PROJECT 1 REPORT (PHASE 2)

Name: Seung Yup Yum

Student ID Number: 76777057

UCI Net ID: syyum

Write up

Basic Explanation

The algorithms implemented in this project are largely divided into four categories: Insertion Sort, Merge Sort, Shell Sort, and Hybrid Sort. As the name suggests, all four are to sort one list in ascending or descending order, and in this project, it is unified to sort one list in ascending order.

Insertion Sort

Insertion sort is simplest one. For this project, it starts from the beginning, assume the first element is already sorted, go to the left. If it finds an element that is smaller than left element, it swaps until it found the right position.

Merge Sort

It involves divide-and-conquer. It keeps splitting half until the size of the list is 0 or 1, and it merges. The key idea of merge sort is that the list that contains 0 or 1 element is already sorted, and merging 2 sorted lists is much easier than sorting whole list.

Shell Sort

Shell Sort requires sequence. Due to the sequence, it has 4 different types, but the key concept of the shell sort is same. Simply, shell sort is completing insertion sort that is gap apart. If the gap is 2, it does insertion sort for all odd index elements, and it does insertion sort for all even index elements. The last gap of sequence must be 1, since it needs to do basic insertion sort step at the last iteration

Shell Sort 1

The sequence is most common one.

$$Gap\ Sequence = \frac{n}{2^k}\ where\ k = 1, 2, 3\ \cdots \log_2 n\ , where\ n = size\ of\ the\ input$$

It is equal to keep dividing 2.

Shell Sort 2

The sequence is

$$Gap\ Sequence = 2^k + 1\ where\ k = \log_2 n\ \cdots 3, 2, 1.\ 1\ is\ added\ as\ last\ element.$$

$$where\ n = size\ of\ the\ input$$

Since $\log_2 100{,}000 = 16.61$, and it is already descending order, it does not need to generate the actual sequence before starting sort. Also, for small size input, the first value from gap sequence equation gives larger value than the size of input. However, if the size is large enough, the first value from the gap sequence equation is less than the size of the input. Therefore, from $2^{\log_2 n} + 1$, calculate the first value of gap by keep taking out 1 from the exponent until it reaches to the largest number that is smaller than the size of the input.

Shell Sort 3

The sequence is

$$Gap\ Sequence = all\ 2^i 3^j \leq\ size\ of\ the\ list$$

At first, it is hard to achieve a fast way to generate sequence. After some research, an article is found that is referenced. Reference: https://stackoverflow.com/questions/25344313/generating-ascending-sequence-2p3q. Since the generating sequence algorithm is linear and the taken time to generate sequence is negligible, It makes actual list called gap sequence at the beginning before starting sort.

Shell Sort 4

The sequence is

$$Gap\ Sequence = all\ \frac{3^n - 1}{2} \leq\ n, where\ n = size\ of\ the\ input$$

Since the gap between gap sequence is huge, it is too much time waste to calculate descending order. For example, n = 100,000, the first gap value is $\frac{3^{11}}{2} = 88573$. If 100,000 is directly put it in to the equation and trying to find by taking out 1, it has to evaluate from $3^{99{,}999}$ to 3^{11}. Therefore, it starts from the 1, calculates the gap sequence and generates the list of gap sequence.

Hybrid Sort

For this project, Hybrid Sort is a combination of Merge Sort and Insertion Sort. The whole process is the same as merge sort, but during the dividing process, when the list size reaches to the value ***H,*** it stops the merge sorting, it does insertion sort instead. There are 3 different types of Hybrid Sort in this project, but the only difference between them is ***H*** value.

