Part 1

Part B fixes: I revised the time function, as well as measuring the 1..49 in the submitted function. The time function was changed as follows:

```
fn main() {
   let overall_start = SystemTime::now();
   for k in 0..=49 {
       let start = SystemTime::now();
       let result = fibonacci(k);
       let duration = start.elapsed().expect("Time measurement failed");

       println!("fibonacci({}) = {}, Time: {:?}", k, result, duration);
   }

   let overall_duration = overall_start.elapsed().expect("Time measurement failed");
   println!("Total execution time: {:?}", overall_duration);}
```

Explanation:

When doing this problem, I ran through a variety of different scenarios from 0-10, to 0-49. From this data, I compiled a table (see journal pdf). I forgot to change the for i loop back to 0-49 thus resulting in the mistake when submitting the code.

During problem my understanding changed because I realized that I need to be more thorough with my code especially when submitting, specifically to check for both the timing method, as well as with the for i loop.

Part 2

I wrote, “With u8 and Cargo run –release, the numbers start looping after f(13), however with Cargo run, it doesn’t work at all, instead giving “thread 'main'panicked at src/main.rs:6:20: attempt to add with overflow,”and because it isn’t in release, it doesn’t work with overflow. It may be better to run the regular version over the –release version due to accuracy, because after f(13), the –release version becomes inaccurate.”

My understanding has changed, because while I was correct here, I was not specific enough in the reasoning for this particular part–I should have added that the reasoning was because of stack overflow-where the memory exceeds what could be held in u8 -> needing a integer bit size much larger than that to hold the adequate fibonacci number.

Part 3

Used a different equation to obtain the same result, omitting the for loop, and instead using a different equation (which we learned about in ds120, proven by induction).

While receiving the same answer, I used a more efficient method rather using a for loop to get the answer.

This is what the code should look like.

```
use std::io;
fn sumcube(k: u8) -> u32 {
    let mut sum = 0;
    for i in 1..=k as u32 {
        sum += i.pow(3);
    }
    sum
}
```

I was not thorough with reading the instructions, instead believing that it was smarter to use the more efficient equation as opposed to the for i solution, following the mathematical equation given as a part of the solution.