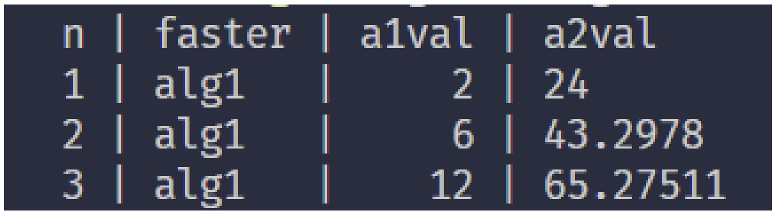
Ryan Christopher
MET CS566
Assignment 1

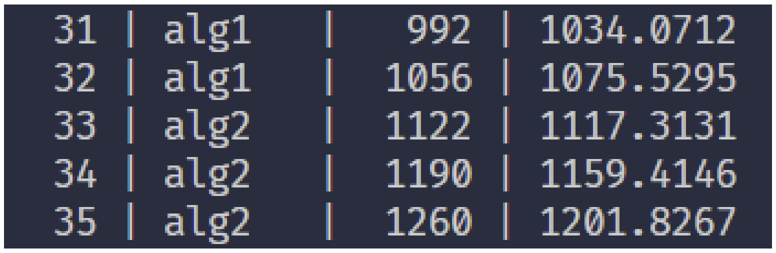
1. Suppose algorithm 1 does f(n) = $n^2 + n$ steps in the worst case, and algorithm 2 does g(n) = $14n^{1.25} + 10$ steps in the worst case, for inputs of size n. For what input sizes is algorithm 1 faster than algorithm 2 (in the worst case)?

   Algorithm 1 is faster than algorithm 2 where n <= 32. Each value of n leading up to 32 has algorithm 1 performing fewer steps than algorithm 2, and when n = 32 algorithm 1 performs 1056 steps in the worst case and algorithm 2 performs 1076 steps (rounded, specifically 1075.5295) in the worst case.



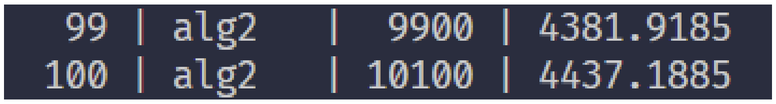
```
  n | faster | a1val | a2val
  1 | alg1   |     2 | 24
  2 | alg1   |     6 | 43.2978
  3 | alg1   |    12 | 65.27511
```

   However, every value for n after has algorithm 2 performing faster than algorithm 1, where when n = 33 algorithm 1 performs 1122 steps in the worst case and algorithm 2 performs 1117 (rounded, specifically 1117.3131) steps in the worst case.



```
 31 | alg1   |   992 | 1034.0712
 32 | alg1   |  1056 | 1075.5295
 33 | alg2   |  1122 | 1117.3131
 34 | alg2   |  1190 | 1159.4146
 35 | alg2   |  1260 | 1201.8267
```

   The gap between algorithm 1 and algorithm 2 becomes larger for each value of n after 33, and by the time n = 100, algorithm 1 performs 10,100 steps in the worst case while algorithm 2 performs 4437 steps (rounded, specifically 4437.1885) in the worst case.



```
  99 | alg2   |  9900 | 4381.9185
 100 | alg2   | 10100 | 4437.1885
```

2. Prove or disprove: T(n) = SUM{ i^4 } = Θ(n^4), where *i* changes from *i* = 1 to *i* = *n*.

SUM{ i^4 } != Θ(n^4)
Proof by contradiction:
Suppose SUM{ i^4 } = Θ(n^4). Then there exists positive constants c_1 and c_2 where SUM{ i^4 } lies between $c_1(n^4)$ and $c_2(n^4)$. If c_1 were to be 1 and c_2 were 2 then SUM{ i^4 } would lie below $c_1(n^4)$ and above $c_2(n^4)$ only when 1 < n < 8. If c_1 were to be 1 and c_2 were 5 then SUM{ i^4 } would lie below $c_1(n^4)$ and above $c_2(n^4)$ only when 1 < n < 23. This trend continues regardless of how high c_2 increases in value as n approaches infinity where SUM{ i^4 } will always lie above $c_2(n^4)$ at a certain n_0. This is a contradiction to the definition of Θ, therefore SUM{ i^4 } != Θ(n^4).

