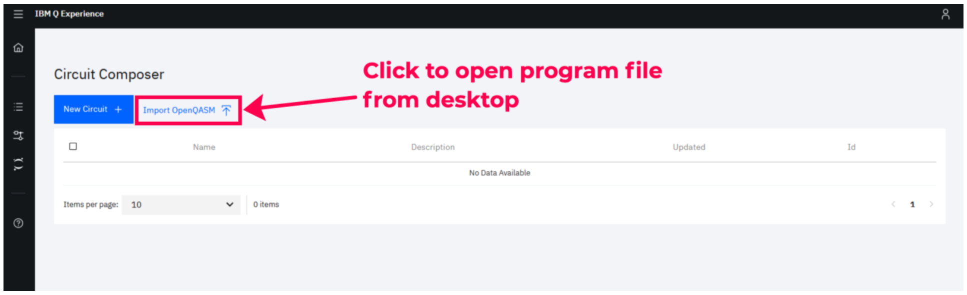
```
OPENQASM 2.0;
include "qelib1.inc";

// Initialize Quantum and Classical Registers
qreg q[7];
creg c[2];

// Generate All Combinations
h q[0];
h q[1];
h q[2];
h q[3];

//// ITERATION 1
// Constraints (to tag optimal solution)
x q[4];
x q[5];
cx q[0],q[1];
cx q[2],q[3];
x q[0];
x q[1];
x q[3];
x q[1];
x q[3];
ccx q[1],q[3],q[4];
x q[1];
x q[2];
x q[3];
ccx q[0],q[2],q[5];
x q[0];
x q[2];
ccx q[4],q[5],q[6];
x q[0];
x q[2];
z q[6];
```

The first line specifies the version of the Open Quantum Assembly Language (OpenQASM) for our programs. On the second line, we pull in an `include` file containing the specifications for commonly used functions in a quantum program. These two lines form the header in every program we write for IBM's Quantum Computer. We will cover the remaining lines in subsequent chapters.

Quantum Programming Language Versus Conventional Languages

Although the statements in a quantum program resemble those of digital computers, they are instructions to invoke quantum phenomena to solve computationally intensive problems. They're not a direct replacement for those used in conventional computer languages. They're based on a fundamentally new template with its own set of concepts and schemes, which you'll learn about in subsequent chapters.

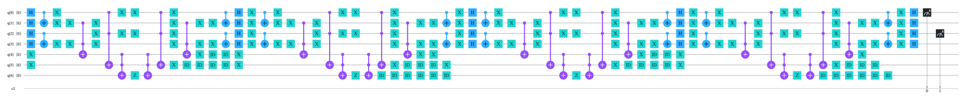
To import this program, click the Circuit Composer tab on the bar on the left edge of the browser to go to the page that lists your programs. On the top of this list, click the Import QASM File button and select the program you just created from your desktop:



