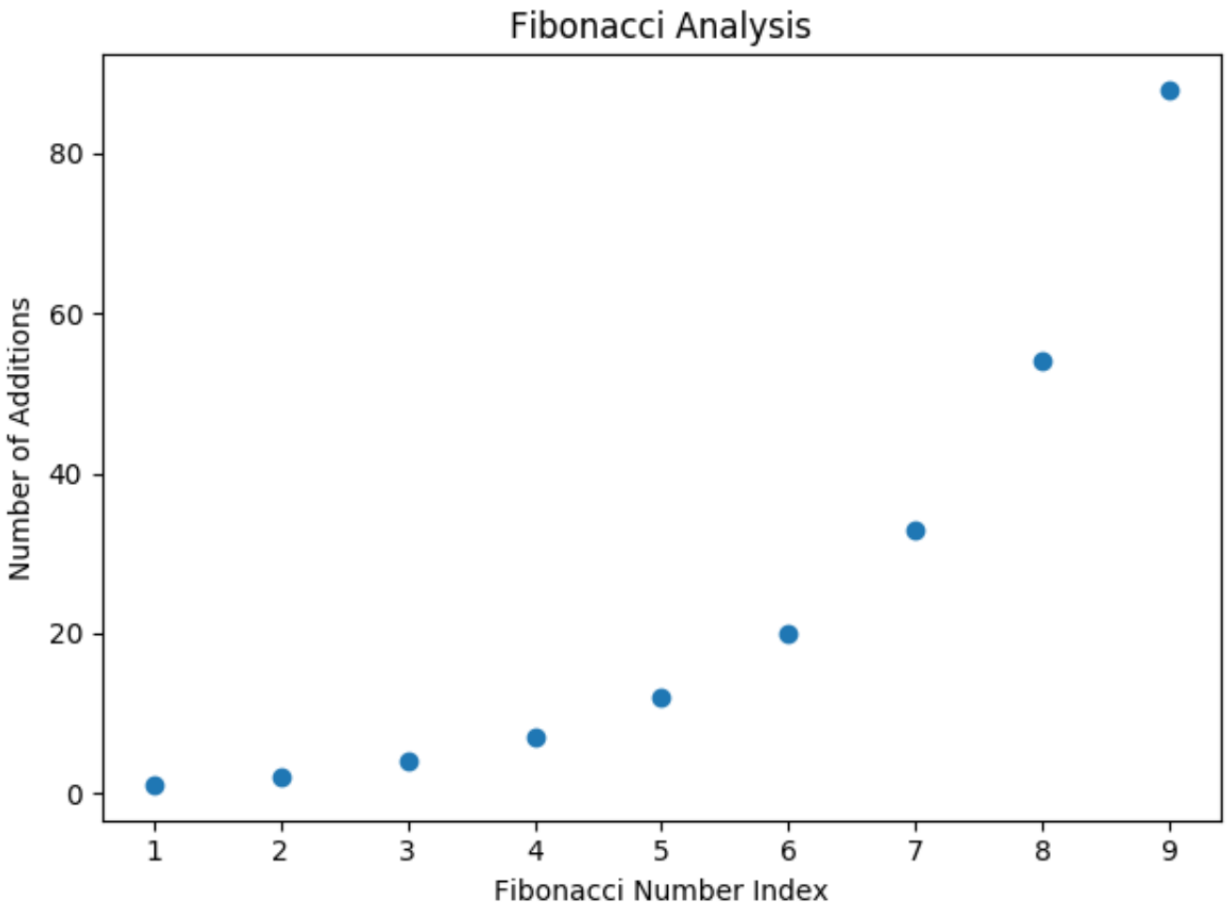
