Course: CSC 220.02
Student: Kullathon “Mos” Sitthisarnwattanachai, **SFSU ID:** 921425216
Teammate: n/a, **SFSU ID:** n/a
Assignment Number: 03
Assignment Due Date & Time: 04-02-2020 at 11:55 PM

PART A – The Linked Bag, 20 points

- See assignment03PartA in the attached files.

PART B – Stack, 15 points

- See assignment03PartB in the attached files.

PART C – Recursion, 15 points

- See assignment03PartC in the attached files.

PART D – Recursion, 10 points

- See assignment03PartD in the attached files.

PART E – The Efficiency of Algorithms, 15 points

1. **Show how you count the number of operations (not only basic operations) required by the algorithm, 5 points:**

```
int i, n = 5, sum = 5;
for (i = 5; i < 3 * n; i++) {
   sum *= n + i * 7 + 37;
}
```

- **Determining the number of iterations**

  Because n will always be 5, we can determine that the loop body will stop executing when `i` has a value of 15. Therefore, the loop body will execute 10 times.

- **Once-off assignments**

  There are 3 assignments that occur once only: the assignment of n and `sum` prior to the loop, and the initial assignment of `i` in the initialization expression of the loop.

  - So far, there are 3 operations in total.

- **Termination expression**

  The termination expression of the loop has 2 operations: one for comparison, and the other for multiplication. The expression will be executed 10 + 1 times, once for each execution of the loop, and once more when it is determined that the loop will no longer execute. As such, the number of operations in the termination expression is $11 \times 2 = 22$.

  - $3 + 22 = 25$

    This brings the running total number of operations to 25.

- **Incrementation expression**

  The incrementation expression can be rewritten as `i = i + 1`.

  There are 2 operations in the incrementation expression: one for the addition, and the other for the re-assignment of `i`. The statement will be executed 10 times, once for each execution of the loop. As such, the total number of operations in the incrementation expression is: 10 × 2 = 20.

    - 3 + 22 + 20 = 45

      This brings the running total number of operations to 45.

- **Body**

  The sole statement in the loop body can be rewritten with additional parenthesis for clarification as `sum = sum * (n + (i * 7) + 37);`.

  There are 5 operations in the above statement: one for each mathematical operator, and one for the re-assignment of `sum`. The statement will be executed 10 times, once for each execution of the loop. As such, the total number of operations in the body is 10 × 5 = 50.

    - 3 + 22 + 20 + 50 = 95

      This brings the running total number of operations to 95.

- **Total**

  Since there are no further operations after the execution of the loop, **the total number of operations in the algorithm is 95**.

2. **Consider Loop A and Loop B in the box to the right, 5 points:**

   Although Loop A is O(n) and Loop B is O(n^2), Loop B can be faster than Loop A for small values of n. Design and code a creative experiment to find a value of n for which Loop B is faster.

- See assignment03PartE in the attached files.

  For your convenience, the test for Loop A and Loop B can also be run at an online environment here: https://replit.com/@mosguinz/Loop-A-vs-Loop-B.