Hybrid Sort 1

$$H = n^{0.25}\ where\ n = size\ of\ the\ input$$

Hybrid Sort 2

$$H = n^{0.5}\ where\ n = size\ of\ the\ input$$

Hybrid Sort 3

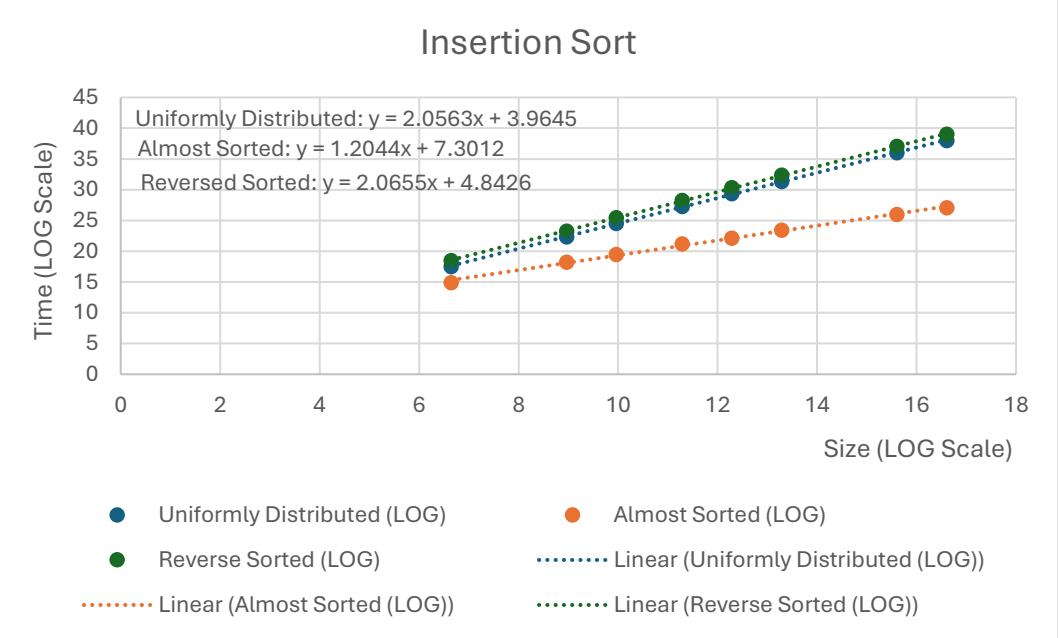
$$H = n^{0.75}\ where\ n = size\ of\ the\ input$$

Constraints

- The used function to measure time is ***time.process_time_ns().*** It measures the CPU time of execution of function.
- The iteration is 5 times, and each time, the input is newly generated.
- The size of input is ***[100, 500, 1000, 2500, 5000, 10000, 50000, 100000].***
- Each input has 3 types: Uniformly Distributed, Almost Sorted, Reversed Sorted.
- Uniformly Distributed is made by using Fisher-Yates algorithm.
- Almost Sorted is made by using algorithm that is stated in Canvas.
- Reversed Sorted is made by using ***list.reverse()*** function.
- Shell sort follows the example in lecture slide. If the first swap happens, then it keeps checking all the way down and swap if the condition is violated.
- The plot that will be shown below is already log scale. After get the average time for each input, ***LOG(cell number,2)*** is used in EXECEL to generate log base 2 number.
- The program used to store data and plotting is EXECEL. It helps to make trendline effectively to see and generate slope and linear equations.