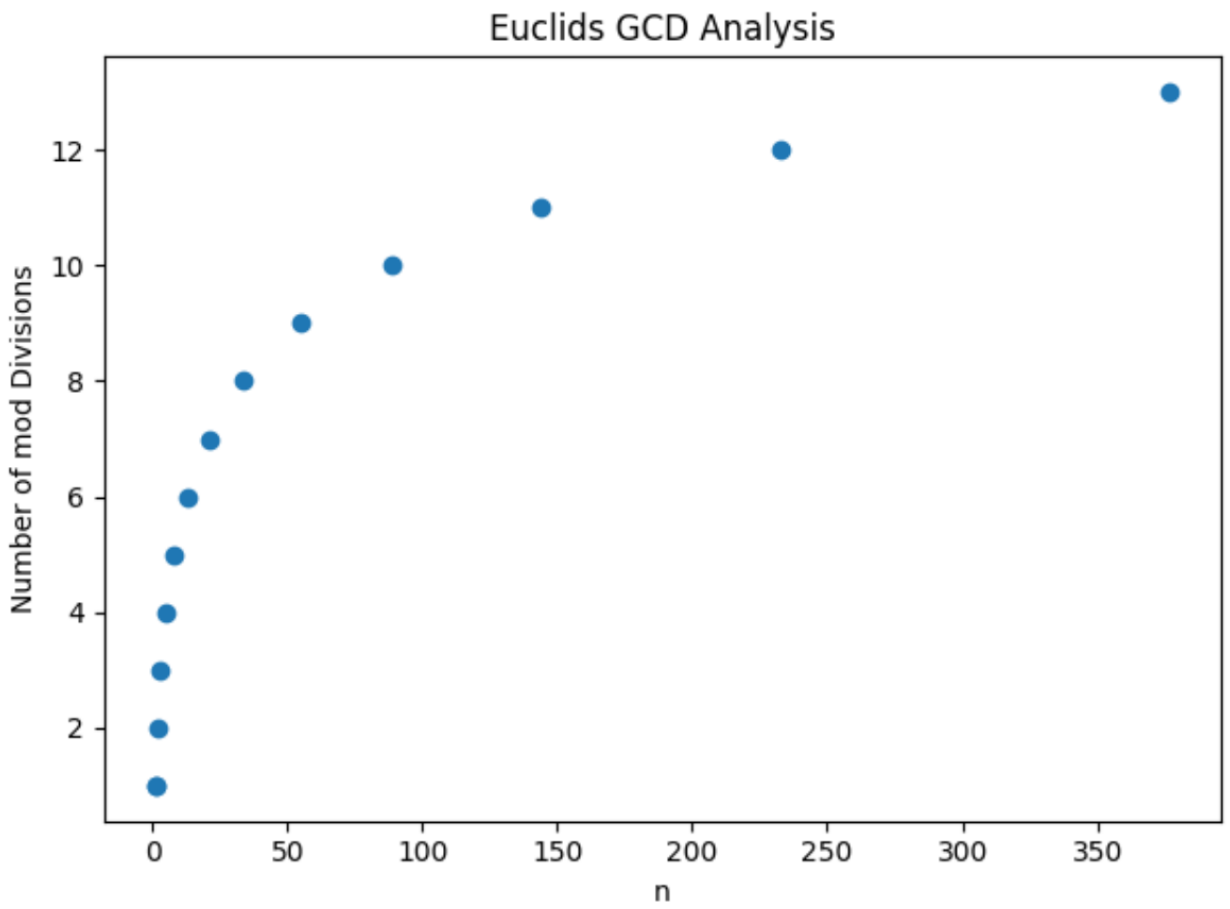
Task 1

