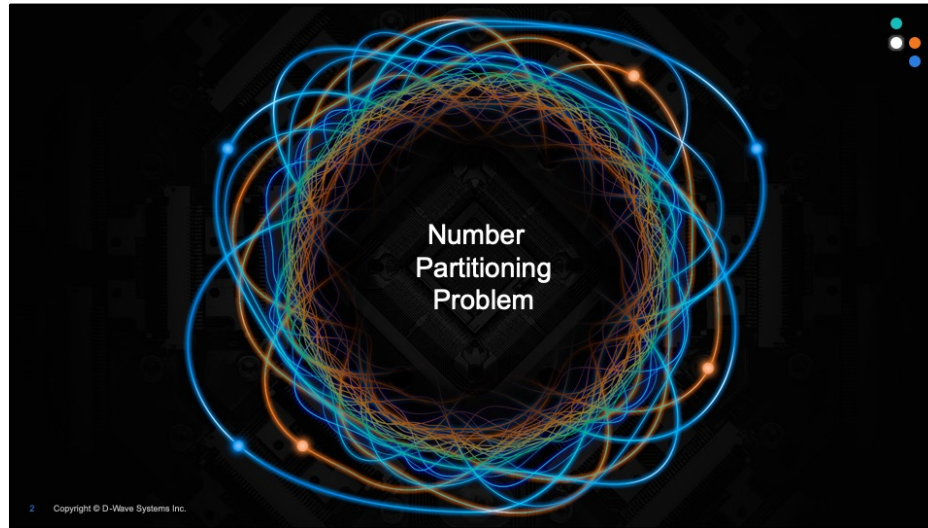


In this module, we'll look at how to write Ocean programs for QUBOs that have an objective function and no constraints.

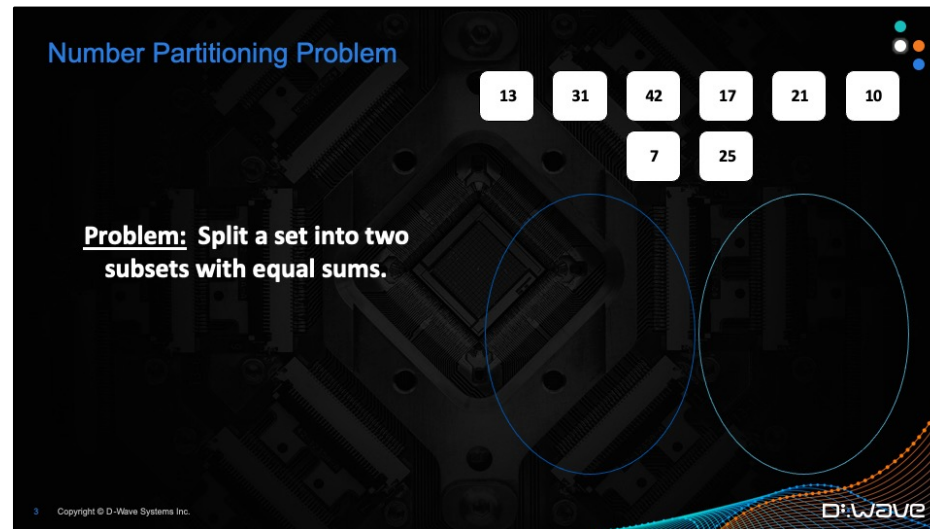


We'll go through this module by exploring a problem called the number partitioning problem.

During this lecture we'll go over the QUBO formulation for this problem and your homework assignment will be to write the Ocean program to run it on the QPU.