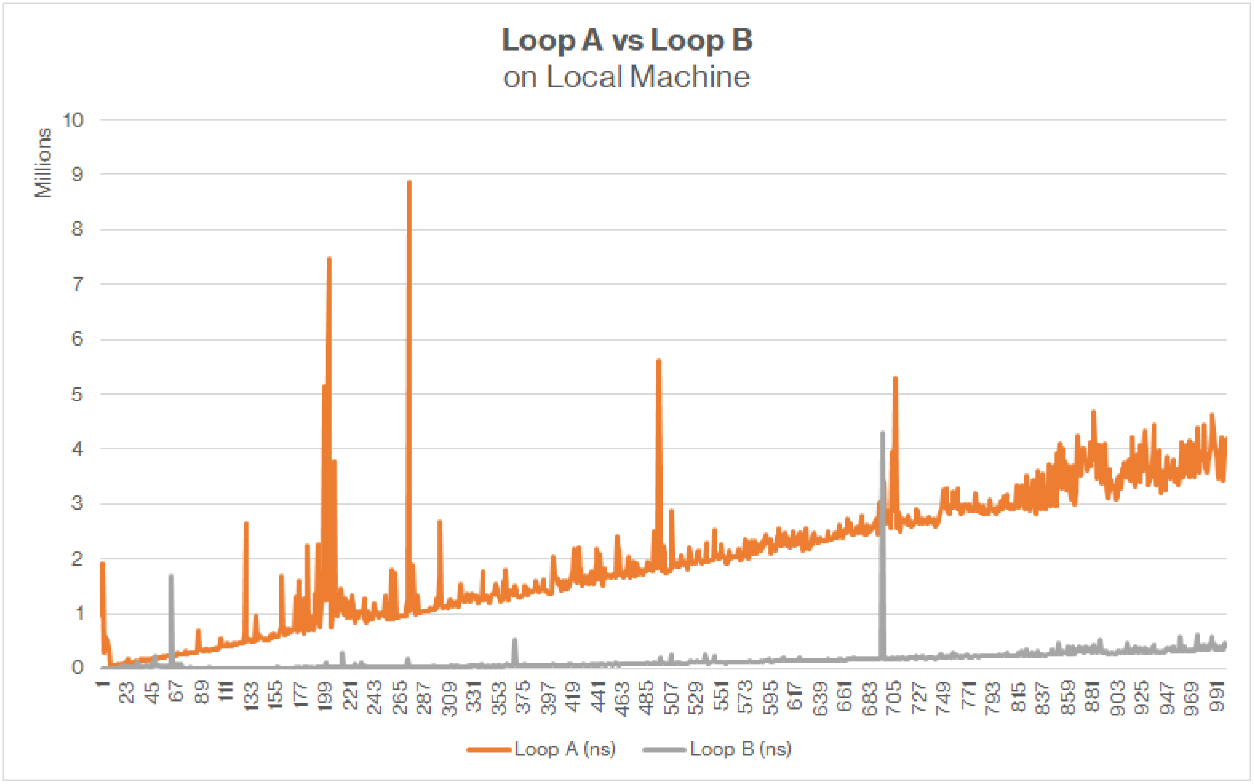
  My initial assumption was that Loop B would be faster than Loop A for all values such that $n < 10000$. This is because both Loop A and B are exactly identical with the exception that the inner loop in Loop A would terminate when its counter reaches 10,000. Whereas in Loop B, the inner loop would execute n times.

  However, when I tested this assumption, I found that Loop A would beat Loop B at a much smaller value of n. Specifically, $n = 95$.

  Initially, I took the naïve approach of stopping the testing loop (where n is being incremented) as soon as Loop A beats Loop B with the assumption that once beaten, the difference in the time would only grow larger.