Plotting Insertion Sort and Merge Sort

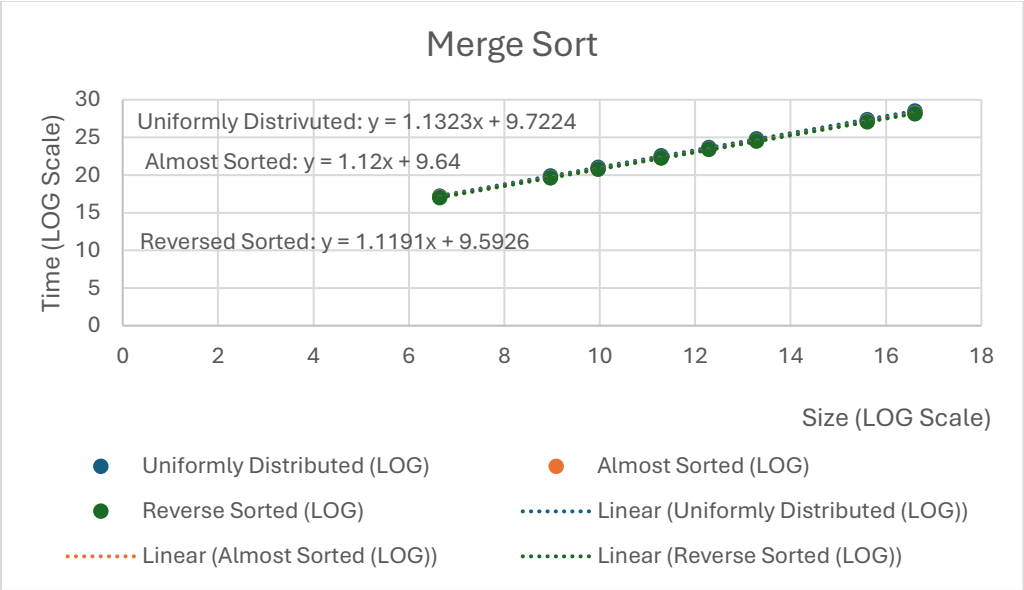
Insertion Sort



Based on the result of the experiment,

- The slope of the asymptotic best fit line for Uniformly Distributed Input is 2.0563.
- The slope of the asymptotic best fit line for Almost Sorted Input is 1.2044.
- The slope of the asymptotic best fit line for Reversed Sorted Input is 2.0655 .

Merge Sort

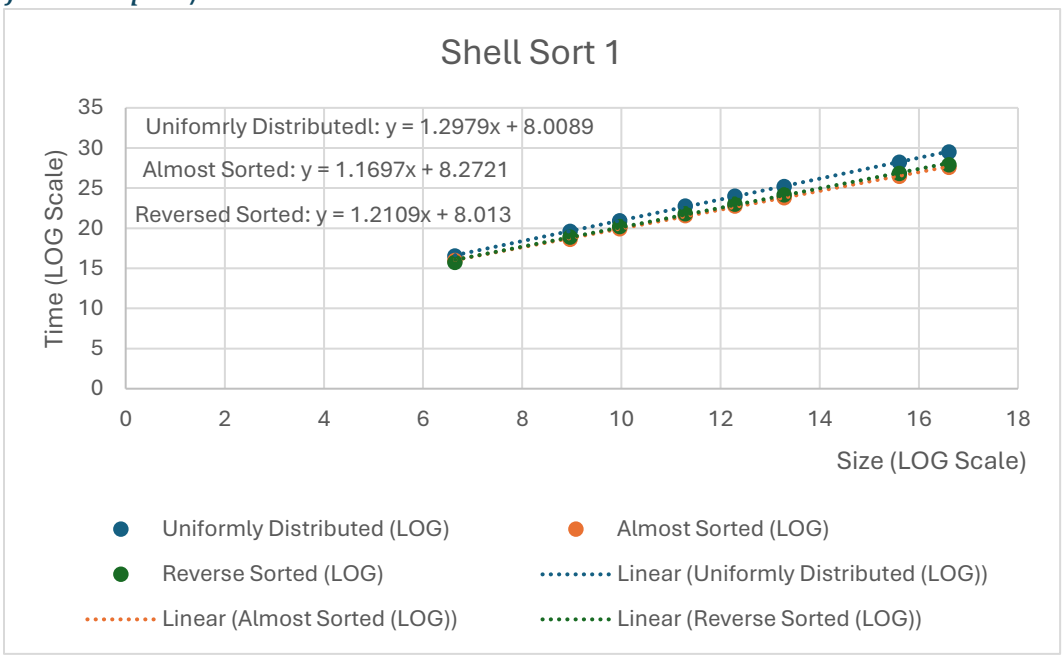


Based on the result of the experiment,

- The slope of the asymptotic best fit line for Uniformly Distributed Input is 1.1323.
- The slope of the asymptotic best fit line for Almost Sorted Input is 1.12.
- The slope of the asymptotic best fit line for Reversed Sorted Input is 1.1191.

