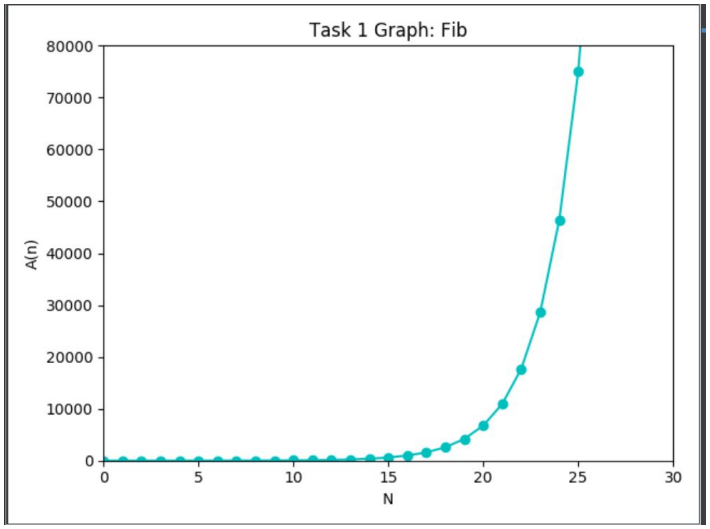
Alondra Lona
Alex Barajas-Ritchie
CS 415 Spring20

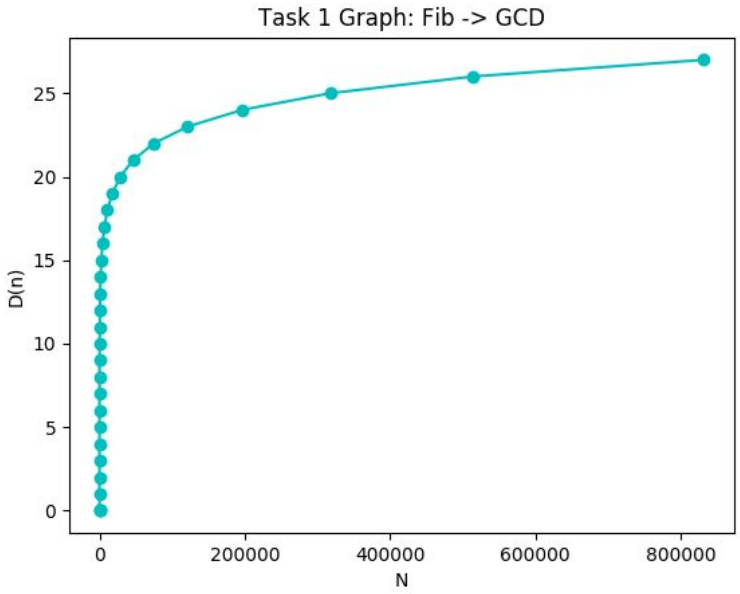
Project 1 Report

Task 1

i) For fibonacci, we implemented the code recursively to compute the kth integer in the sequence, and then counted the number of additions that were performed by the algorithm. The asymptotic complexity for fibonacci is Φ^k . We decided to have a large range in order to see how massive it can grow with even the biggest of values.

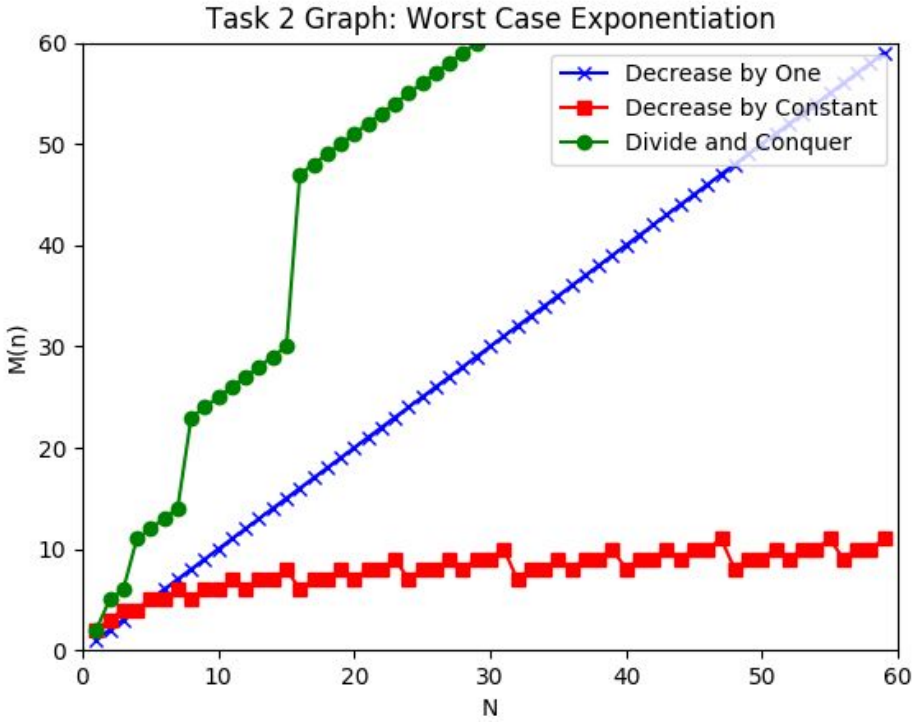


ii) For calculating the asymptotic complexity of Euclid's algorithm with the fibonacci sequence, we determined that when the fibonacci sequence is applied to Euclid's algorithm the worst case complexity occurs. N is the fibonacci sequence. Y is the amount of modular divisions.