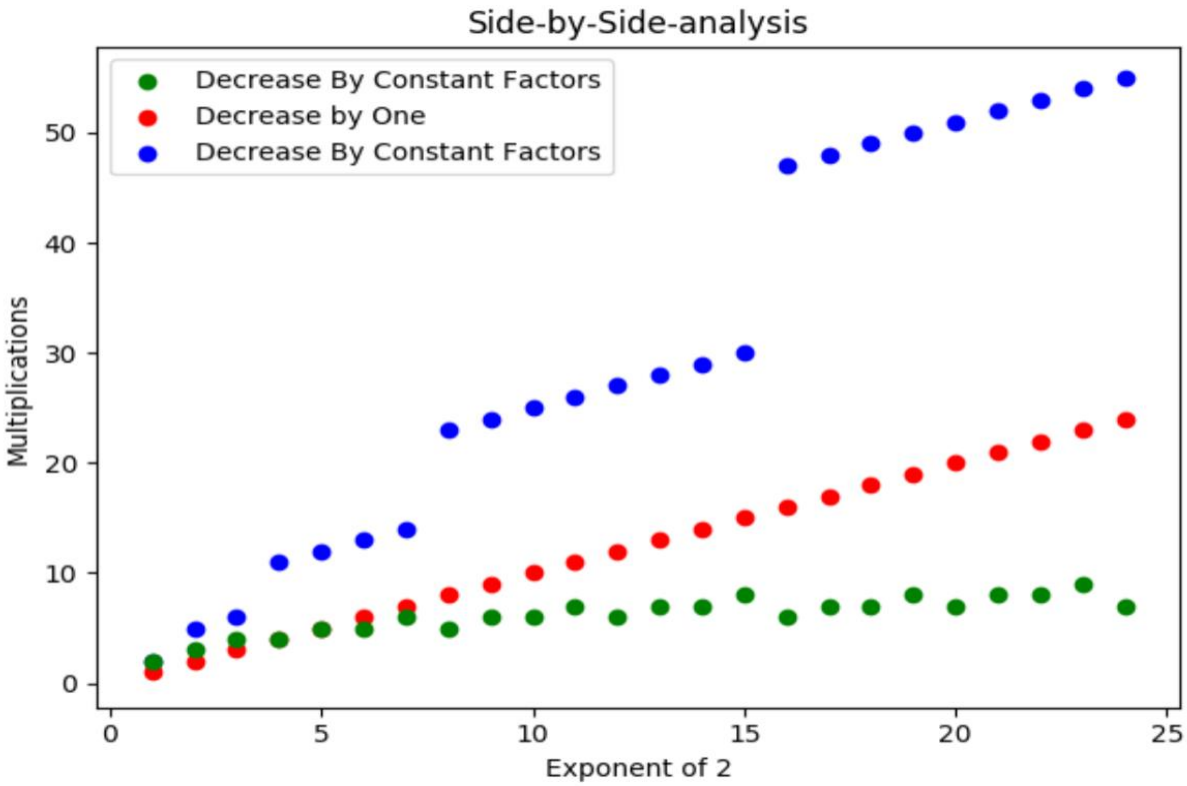


As n gets larger, the number of addition operations grows exponentially in A(k) because of the dual recursive calls in A(k): A(k-1) and A(k-2). This ends up with the complexity becoming $\theta(n^2)$.



The efficiency class of Euclid's algorithm with two Fibonacci values, k-1 and k, has a linear growth rate because the number modulo operations and k are equal: n modulo operations = k. This means that the complexity of the algorithm is $\theta(n)$.