☐	Name	Description	Updated	Id
		No Data Available		

Items per page: 10 | 0 items | 1

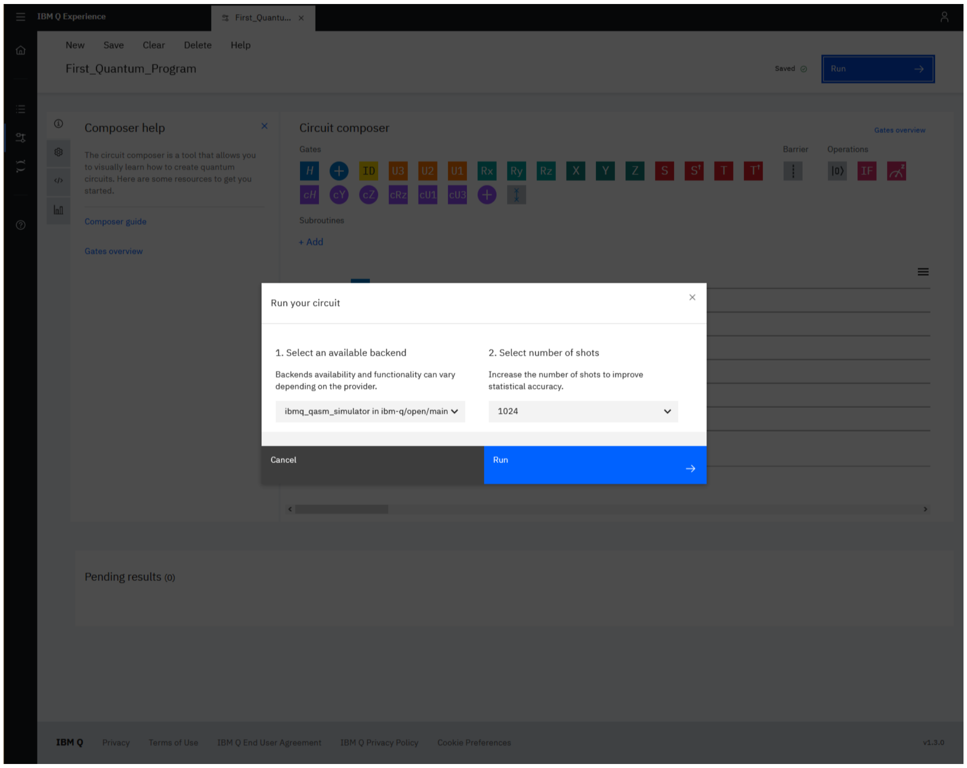
Once the program has been loaded, you'll see it in the list. Click it to open the program in the Composer. You'll see the following visual representation of the code you just imported:



To see the code listing, click the Circuit Editor tab on the left edge of the Composer.

We'll explain how the code works in subsequent chapters. For now, though, Save the code and click the Run button on the top right.

On the next screen, you have a choice of quantum computers, the *backend*, available along with a simulator in the drop down on the left:



The circuit composer is a tool that allows you to visually learn how to create quantum circuits. Here are some resources to get you started.

Run your circuit

1. Select an available backend

Backends availability and functionality can vary depending on the provider.

ibmq_qasm_simulator in ibm-q/open/main

2. Select number of shots

Increase the number of shots to improve statistical accuracy.

1024

Cancel

Run

If you choose to run the program on a real quantum computer, your program will be placed in queue. Depending on the workload of the specific quantum computer selected, my programs have taken anywhere from fifteen minutes up to a day to get back the results. Of course, the actual runtime of your code on the computer is very quick. Using the simulator, on the other hand, will give you results almost immediately.

Use the default in the second drop-down, Number of Shots. We will explain later when to change this value.

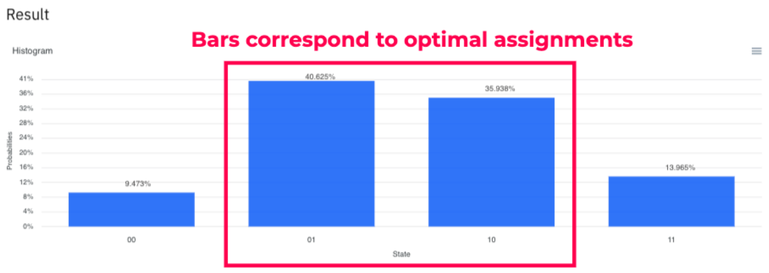
Quantum Computer Simulators

Quantum computers rely on marshalling the quantum mechanical nature of subatomic particles to get them to perform computations for super-hard applications. These quantum features include *superposition* and *entanglement*, concepts we'll study later, and need special-purpose hardware from the ground up. These phenomena can't be reproduced on classical machines. Consequently, although simulators mimic these characteristics on digital computers, they can't tap into their inherent potential power. Hence, simulators are good for only small applications.

Examining the Output

The link to the Results of your program, whether it's still waiting to be executed or is complete, appear below the Composer. Scroll the page to get to it and click the link when it's available to view the results. (Every time you execute a program, a new Results link is created.)