3. Write out the algorithm (pseudocode) to find *K* in the **ordered** array by the method that compares *K* to every mth entry (1 < m is less than array size n) until *K* itself or an entry larger than *K* is found, and then, in the latter case, searches for *K* among the preceding m.

```
vals = ordered array of values
k and m are given as parameters
index = 0
# search array every mth entry
while index ⩽ length of vals:
     if vals[index] == k then:
          return true
     else if vals[index] > k then:
          break loop
     else:
          index += m
          if index > length of vals:
               index = length of vals − 1
               break loop
# if entry larger than k found, search for k among preceding m
while vals[index] ⩾ k:
     if vals[index] == k:
          return true
     index -= 1
return false
```

How many comparisons does your algorithm do in the worst case?

In the worst case, my algorithm performs (n/2) + 1 comparisons where n is the number of values in the ordered array.

Test your algorithm for m = 3, 4, 5 ... and come up with the general formula.

Suppose vals = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 20, 25, 50, 100]
if m = 3 and k = 10:
    0, 3, 6, 9, 50, 25, 20, 9 → return false (8 comparisons)
if m = 4 and k = 9:
    0, 4, 8, 50, 25, 20, 9 → return true (7 comparisons)
if m = 5 and k = 21:
    0, 5, 20, 100, 50, 25, 20 → return false (7 comparisons)

The general formula for the worst case of this algorithm would be:
f(k, vals) = (len(vals)/2) + 1.

4. Compare two algorithms to raise an integer to an integer power. Assume in both cases that n, the exponent, is a power of 2.
<u>Algorithm 1:</u>
$x^n = x * x^{(n-1)}$ $x^0 = 1$
<u>Algorithm 2:</u>
$n = 2^m$ x^n = ((x ** 2) ** 2) ** 2 ...
note: the symbol power (**) is used m times here, i.e., x ** 8 = ((x ** 2) ** 2) ** 2, because 8 = 2 ** 3.
Which algorithm is more efficient with respect to the number of multiplications?

Algorithm 1 is more efficient with respect to the number of multiplications, as algorithm 1 contains a total of 2 multiplications. The first being $x^{(n-1)}$ and the second multiplying by x. Algorithm 2 requires m number of multiplications, which will require more multiplications than algorithm 1 for all values of m greater than 2.

5. You are given 18 bags of gold coins. One bag contains all false coins that weigh several grams less. What is the minimum number of weighting required to identify the false coins bag? Does the answer depend on the balance type: digital or not?

   This question depends on if we know the number of grams that the real coins and false coins weigh. If we know the exact amount that the real and false coins weigh, then only 1 weighting is required. To achieve this, you would take 1 coin from the first bag, 2 coins from the second bag, 3 coins from the third bag, and so on until you take 18 coins from the eighteenth bag. The expected total would be 171 multiplied by the weight of the real coin (as 171 coins in total would be used in this weighting). Depending on the total, we could calculate which bag contains the false coins by subtracting the weight of the real coins from the false ones, and however many times that amount is missing is the number of the bag containing the false coins.

   If we do not know the weight of the real and false coins, then two weightings would need to occur. We would still use the same strategy of 1 coin from the first bag, 2 coins from the second, and so on until we get to 17 coins from the seventeenth bag. We would then weigh the coins from bags 1 through 17. After that weighting we would weigh 18 coins from the eighteenth bag. This would let us determine the real weight by calculating the average weight of the coins from bags 1-17 and bag 18. Whichever average is higher would be the weight of the real coins, and could then be used to determine which bag contains the false by using the same logic as the first solution.

   The answer does not depend on the balance type as long as you are able to weigh the items in grams and can zero the scale again.