Task 2

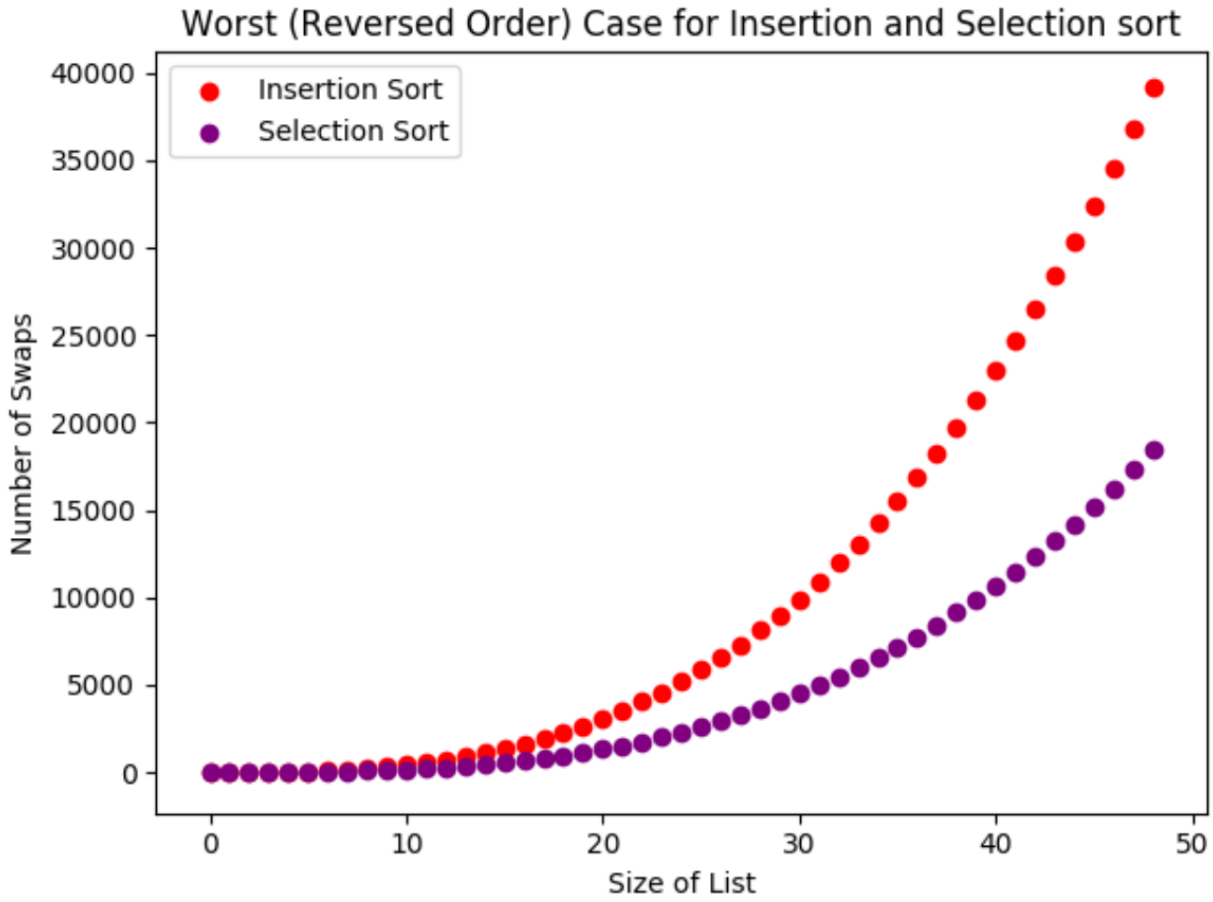


- The efficiency class of the Decrease-by-One algorithm of exponentiation is linear because the number of multiplications and n values are equal: *k* multiplications = n. Just like Euclid's algorithm combined with the Fibonacci sequence, the complexity of the Decrease-by-One algorithm is $\theta(n)$.