Number Partitioning Problem

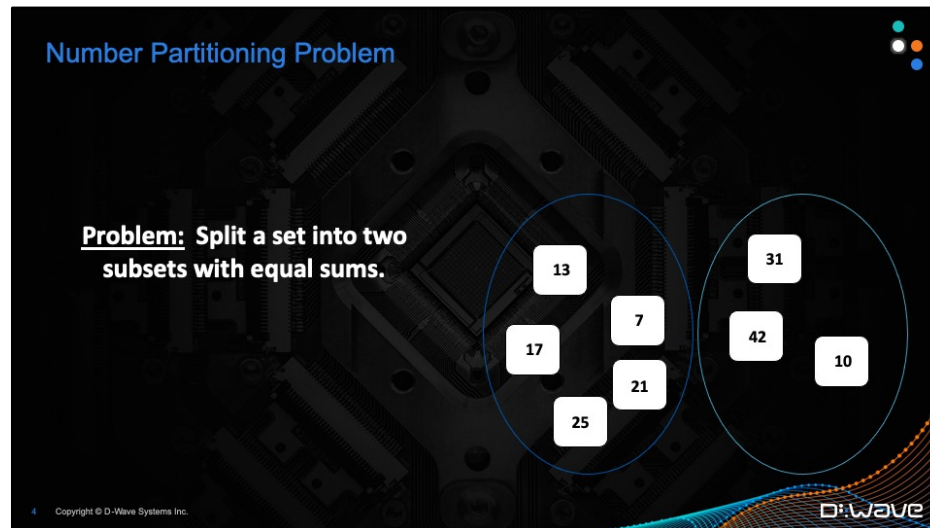
Problem: Split a set into two subsets with equal sums.



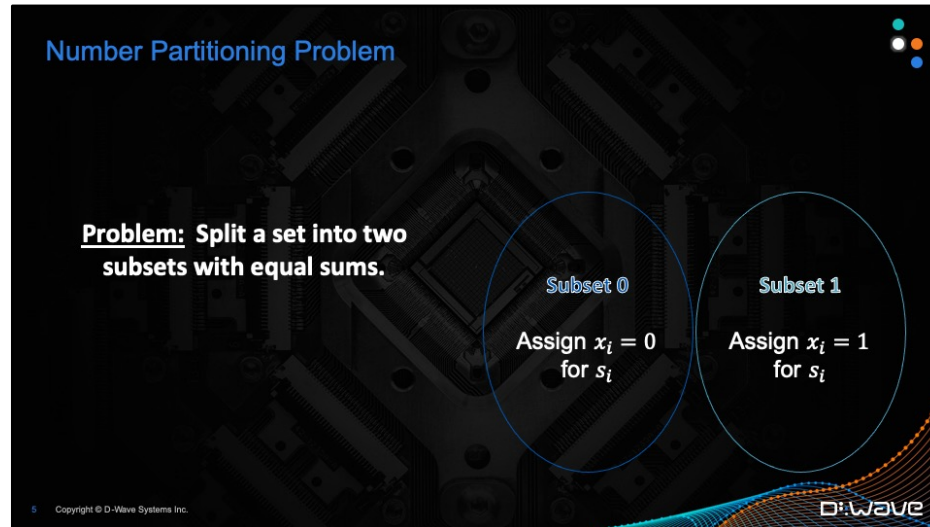
The diagram illustrates the Number Partitioning Problem. At the top, a set of numbers is displayed in white boxes: 13, 31, 42, 17, 21, 10, 7, and 25. Below the numbers, two large, empty circles represent the two subsets into which the numbers must be split. The background features a dark, abstract pattern of lines and dots, with a small D-Wave logo in the bottom right corner.

Copyright © D-Wave Systems Inc.

The number partitioning problem asks you to take a set of numbers and split it into two subsets with equal sums.



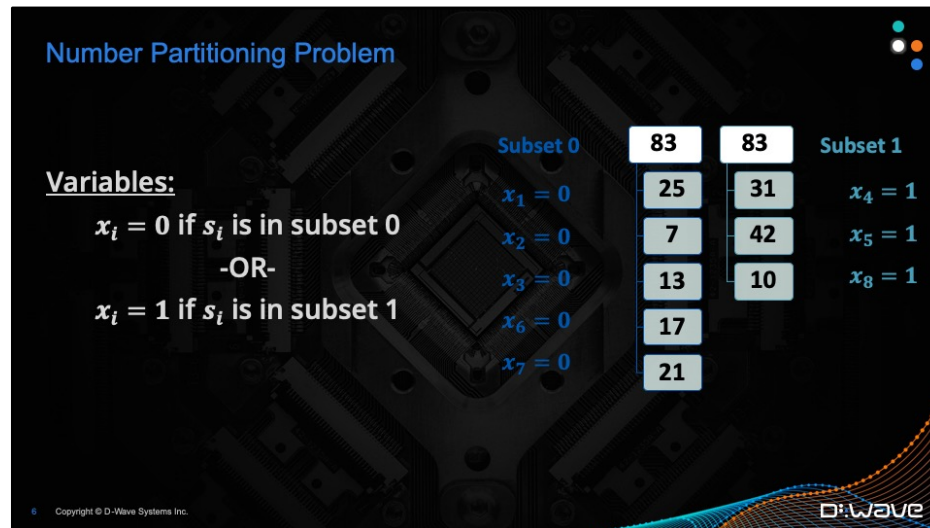
For example, if we have this set of 8 numbers we might split it as shown here. With this split each subset has a sum of 83.