Task 2

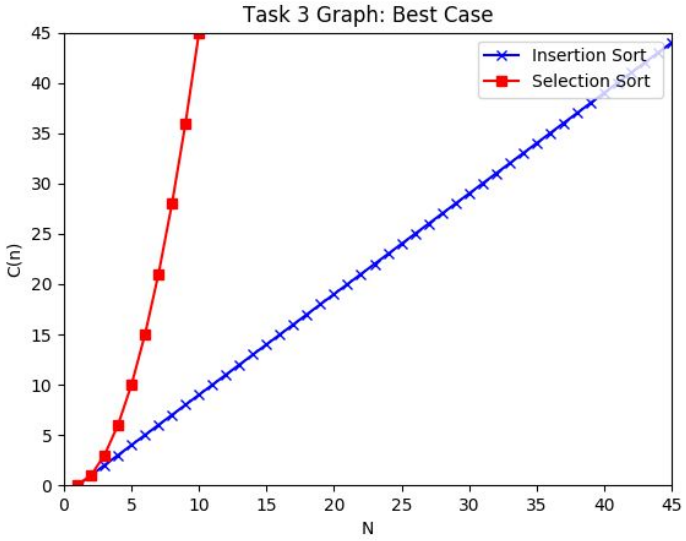
In the graph we can see three exponential functions being plotted at their worst time complexity. All three functions complete the same task but with many different complexities. Decrease by one - n, decrease by a constant - Logn, Divide and Conquer - n^2. We can use this knowledge to implement the best exponential function for our program's needs. For the three functions to compute a^n we chose 5 for a and values from 0 – 60 for n. We believed these sets of number best illustrated the complexity of all the algorithms comparatively.



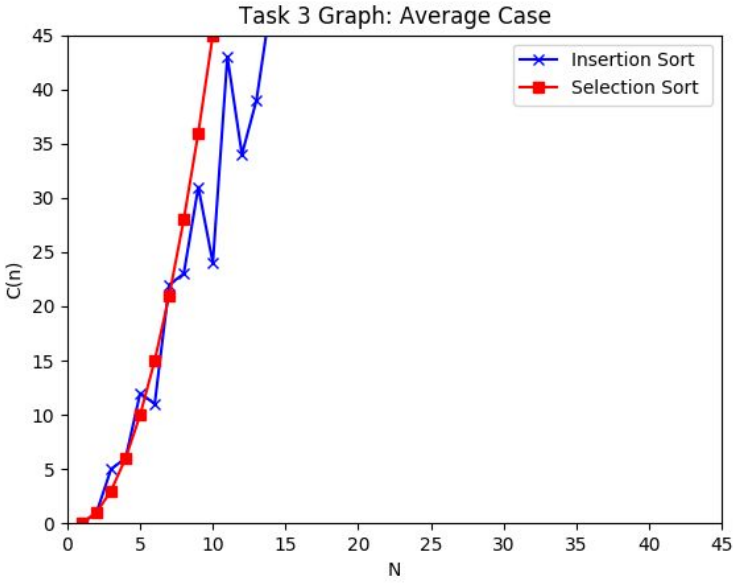
Task 3

For Insertion Sort, the complexities for best case, average case, and worst case are O(n), O(n^2), O(n^2) respectively. For selection sort, the complexities for best case, average case, and worst case are O(n^2), O(n^2), O(n^2) respectively. We chose values that we would be able to see a clear growth rate with and decided to create 60 different size arrays (of size 1-60) filled with random values. We chose our x-axis to have a range of 0-45 because this demonstrates a clear growth rate, one that could be easily distinguishable for all three cases. The lists that we fed the cases were all random reverse order, sorted array, and a non sorted array for worst case, best case and average case respectively.

In our best case graph, the best case is that the values are already sorted therefore insertion sort is linear and it's complexity is O(n) and we see that displayed as the values grow. In the same graph, selection sort grows exponentially unlike linear growth thus resulting in an asymptotic complexity of O(n^2).



Our average case graph shows both sorts being O(n^2). Our average case is that we have to sort the array and make n amount of comparisons. There will still be a large amount of comparisons made for both. Selection sort in contrast with insertion sort is a much more stable algorithm. The graph demonstrates various spikes in the insertion sort and that makes that one less stable even though they share the same asymptotic complexity.



The worst case is the arrays are reversed and then sorted, giving a complexity of O(n^2) for both insertion and selection sort. Even though we know insertion sort can have an advantage, since the values are reversed, it won't be as efficient and neither will selection sort. They both have to do a large amount of comparisons.