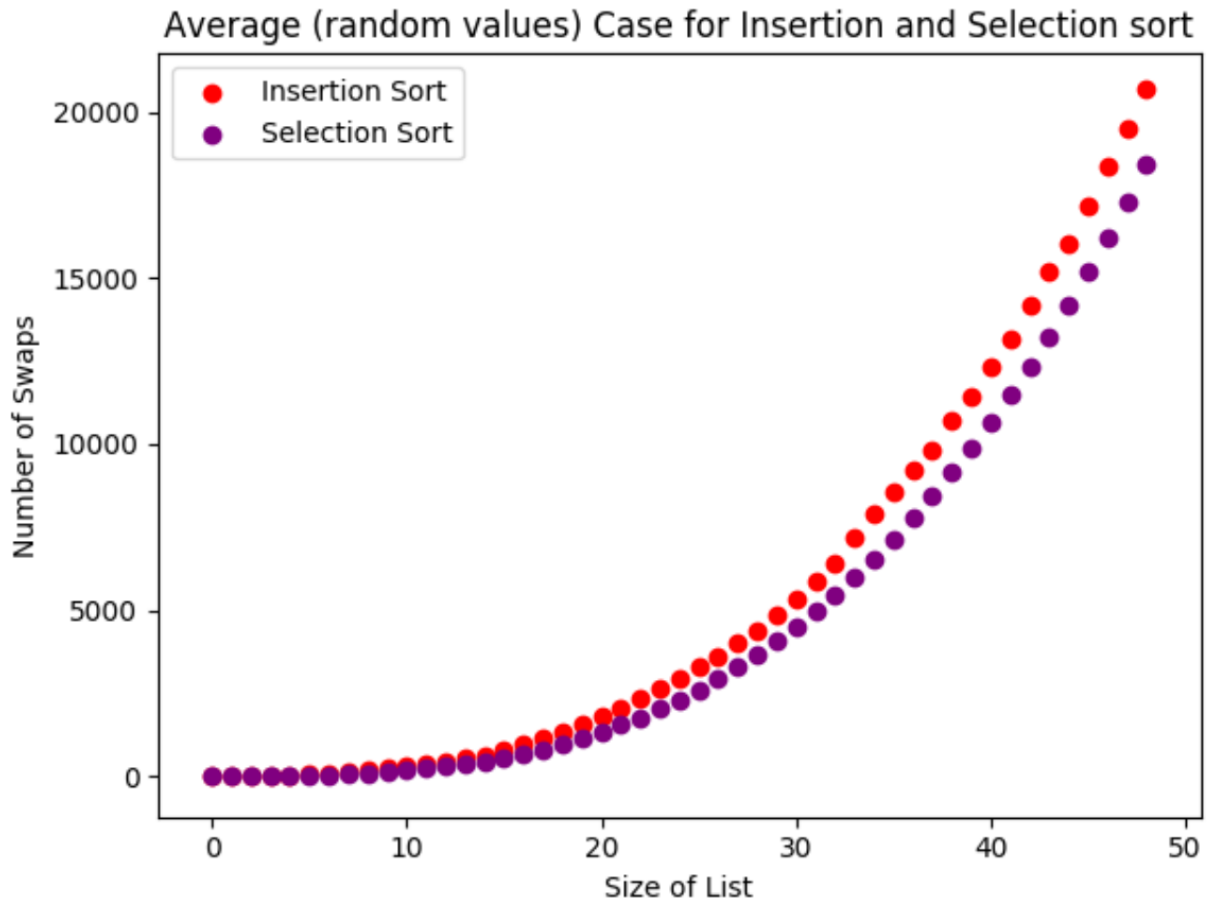
- The growth rate of the Decrease-by-Constant algorithm is logarithmic which means the algorithm decreases the input(n) by half, $\lfloor n/2 \rfloor$, at every recursive call. There are also instances when the number of multiplication operations at k is less than the number of operations at k-1 because there is one more multiplication operation occurring when n is odd. An example of this occurring is when k=12 and k-1=11. The complexity is $\theta(\log n)$.

- The graph of exponentiation has a staircase pattern, and each "step" has a length of 2^n. In this case, we have a step that starts at n=2 with a length of 2, n=4 with a length of 4, n=8 with a length of 8, etc. This pattern occurs when n is an exponent of 2. The graph increases linearly until the next exponent of 2. At each step the graph grows linearly until the next step, or exponent of 2. The complexity of this algorithm is $\theta(n^2)$.