Plotting 4 Shell Sort

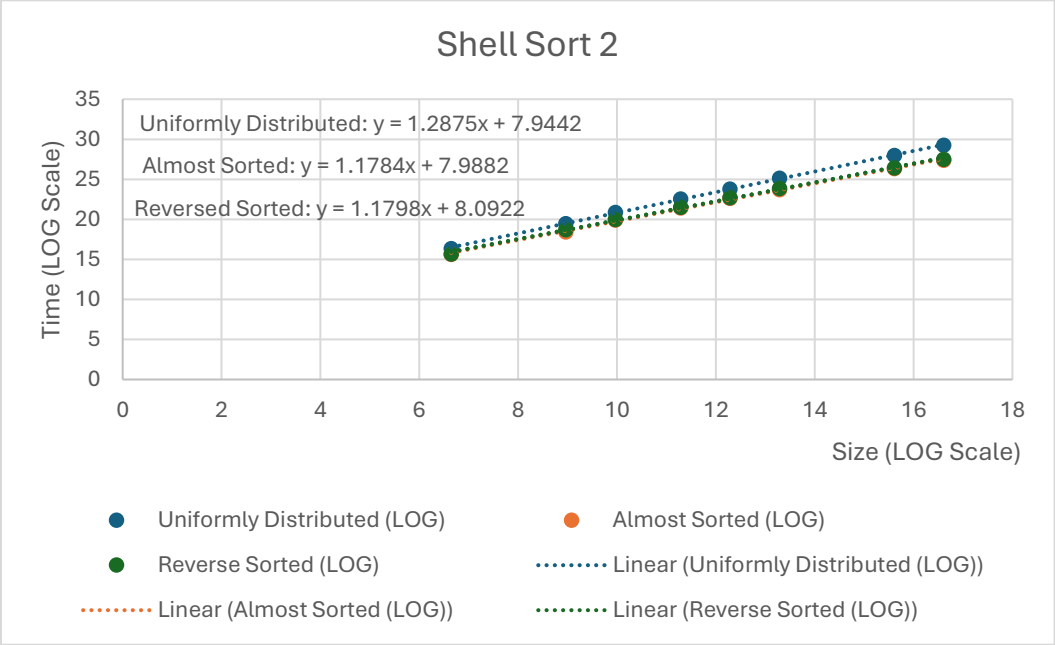
Shell Sort 1 ($Gap\ Sequence = \frac{n}{2^k}\ where\ k = 1, 2, 3\ \cdots \log_2 n\ , where\ n = size\ of\ the\ input$)



Based on the result of the experiment,

- The slope of the asymptotic best fit line for Uniformly Distributed Input is 1.2979.
- The slope of the asymptotic best fit line for Almost Sorted Input is 1.1697.
- The slope of the asymptotic best fit line for Reversed Sorted Input is 1.2109.