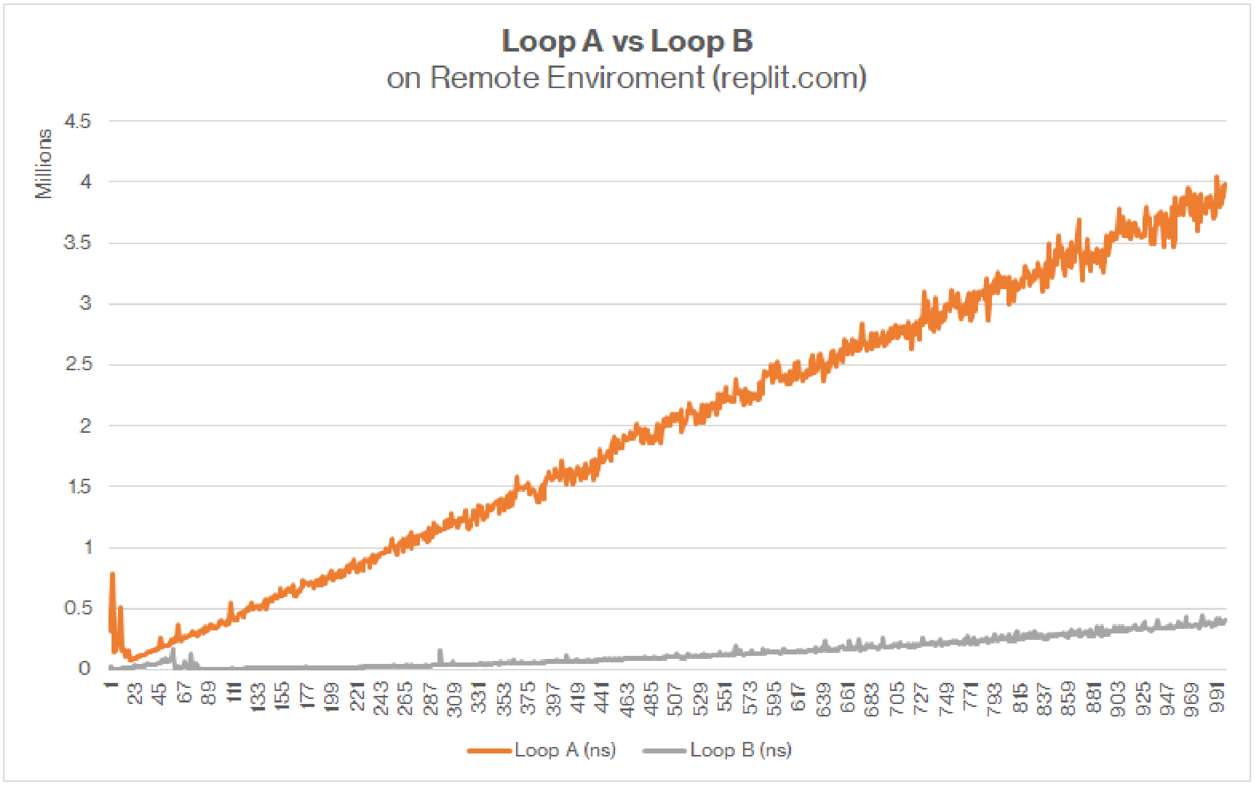
  It turned out that my initial result was an anomaly, so I added a case in order to ensure that the timing had to be beaten ten times in a row before concluding the value. I landed at the value $n = 55$ with this method, only to notice more anomalies along the way.

  So, I ended up running the test for all values of $n \in [1, 1000]$ and then graphing my findings. Below is the graph for the time taken of Loop A vs Loop B from $n = 1$ to $n = 1000$, inclusive.



