Ha M Le

Professor San Skulrattanakulchai

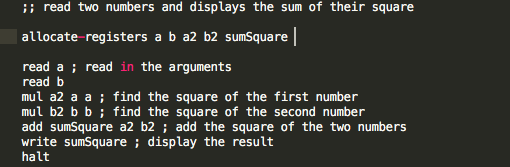
2019 mcs-178-001

15 September 2019

In this project, we are asked to provide different solutions to the same problem using SLIME programs. From the data and statistic gathered from SLIME, we will draw conclusions about the efficiency of the three different programs that we have come up with. We also have a warm up exercise in the beginning of the project.

**Task 1: Warm up**

The first task of the project requires us to write a program that take 2 numbers and produce the sum of their squares. Below is the SLIME code for the exercise.



**Task 2:**

This program read in a number and its exponent by the user, solve the equation by using a while-loop, and produce the result to the console. Since the program only has one while loop, the number of instructions it takes to produce the result will be (instructions), with e being the exponent.

The constant 5 in the equation above is the number of instructions inside the loop, while the constant 10 refers to the instructions outside the while-loop, which are only used once during the program.

**Task 3:**

This function also read in a number and its exponent from the user, solve the equation and produce the answer on the console. However, this program handles the even exponent slightly different from the function in the second task.

If the program notices an even exponent, it will divide the exponent by two and square the number b. This makes the function more efficient than the first one when dealing with even exponent (especially with exponent with the form ). The function handle odd exponent with the same method as the previous program.

For exponent with the form, the number of instructions that the program needs to finish executing will be , which will be shorten as .

The constant 12 is the number of instructions outside the loop, which are executed only once during the program. The expression is the amount of time the function deals with even exponent, while the constant 7 is the number of instructions it takes to finish a while-loop and the if label (in the case of the even exponent). The last constant in the formula, the number 10, refers to the very last step of the program, when the exponent equals to 1. When the exponent equals to 1, the program will jump back to the while loop (which has 5 instructions), then jump to the “odd case” (which has 3 instructions), jump back to the while-loop label one more time to execute the comparison (which takes 2 instruction), then jump to the end label to display the result. That gives us a total of 10 more instructions, which is why the formula is.

However, this formula only works in the case if the exponent is the power of 2, such as 2,4,8,16 and 32. In other cases, the program runs in a slower speed, especially when dealing with odd exponent. However, the speed of this function proves to be much more efficient than the previous one.

**Task 4: The recursive function**

The fourth task asks us to solve the aforementioned problem but using a recursive approach. Since the program takes a lot of instructions to store and load data to the stack point, this program takes the most time out of the three functions.

The formula for the number of instructions its take to execute exponentiation is , which is shorten as .

The constant 9 is the number of instructions outside the store, load and baseCase labels, which are only executed once during the program. is the number of instructions it takes to store, load and handle multiplications. The constant 2 is the number of instructions in the base case, and we also have to take into account the extra jeqz in the very end of the program, when e = 0. All of these analyses lead us to the final formula for the speed of this program: (instructions).

**Real statistics gathered from SLIME**

In the previous parts of this report, we have reached three different formula for the three functions presented in task 2,3 and 4:

The predicted formula for the number of instructions that the first SLIME program produces is (instructions).

The predicted formula for the number of instructions that the second SLIME program produces is (instructions).

The predicted formula for the number of instructions that the third SLIME program produces is (instructions).

After coming up with the formulas, we have to test whether if our predicted results are corrected. Below is the table displaying the statistics gather from SLIME. We only test with the exponents which are the power of 2, since the formula in task 2 only work with them.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Input | Expected (task 2) | Real  (task 2) | Expected (task 3) | Real  (task 3) | Expected (task 4) | Real  (task 4) |
|  | 20 | 20 | 29 | 29 | 40 | 40 |
|  | 30 | 30 | 36 | 36 | 68 | 68 |
|  | 50 | 50 | 43 | 43 | 124 | 124 |
|  | 90 | 90 | 50 | 50 | 236 | 236 |
|  | 170 | 170 | 57 | 57 | 460 | 460 |

We can clearly observe that all of our predicted results match perfectly with the real results taken from SLIME’s counters.

**Conclusion**

The first program takes least instructions to perform small exponents, but as the exponents get larger, the program from task 3 performs significantly faster than task 2 program. The program that takes the most amounts of time to finish the exponentiation is the recursive program, since it is the most complex program that requires multiple steps. Overall, we can see that different methods require different amount of time to execute, and it is important to find a solution that takes least time to process.