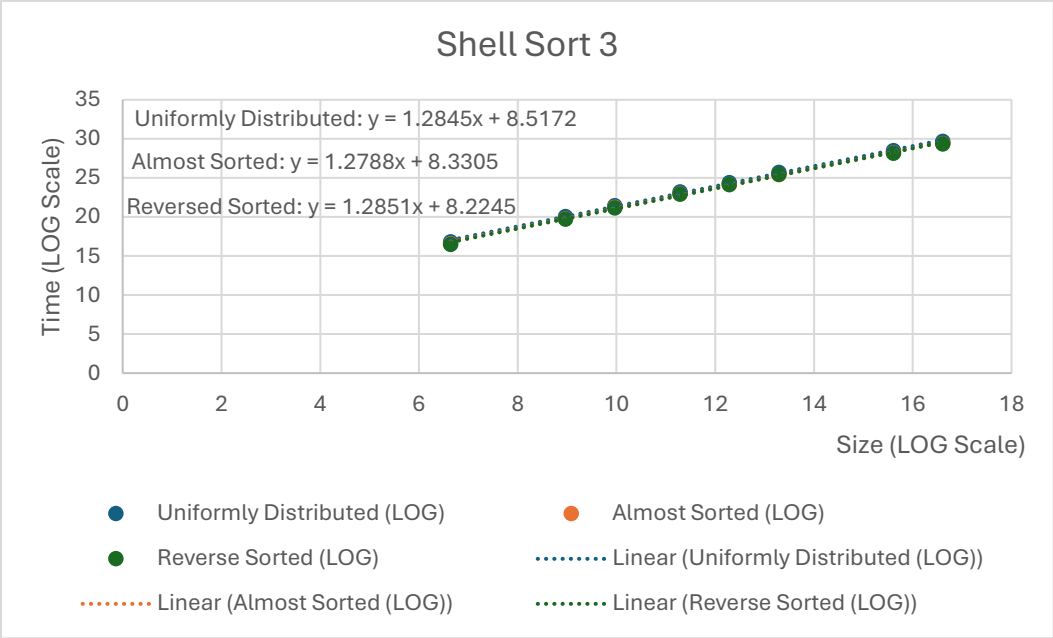
Shell Sort 2 ($Gap\ Sequence = 2^k + 1\ where\ k = \log_2 n\ \cdots 3, 2, 1.\ 1\ is\ added\ as\ last\ element.$)



Based on the result of the experiment,

- The slope of the asymptotic best fit line for Uniformly Distributed Input is 1.2875.
- The slope of the asymptotic best fit line for Almost Sorted Input is 1.1784.
- The slope of the asymptotic best fit line for Reversed Sorted Input is 1.1798.

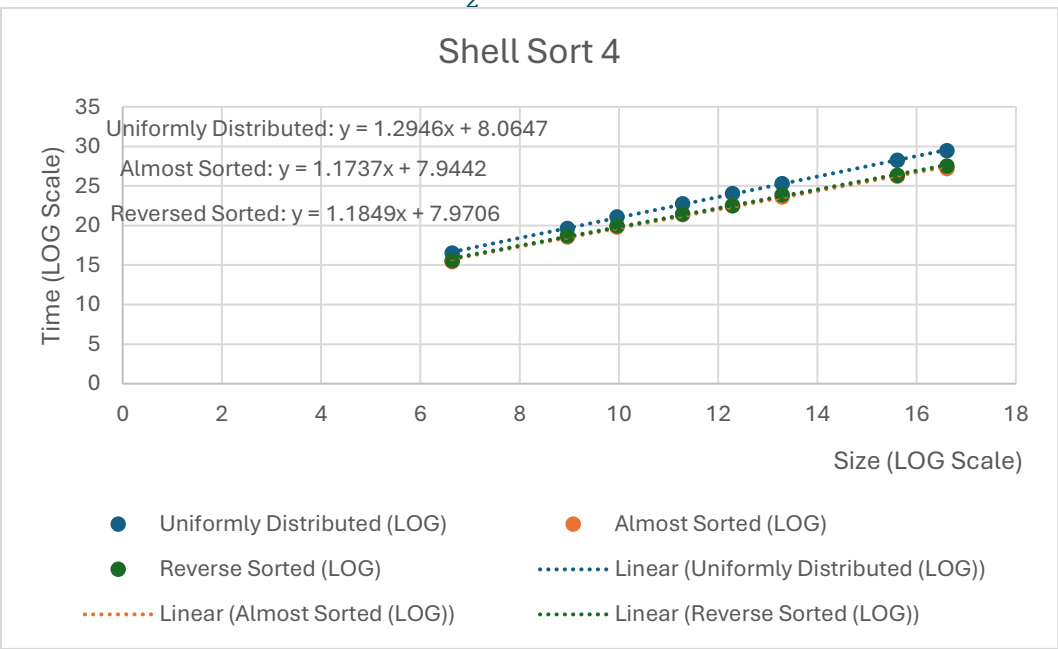
Shell Sort 3 ($Gap\ Sequence = all\ 2^i 3^j \leq\ size\ of\ the\ list$)