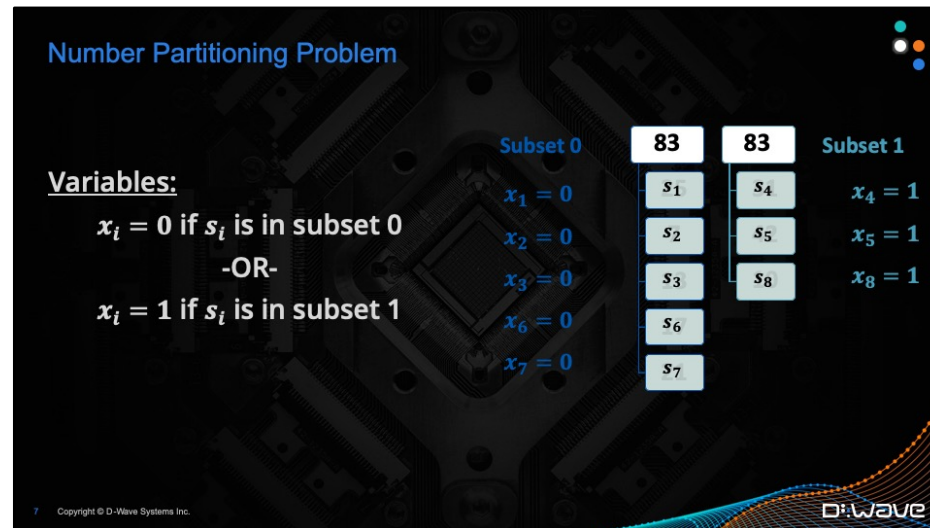
To write our QUBO for this problem we'll choose our binary variables to help us split the complete set into two subsets.

We label the subsets "0" and "1" and make a binary variable for each number in the complete set.

We put all of the numbers with binary variable equal to 0 in Subset 0, and all of the numbers with binary variable equal to 1 in Subset 1.



For our complete set of 8 numbers that we saw earlier, this is how our partition into Subset 0 and Subset 1 looks.

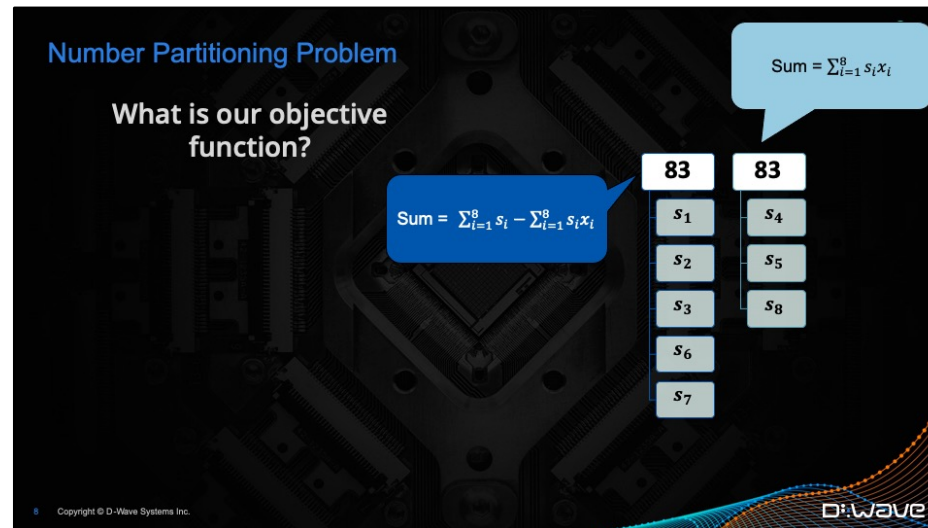


Since we're writing a program we might want to make it more general so that it will work for any set of numbers instead of this specific list.

To do that we can use the variable s_i to represent a number in our set.

Instead of a list of numbers like 13, 31, 42, 17 and so on, we have the set s_1, s_2, s_3, s_4 , and so on.

Keep in mind that these s variables are fixed numbers that are handed to us, while the x variables are our binary variables that the system will return.



