Program 3 Write-up

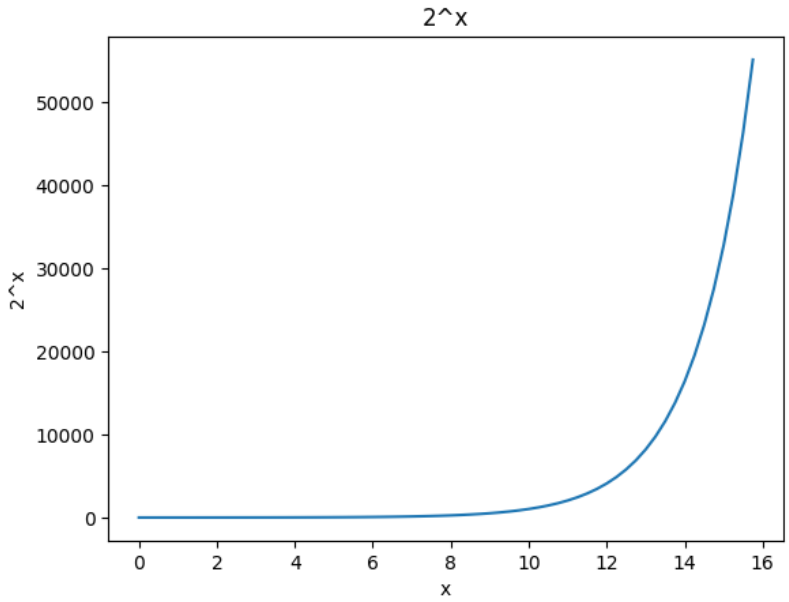
Variables:
P: Puzzle
B: # bits in puzzle
M: Message the challenge is looking for
h(): SHA-256 function
h(m): $[***...***, P]$

Looking for:
Evaluate how many attempts it takes to guess P for B=4,8,12,16

So the objective of this program is to determine how many guesses it will take in order to have the same last b bits as the original hash. Naively we can see that this distribution will be exponential as b goes up by looking at the probability of having a random bit match.

Bits	**1**	**2**	**...**	**n-1**	**n**
Prob	$\frac{1}{2^1}$	$\frac{1}{2^2}$	...	$\frac{1}{2^{n-1}}$	$\frac{1}{2^n}$

This would follow the following graph if our predictions are correct.



Now lets actually implement it. To do this we're going to do this in python using the following imports

```
from hashlib import sha256 as h
import matplotlib.pyplot as plt
```

```
import random as r
```

The method for generating the original M value is we grab a random string of length 50 using the following function.

```
def get_hash():
        s = ''
        for i in range(50): s += chr(r.randint(97,123))
        return h(s.encode()).hexdigest()
```

Then we run this random string through SHA-256 and use it as our puzzle. So now have our puzzle, the next step is to count how many guesses it will take for the last B bits of the the guess and the puzzle to match. To get the last B bits we use the following function

```
def get_last_n_bits(s: str, b: int) -> str:
        bitmap = {'0':0, '1':1, '2':2, '3':3, '4':4, '5':5, '6':6, '7':7,
'8':8, '9':9, 'a':10, 'b':11, 'c':12, 'd':13, 'e':14, 'f':15}
        bits = ''

        for i in range(b-1, -1, -1):
                c = s[-(i//4+1)]
                bits += str((bitmap[c]>>(i%4))&1)
        return bits
```

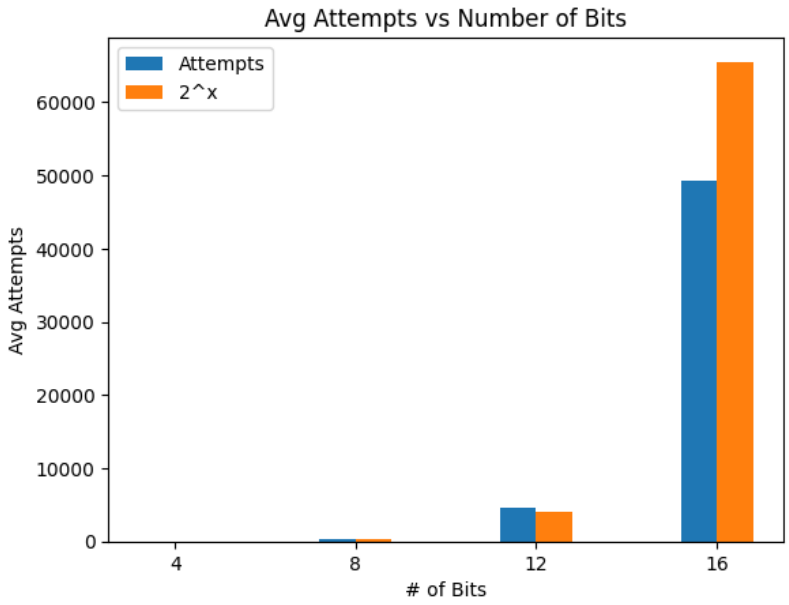
In order to get these averages we solve the puzzle 100 times and then take the average of all the attempts using the following function

```
iterations = 100

avgs = []
bits = [4,8,12,16]
for b in bits:
        avg = 0
        for i in range(iterations):
                m = get_hash()
                count = 1
                while not compare_last_n_bits(m, get_hash(), b): count += 1
                avg += count
        avg /= iterations
        avgs.append((b,avg))
```

From there we get the following graph, showing us that as the number of bits goes up the more guesses it will take and that this progression generally follows the same scaling as the equation

2^b where b is # of bits.



Bits	Avg guesses	2^n
4	15.65	16
8	234.18	256
12	4,510.16	4,096
16	49,267.89	65,536