Based on the result of the experiment,

- The slope of the asymptotic best fit line for Uniformly Distributed Input is 1.2845.
- The slope of the asymptotic best fit line for Almost Sorted Input is 1.2788.
- The slope of the asymptotic best fit line for Reversed Sorted Input is 1.2851.

Shell Sort 4 ($Gap\ Sequence = all\ \frac{3^n - 1}{2} \leq\ n, where\ n = size\ of\ the\ input$)

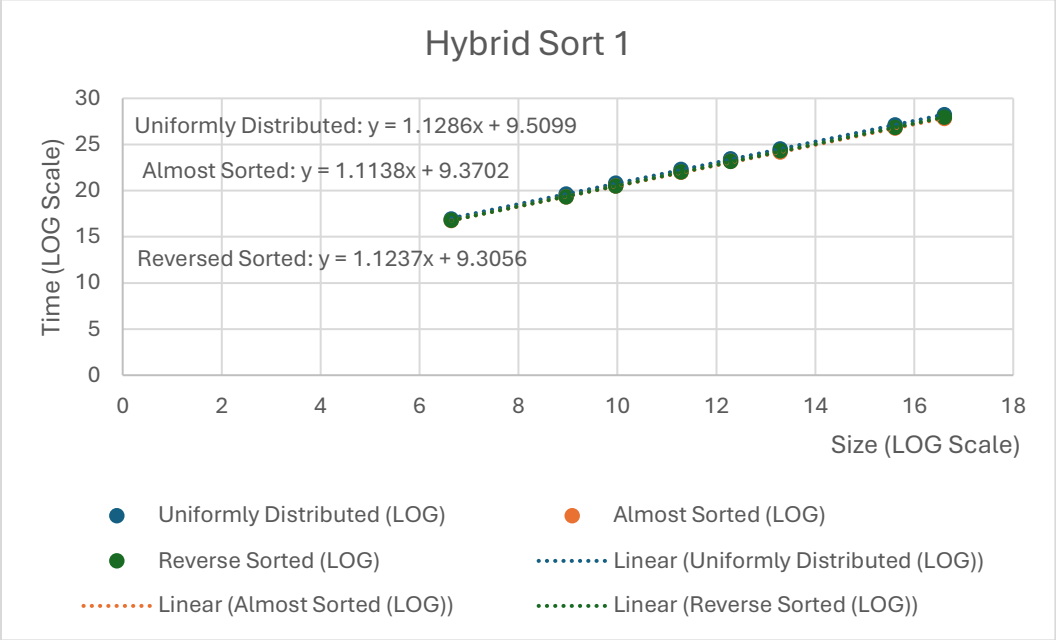


Based on the result of the experiment,

- The slope of the asymptotic best fit line for Uniformly Distributed Input is 1.2946.
- The slope of the asymptotic best fit line for Almost Sorted Input is 1.1737.
- The slope of the asymptotic best fit line for Reversed Sorted Input is 1.1849.