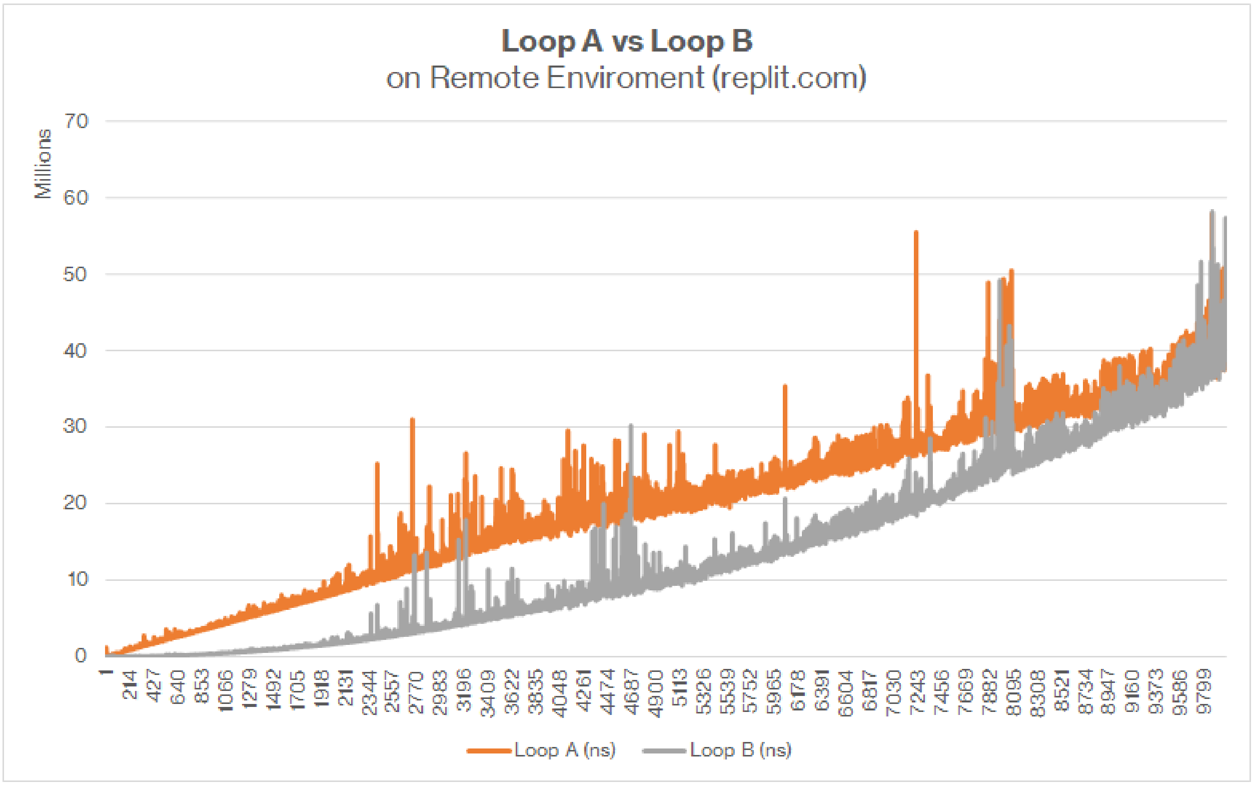
As you may notice, there are many irregular peaks in the graph for both Loop A and Loop B. Although, you can still see the trends upwards for Loop A and Loop B.

I thought the irregularities may be due to the environment that the code is running in. So I tried running the same code on an online IDE, repl.it, to see if there was any difference.



Notice that, even on a smaller scale, the execution times on the remote environment are much more consistent than the one on my computer. Here, the trend is even clearer that the execution time for Loop B is approaching Loop A.

So to confirm my initial suspicion, I ran the code on the remote environment in order to test the loops up to $n = 10000$.



Despite the outliers, you can still clearly see the trend that Loop B grows at an increasing rate, and eventually reaching meets Loop A as n approaches 10,000.