On the Results page, scroll down to the Result section, where you'll see a graphical output similar to the following figure:



We'll go over interpreting the output of the program in the next chapter. For now, I just want to point out a few salient points about how the program reports the results of an execution.

Quantum computers work with the quantum equivalents of binary bits. So, while they go about computing a solution differently from classical machines, we'll still deal with the 0 and 1 binary concepts in our programs.

In the Hotel Scheduling program, we defined the quantum equivalents of the classical bits representing the various options for the talk show hosts to perform. Thus, the program returns values for these options as strings of 0s and 1s, which you'll see at the bottom of each bar.

For reasons that'll become clear in subsequent chapters, the taller middle two bars correspond to feasible schedules. The binary strings at the base of these bars are `01` and `10`. These are the ones that correctly solve the Boolean logic expressions for the simpler version. The others are discarded. I realize you may not understand the how and why of selecting this particular string of `0`s and `1`s gives the optimal solution nor how it relates to the problem variables. But it'll soon start making sense. Although we'll go over this example in detail in Searching for an Optimal Schedule, on page ?, the point I want to make now is that quantum programs return valid results for real world problems—they're not just laboratory experiments.

This string of `0`s and `1`s corresponds to the following solutions:

$k = 0 \mapsto$ Jimmy Kimmel performs at Alladin on Day 1 and at Bellagio on Day 2
$m = 1 \mapsto$ Bill Maher performs at Bellagio on Day 1 and at Caesars on Day 2

and

$k = 1 \mapsto$ Jimmy Kimmel performs at Bellagio on Day 1 and at Aladdin on Day 2
$m = 0 \mapsto$ Bill Maher performs at Caesars on Day 1 and at Bellagio on Day 2

In other words, limiting our attention to just the assignments for Bellagio, the solutions represent two valid schedules:

	Solution 1	**Solution 2**
Day 1	Maher	Kimmel
Day 2	Kimmel	Maher

This, then, is a typical way quantum programs are used in practice in the industry: create a system of binary logic expressions, model the task as a Boolean Satisfiability problem, and then set up those expressions in a program. The more variables you use, the more complex the expressions—in this case, as the number of talk show hosts, days, and hotels increase, the number of possible solutions increases exponentially. In other words, the problem grows astronomical quickly. Traditional methods quickly reach their limit with such problems, which means that developers settle for imperfect solutions, which leaves money on the table. As you'll soon understand, a quantum computer solves such problems in a heartbeat.

Even though this problem can be solved using a custom-built technique, as explained in *Knuth [Knu11]*, a quantum program is still more efficient in the sense that it's both easier to set up and it returns a solution quickly.

Quantum Computers Are Still in Their Infancy

The number of bits that a quantum computer can handle, their stability, and the speed of computations are improving continually. So, although quantum computers haven't quite achieved the *quantum advantage*, the point at which quantum computers are faster than classical ones, the gap between them is closing—dare I say—daily.

Bottom Line

Quantum computing is real—it's no longer just theory and wishful thinking, nor do you need pots of money to use one. It has literally come to a theater near you.

Unlike other kinds of computer technology, quantum computing works on a totally different set of principles and needs specialized hardware. As a result, classical code won't work on them. You have to rewrite your programs from the ground up.

Although I glossed over several aspects of running the Hotel Scheduling Problem on a quantum computer, I wanted to drive home the point that quantum programs are:

- Not isolated statements that do nothing useful other than demonstrate esoteric concepts; they can do useful computations for standard applications.
- Like conventional programs in the sense that you use standard interfaces to program and get your code to execute on them—you're not working in lab coats in sterile environments on particle accelerators.
- Programmable using standard statements. While these types of computers are based on quantum mechanics, the programming statements are mundane—no arcane constructs—and use familiar programming instructions.

In the next chapter, we'll learn about quantum mechanics principles in a way that emphasizes its connection with computer science and makes the quantum aspects more concrete.