How do we write our objective function using these variables that we've set up? We need to think of a way get the sum of each subset so that we can check if they are equal.

Starting with Subset 1 is a bit easier, since we can do a weighted sum as shown. This summation of s_i times x_i will add up only the numbers in subset 1.

For subset 0 we have to be a little bit more creative. There are a few ways to think about this, but the easiest way is to remember that subset 0 is everything that is NOT in subset 1.

If we add up all of the numbers and subtract the ones in subset 1 we are left with sum of subset 0. So here we see the summation of all of the numbers s_i minus what we figured out was the sum of subset 1.

Number Partitioning Problem

What is our objective function?

Sum = $\sum_{i=1}^8 s_i - \sum_{i=1}^8 s_i x_i$

Sum = $\sum_{i=1}^8 s_i x_i$

Find equal sums
-OR-
Minimize the difference

Minimize $\left[\sum_{i=1}^8 s_i - \sum_{i=1}^8 s_i x_i - \sum_{i=1}^8 s_i x_i \right]^2$

Copyright © D-Wave Systems Inc.

D-WAVE

Now to figure out our objective function we think back to the problem at hand.

When are these two sums equal?

Finding equal sums is the same as minimizing the difference, so we can use subtraction.

Our goal is to figure out when this difference is 0, so we need to square this expression to make sure that the smallest possible value is 0.


You can think about this like taking the absolute value of the difference – the smallest possible difference here is 0 which is just what we're looking for.

Number Partitioning Problem

General Objective Function

$$\text{minimize } \left(\sum_{l=1}^n s_l - 2 \sum_{l=1}^n s_l x_l \right)^2$$

Constant Number

10 Copyright © D-Wave Systems Inc. 

Now that we have our objective function we need to get it ready for an Ocean program.

Our requirement is that we have the linear and quadratic coefficients in a matrix, so we need to multiply this all out and simplify to figure out what those coefficients are.

Number Partitioning Problem

General Objective Function

$$\text{minimize } \left(C - 2 \sum_{i=1}^n s_i x_i \right)^2$$

Constant Number

11 Copyright © D-Wave Systems Inc.

To make things easier on the eyes, I'll replace this summation of the s_i 's with a single constant C .

Since there are no binary variables in this term it will just be a constant number that is found by adding up all of the numbers in our complete set.

Number Partitioning Problem

Finding our Matrix

First we need to multiply out our objective function:

$$\left(C - 2 \sum_{i=1}^n s_i x_i\right)^2 = C^2 - 4C \sum_{i=1}^n s_i x_i + 8 \sum_{i=1}^n \sum_{j=i+1}^n s_i s_j x_i x_j + 4 \sum_{i=1}^n s_i^2 x_i^2$$

12 Copyright © D-Wave Systems Inc. D-WAVE

Since the math gets a bit hairy I'll step through the multiplication and simplifying here for you.

Remember that Wolfram Alpha can come in handy for this!

The first step is to multiply everything out.


When we square this polynomial we get this big expression on the right, and remember that formula is on the written guide in the math refresh module.

Number Partitioning Problem

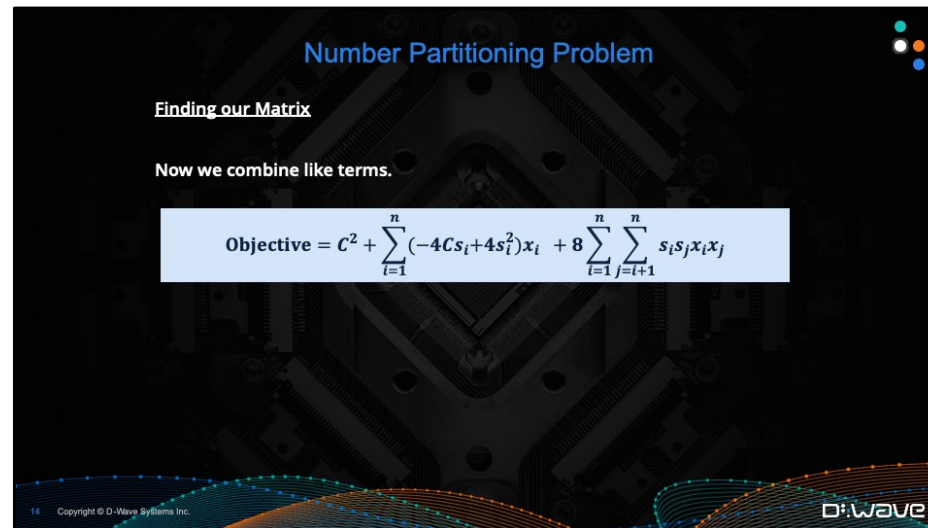
Finding our Matrix

Next we can replace x_i^2 with x_i .