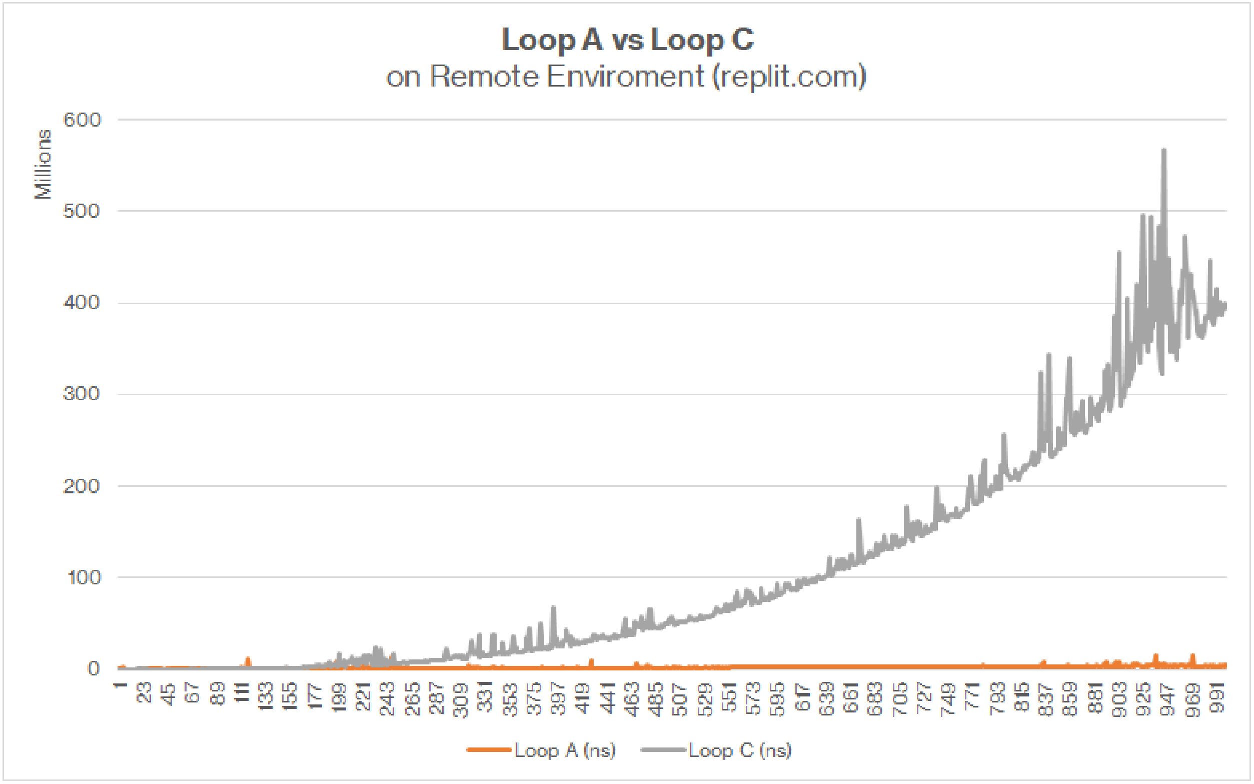
From the observations made, I conclude that Loop B is faster for all positive integer n, such that $n < 10000$.

3. Repeat question E.2 but use Loop C in place of Loop B, 5 points:

- See assignment03PartE in the attached files.

  For your convenience, the test for Loop A and Loop C can also be run at an online environment here: https://replit.com/@mosguinz/Loop-A-vs-Loop-C.

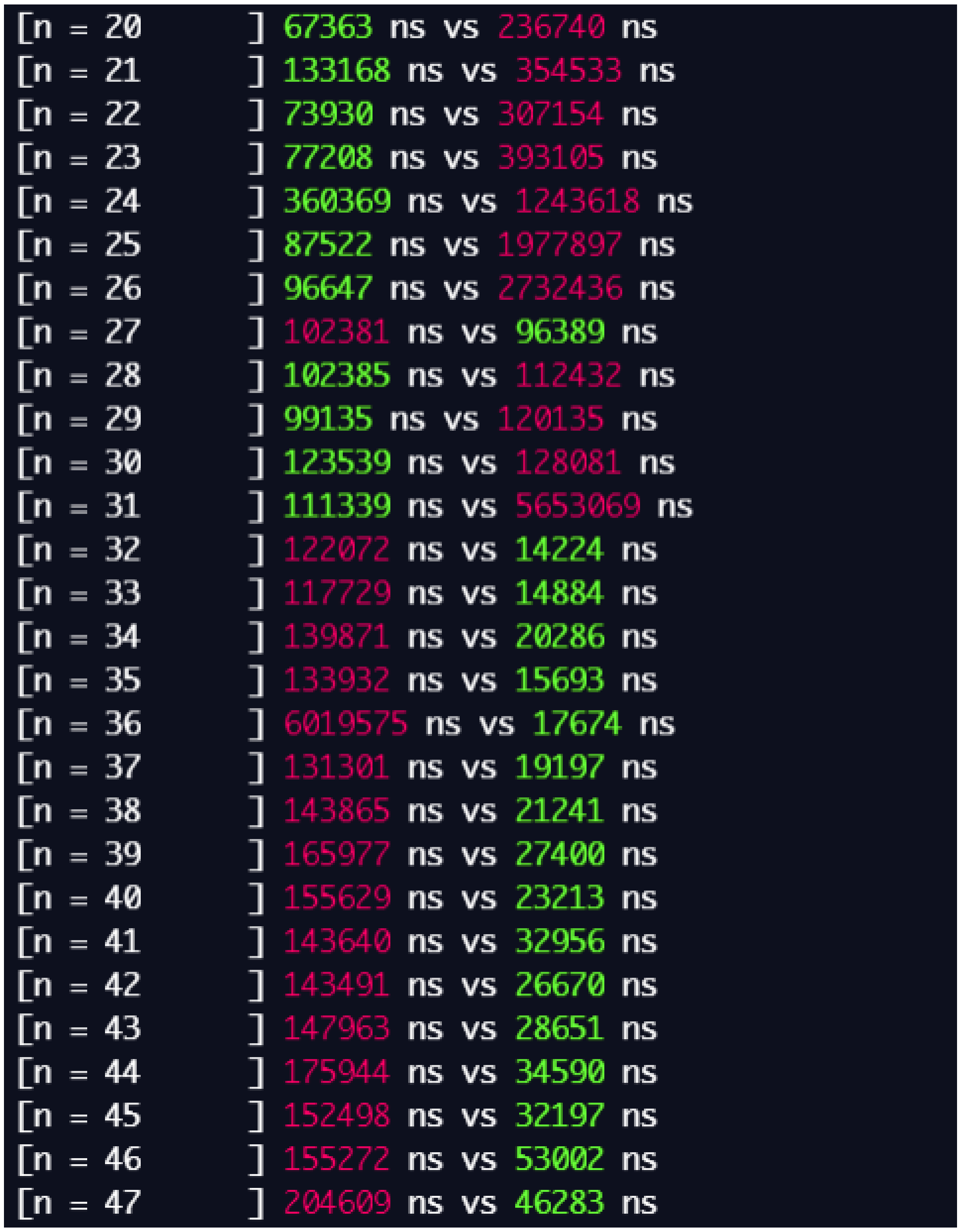
Given the consistency of the repl.it over my local environment, I have chosen to use it to run the tests for Loop C. Below is the graph for the time taken of Loop A vs Loop C from $n = 1$ to $n = 1000$, inclusive.