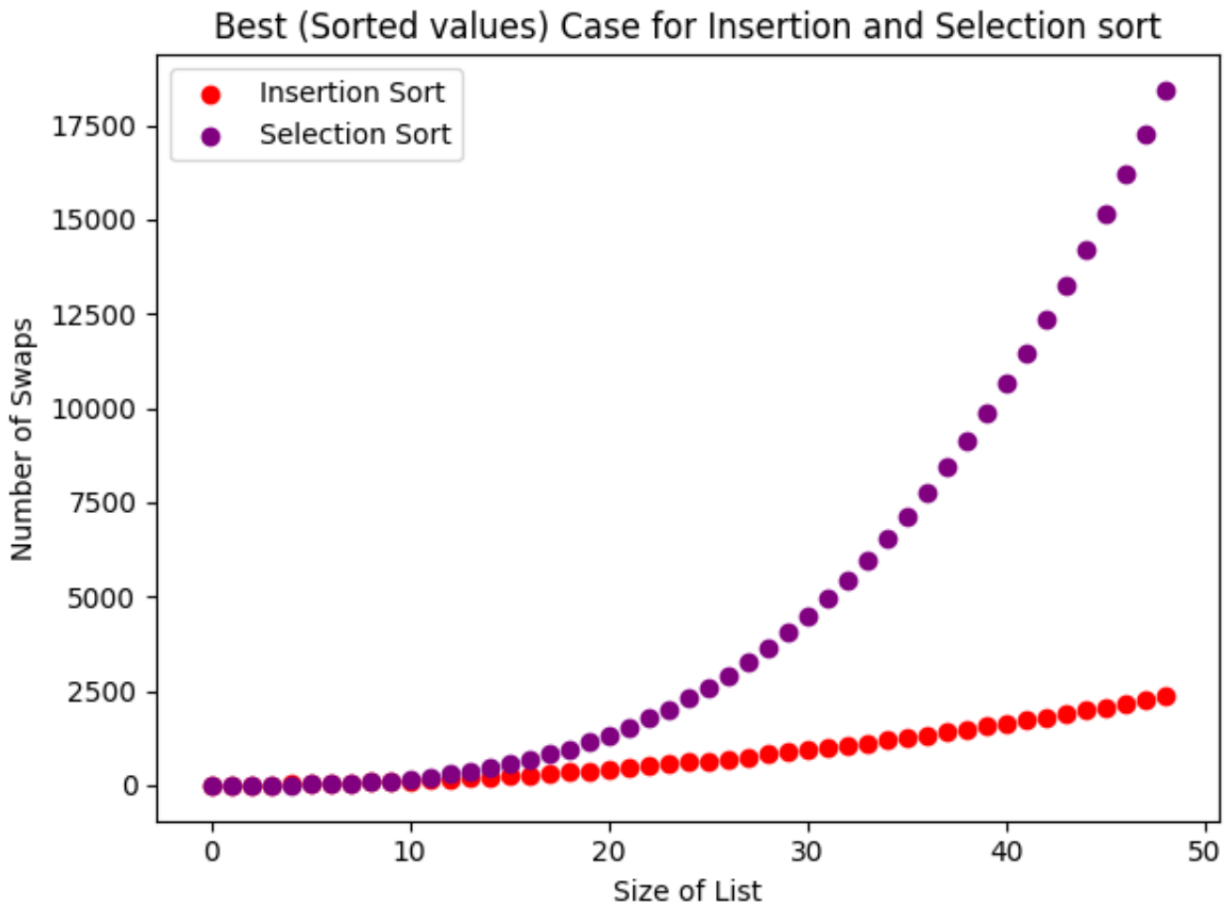
Task 3



Both the Selection Sort and Insertion Sort algorithms have the same curve in this case. The two algorithms make the same amount of comparisons at different sizes of n. Selection Sort compares each element of an array with every element of the same array, and insertion sort has to traverse the array backwards every iteration. This leaves both algorithms with a complexity of $\theta(n^2)$ in the worst case.



For the average case, Selection Sort and Insertion Sort vary in growth rate depending on the order of input. The complexity for the average case is $\theta(n^2)$.



In the best case, an already sorted array, Insertion Sort compares the elements only n times because the algorithm never has to traverse the array backwards. This makes the complexity $\theta(n)$ in the best case. On the other hand, Selection Sort still compares each element with all the elements in the array. The complexity is still $\theta(n^2)$ in the best case.