$$\text{Objective} = C^2 - 4C \sum_{i=1}^n s_i x_i + 8 \sum_{i=1}^n \sum_{j=i+1}^n s_i s_j x_i x_j + 4 \sum_{i=1}^n s_i^2 x_i$$

13 Copyright © D-Wave Systems Inc. 

Next we can replace any squared binary variables with the linear version since 0 squared equals 0 and 1 squared equals 1.




Number Partitioning Problem

Finding our Matrix

Now we combine like terms.

$$\text{Objective} = c^2 + \sum_{i=1}^n (-4Cs_i + 4s_i^2)x_i + 8 \sum_{i=1}^n \sum_{j=i+1}^n s_i s_j x_i x_j$$

14 Copyright © D-Wave Systems Inc. 

Now we have linear terms that we can combine so that they are all together.

Number Partitioning Problem

Finding our Matrix

Now we can build our matrix.

$$\text{Objective} = \sum_{i=1}^n (-4Cs_i + 4s_i^2)x_i + \sum_{i=1}^n \sum_{j=i+1}^n 8s_is_jx_ix_j$$

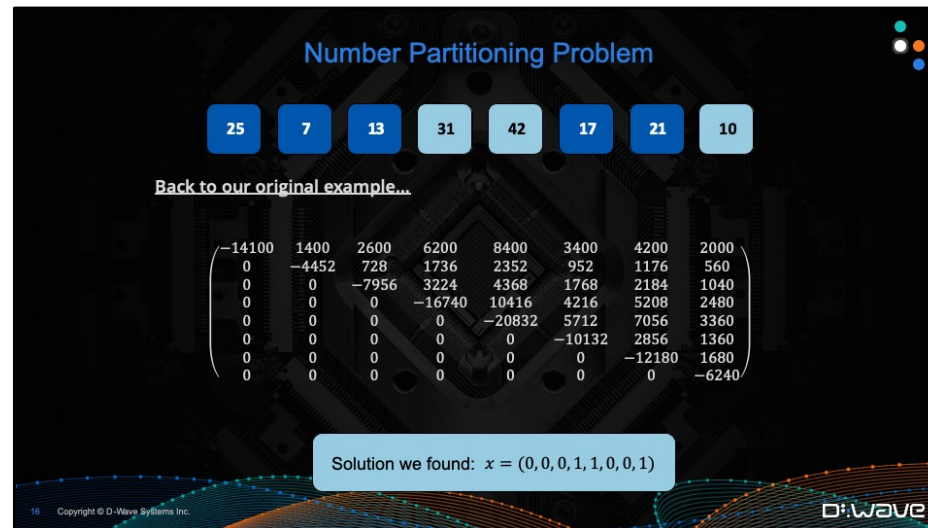
$q_{ii} = -4Cs_i + 4s_i^2$

$q_{ij} = 8s_is_j$

15 Copyright © D-Wave Systems Inc. D-WAVE

Finally we can ignore the constant that is added at the beginning, this C squared, since it's not attached to any binary variables.

Now we can clearly see our linear terms that will go along the diagonal and the quadratic terms.



Back to our original example our matrix would look like this shown here.

Notice that I've written this matrix as an upper-triangular matrix: the bottom left is all 0's.

This means that if I have the term $5x_1x_2$ in my QUBO I put the 5 in row 1, column 2, instead of row 2, column 1.

Does this matter?

The answer is yes and no. It doesn't matter if you put the 5 in position (1,2) in the matrix or position (2,1), but you cannot put it in both places.

The Ocean tools will take your QUBO matrix and add everything from the bottom half (the 0's here) to the top half.