Plotting 3 Hybrid Sort

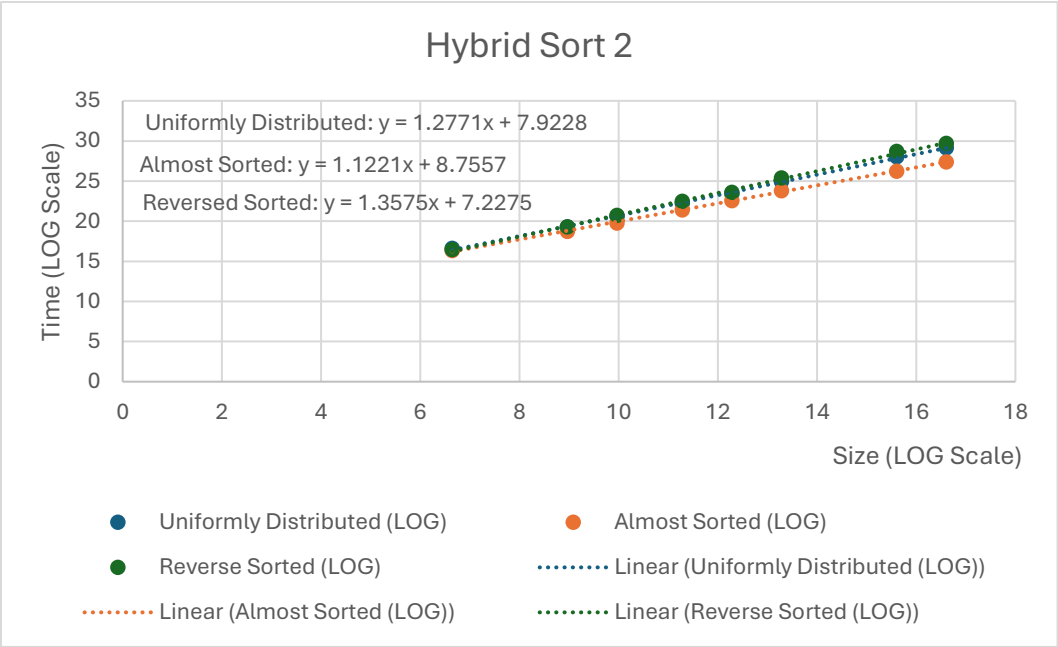
Hybrid Sort 1 ($H = n^{0.25}$)



Based on the result of the experiment,

- The slope of the asymptotic best fit line for Uniformly Distributed Input is 1.1286.
- The slope of the asymptotic best fit line for Almost Sorted Input is 1.1138.
- The slope of the asymptotic best fit line for Reversed Sorted Input is 1.1237.

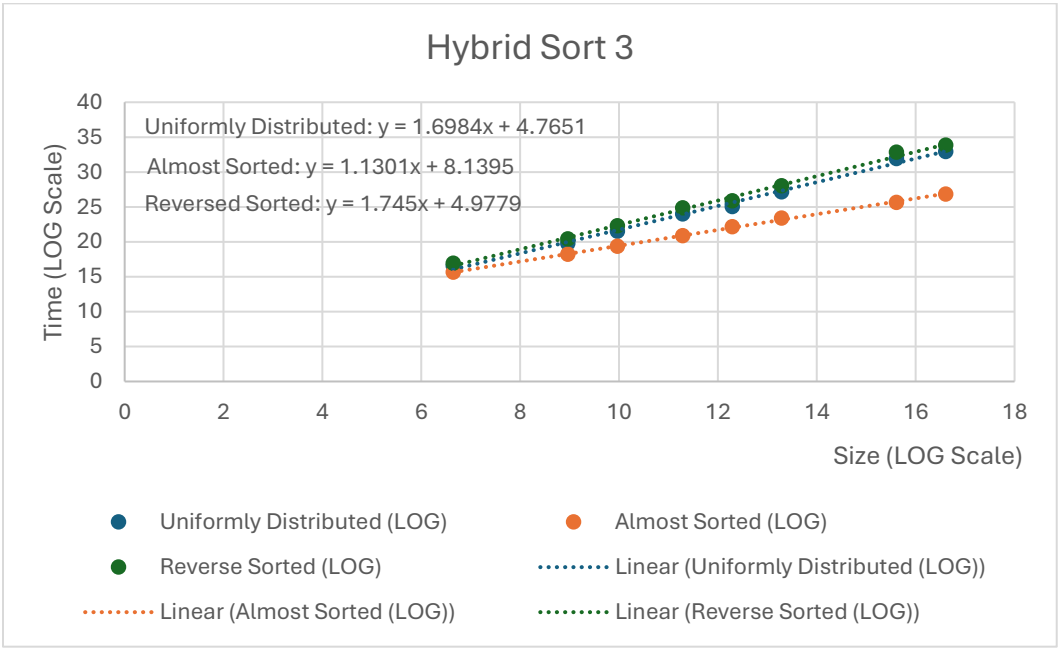
Hybrid Sort 2 ($H = n^{0.5}$)



