The Bernstein-Vazirani Algorithm is designed to solve a very specific problem. Given a function f(x) that calculates the dot product of a binary string x and a secret binary string s, the algorithm aims to find the secret string s with a single query. The classical algorithm would require multiple queries proportional to the length of the secret string.

Here's an implementation of the classical algorithm to find the secret string s:

def dot\_product(x, s):

result = 0

for i in range(len(x)):

result += x[i] \* s[i]

return result % 2

def classical\_vazirani\_algorithm(f, n):

s = [0] \* n

for i in range(n):

x = [0] \* n

x[i] = 1

s[i] = f(x)

return s

# Define the secret binary string s

secret\_string = [1, 0, 1, 1]

# Define the function f(x) for the dot product of x and s

def f(x):

return dot\_product(x, secret\_string)

# Find the secret string using the classical algorithm

n = len(secret\_string)

found\_secret\_string = classical\_vazirani\_algorithm(f, n)

print("Secret string (classical):", found\_secret\_string)

This code defines the function f(x) and the classical algorithm to find the secret string s. It demonstrates the usage of the classical algorithm to find the secret string s. Note that the classical algorithm requires n queries to find the secret string, where n is the length of the secret string.