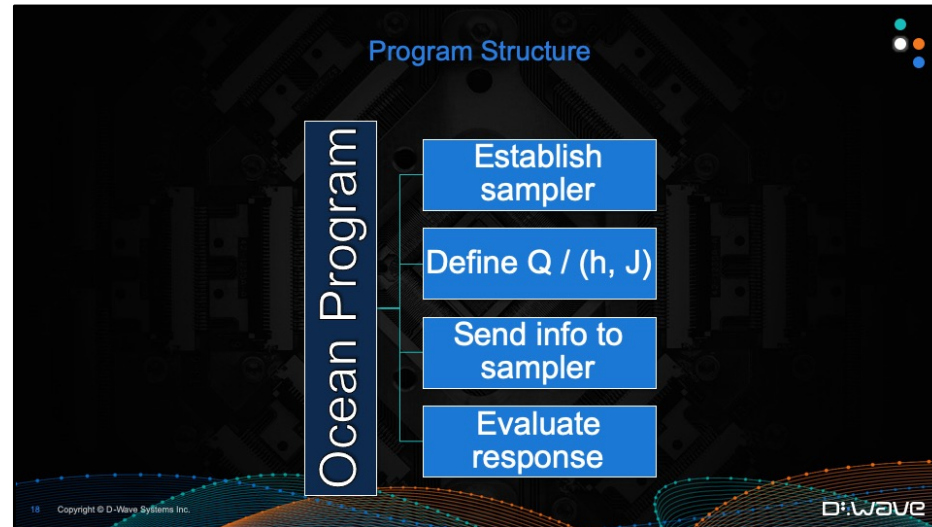
If you put your coefficient in both places then you're actually doubling all of the quadratic coefficients in your QUBO!

The solution we showed earlier with equal subset sums would be returned from the QPU as a binary string like this one shown.



To build up this matrix without any programming so that you can check your work, Wolfram Alpha is an amazing resource.

It's a good idea to have an example matrix handy so that you can check that you have written up your python program correctly.

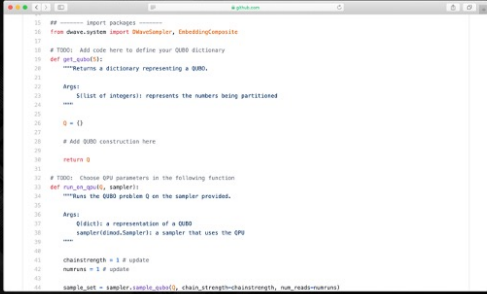


Now that we have our matrix we're ready to write our Ocean program.


Remember the four basic components of the program that we'll need to have: first we set up our sampler, we build our QUBO matrix, then we send the QUBO matrix to the sampler and look at the results.

Number Partitioning Problem Demo

- Open the file “npp.py”.
- Fill in the Python commands to build up your QUBO dictionary.
- Remember to look at the Python Prep slides if you aren't sure of syntax.



```
15 # Import packages
16 from dwave.system import DWaveSampler, EmbeddingComposite
17
18 # TODO: Add code here to define your QUBO dictionary
19 def get_qubois:
20     """Returns a dictionary representing a QUBO.
21
22     Args:
23         S (list of integers): represents the numbers being partitioned
24     """
25     Q = {}
26
27     # Add QUBO construction here
28     return Q
29
30 # TODO: Choose QUBO parameters in the following function
31 def run_qubo(sampler):
32     """Runs the QUBO problem Q on the sampler provided.
33
34     Args:
35         Q (dict): a representation of a QUBO
36         sampler (DWaveSampler): a sampler that uses the QPU
37     """
38     chainstrength = 3 # update
39     numruns = 3 # update
40
41     sample_set = sampler.sample_qubo(Q, chain_strength=chainstrength, num_reads=numruns)
```

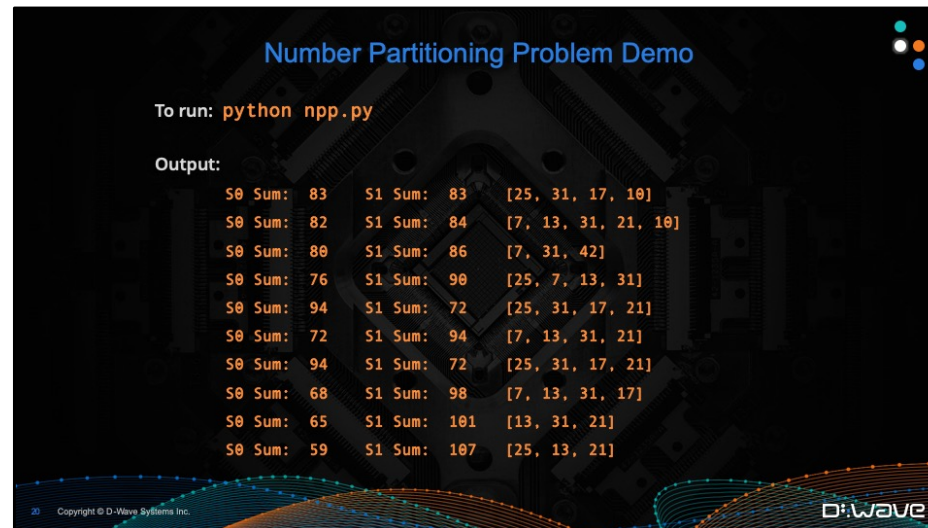
19 Copyright © D-Wave Systems Inc. 