Based on the result of the experiment,

- The slope of the asymptotic best fit line for Uniformly Distributed Input is 1.2771.
- The slope of the asymptotic best fit line for Almost Sorted Input is 1.1221.
- The slope of the asymptotic best fit line for Reversed Sorted Input is 1.3575.

Hybrid Sort 3 ($H = n^{0.75}$)



Based on the result of the experiment,

- The slope of the asymptotic best fit line for Uniformly Distributed Input is 1.6984.
- The slope of the asymptotic best fit line for Almost Sorted Input is 1.1301.
- The slope of the asymptotic best fit line for Reversed Sorted Input is 1.745.

Comparing Different Shell Sort to the Different Hybrid Sord

From the experiment, the results show that all shell sorts (shell sort 1 to 4) have similar running time to the hybrid sort 1 for all different types of input. The most efficient way to sort Almost Sorted input is Hybrid sort 1, and in general, all hybrid sort algorithm is slightly better than shell sort for almost sorted input. Furthermore, for uniformly distributed input, it is hard to say that all shell sort and hybrid sort 1 and 2 have different running time. The experiment results shows that hybrid sort 3 has the highest slope for sorting Reversed Sorted input because hybrid sort 3 starts normal insertion sort too early than other hybrid sort. Generally, the average performance of hybrid sort 1 is better than any other shell sorts or hybrid sorts in this experiment.

Performance difference based on the input distribution

The result shows that the one sorting algorithm that has very different running time for the different input distribution is insertion sort. The slope is 1.2044 for almost sorted input, but it is 2.0563 and 2.0655 for uniformly distributed and reversed sorted input. Also, the result plot of hybrid sort 3 shows that it also has quite different running time for the different input distribution. The slope for uniformly distribution and reversed sorted input is 1.6984 and 1.745, but when the input is almost sorted, the slope is 1.1301.

Sorting algorithms in this experiment that have similar running time for the different input distribution are Merge Sort, Shell Sort3, and Hybrid Sort 1. Merge Sort plot shows that the slope is 1.1323, 1.12, 1.1191 for uniformly distributed, almost sorted, reversed sorted input respectively. The slope of Shell Sort 1 is 1.1845, 1.2788, 1.2851, and the slope of Hybrid Sort 1 is 1.1286, 1.1138, 1.1237.

Best Sorting Algorithm

Personally, I think hybrid sort 1 is the best algorithm in this experiment. The reason that I do not select merge sort as the best sorting algorithm is memory. Merge Sort require $O(\log n)$ space for call stack. It can be seen that it might not be a big deal, but hybrid sort 1 shows pretty similar performance with less memory usage. Since shell sort's performance is not really stays in constant because they require good decision on the gap sequences, it seems hybrid sort 1 (H = 0.25) is best sorting algorithm in this experiment.

But the bottom of hybrid sort in this experiment is basically insertion sort which has the worst performance on reversed sorted input. Therefore, if we make hybrid sort with lower H (less than $n^{0.5}$) that combines with the shell sort would yield much better performance result for extremely large size of input. Since shell sort has better performance for various input distribution

than insertion sort, the hybrid sort that combines with merge sort and shell sort would have better performance than hybrid sort that combines with merge sort and insertion sort.