Matthew Wisler

ECE 368

Project 1

My sequence 2 generation was straight forward and my code simply divides by 1.3 until the input is at 1 and as it does so it stores this in an array to be referenced by my bubble sort for sequence one however my code generates the first 300 terms of sequence one then returns the part of the array needed. Because the sequence begins to grow so fast, the 300^{th} term would require a very large size that will not be handled by these functions. My shell sort function is the standard shell sort code. My improved bubble sort applies the standard bubble sort code, however it uses the gap that is picked for that iteration to increment for the function in place of incrementing only by 1 like in the original bubble sort. Unfortunately my improved bubble sort still took a very long time to sort large input cases

The time and space complexity of my sequence one generation are both O(n) because the function allocates an array to output of size n and it also must transfer the already calculated 3-smooth numbers into that array of size n. My sequence 2 generation has a space complexity of O(n) because it must allocate an array of size n to return. The time complexity of sequence 2 is O(logn) because as the input size grows the change decreases.

Shell Sort

Input	Comparisons	Moves	Time (s)
1,000	4311	66221	0
10,000	64818	1166240	0
100,000	878713	18089535	.04
1,000,000	11710260	259684562	.48

Improved Bubble sort

Input	Comparisons	Moves	Time(s)
1,000	NA	NA	.01
10,000	NA	NA	.18
100,000	NA	NA	18.11
1,000,000	NA	NA	NA

Unfortunately my comparisons and move counters stopped working shortly before submission.

Both of my sorting algorithms store their sequences in an array that is length of the sequence from the function called to generate their sequence, and this is the only additional space that I use in my sorting algorithms. Because of this the additional space needed is O(n).