For your assignment head over to the github repository shown on Canvas.

You'll be editing the file “npp.py”.

I've provided a program outline for you so that you only need to fill in a few sections.

The directions for what you need to complete and how it will be graded are on Canvas.

A terminal window titled "Number Partitioning Problem Demo" showing the output of a Python script. The output lists 10 solutions, each with two subset sums (S0 and S1) and the elements of one subset. The solutions are ordered by the S0 sum in descending order. The D-Wave logo is in the bottom right corner.

```
Number Partitioning Problem Demo

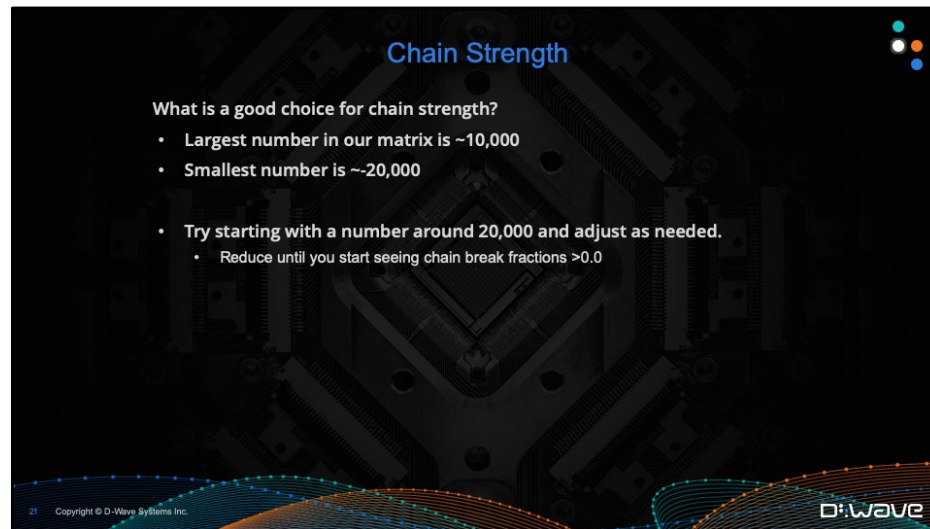
To run: python npp.py

Output:
S0 Sum: 83   S1 Sum: 83   [25, 31, 17, 10]
S0 Sum: 82   S1 Sum: 84   [7, 13, 31, 21, 10]
S0 Sum: 80   S1 Sum: 86   [7, 31, 42]
S0 Sum: 76   S1 Sum: 90   [25, 7, 13, 31]
S0 Sum: 94   S1 Sum: 72   [25, 31, 17, 21]
S0 Sum: 72   S1 Sum: 94   [7, 13, 31, 21]
S0 Sum: 94   S1 Sum: 72   [25, 31, 17, 21]
S0 Sum: 68   S1 Sum: 98   [7, 13, 31, 17]
S0 Sum: 65   S1 Sum: 101  [13, 31, 21]
S0 Sum: 59   S1 Sum: 107  [25, 13, 21]
```

If you run the file `npp.py` either by clicking the green “play” button in the IDE or by typing “`python npp.py`” on the command line you’ll see output like this.

Each line gives you one answer that was returned: it tells you the sum of each subset and the elements that are in one of the subsets.

The lowest energy solutions are listed first, so if you don’t see your best answers at the top you’ll want to double-check your QUBO matrix in your program.



Last but not least, you might want to try some different values for the chain strength parameter to get good results on this assignment. The default chain strength tool is a great start, but try tuning the parameter manually to see how it goes.

Remember from our embedding module that chain strength is what controls how strongly qubits are tied together to represent one of our binary variables.