Here, you can see that Loop C grows at a much faster rate than Loop A. From observation, the execution of the code starts to slow down by a significant amount as n approached 1,000.

Loop C grows at a much faster rate compared to the growth of Loop B. This is because Loop C has one more level of nested loop compared to that of Loop B.

Using the output from the console, we can see the performance of Loop A starts to consistently beat Loop C as n approaches 30.



```
[n = 20        ] 67363 ns vs 236740 ns
[n = 21        ] 133168 ns vs 354533 ns
[n = 22        ] 73930 ns vs 307154 ns
[n = 23        ] 77208 ns vs 393105 ns
[n = 24        ] 360369 ns vs 1243618 ns
[n = 25        ] 87522 ns vs 1977897 ns
[n = 26        ] 96647 ns vs 2732436 ns
[n = 27        ] 102381 ns vs 96389 ns
[n = 28        ] 102385 ns vs 112432 ns
[n = 29        ] 99135 ns vs 120135 ns
[n = 30        ] 123539 ns vs 128081 ns
[n = 31        ] 111339 ns vs 5653069 ns
[n = 32        ] 122072 ns vs 14224 ns
[n = 33        ] 117729 ns vs 14884 ns
[n = 34        ] 139871 ns vs 20286 ns
[n = 35        ] 133932 ns vs 15693 ns
[n = 36        ] 6019575 ns vs 17674 ns
[n = 37        ] 131301 ns vs 19197 ns
[n = 38        ] 143865 ns vs 21241 ns
[n = 39        ] 165977 ns vs 27400 ns
[n = 40        ] 155629 ns vs 23213 ns
[n = 41        ] 143640 ns vs 32956 ns
[n = 42        ] 143491 ns vs 26670 ns
[n = 43        ] 147963 ns vs 28651 ns
[n = 44        ] 175944 ns vs 34590 ns
[n = 45        ] 152498 ns vs 32197 ns
[n = 46        ] 155272 ns vs 53002 ns
[n = 47        ] 204609 ns vs 46283 ns
```