Abhimanyu Agarwal

Report

Objective: The objective was to understand and implement two different types of sorting algorithms that undergo a similar optimization based on two gap sequences and based on the obtained results, possibly analyze algorithm performance (See table).

Shell sort: The Shell sort is basically an optimization to Insertion Sort. Shell sort takes an array of n inputs and sorts it by breaking it down to logical sub-arrays. These sub-arrays are sorted based on the 'gap' that in general scenarios is empirically derived. However, with respect to the project the gap here is essentially determined based on an array of 'gap' sequences (Sequence 1). Space Complexity = $O(N)$ & Time Complexity of Sequence = $O(\max(\log_2 N * \log_3 N)) = O(\log(N)^2)$

Sequence 1= $\{2^p 3^q, \ldots, 2^p 3^q, \ldots, 16, 12, 9, 8, 6, 4, 3, 2, 1\}$

The shell sort would take the very last value of this array and perform insertion sort on each sub-array. The same process would be repeated for different and smaller 'gap' values until the gap value is one. Assume our 'gap' value is X, then our logical sub-arrays would look like:

$\{0, N, 2N, 3N, \ldots, XN\}$ where, $XN < \text{length(array)}$

$\{1, N + 1, 2N + 1, 3N + 1, \ldots, XN + 1\}$where, $XN + 1 < \text{length(array)}$

Once the gap value is 1, the shell sort is basically performing the insertion sort. By that point, the array should be sorted.

Bubble Sort: The Bubble Sort is optimized like the Shell Sort. The difference being that Sequence 2 would be used in order to determine the gap values. Notable aspect being, that each sub-array is sorted using bubble sort which essentially deals with exchange of elements. In Sequence 2, it is observed that gap value in the array is the floor value of the previous integer gap followed by a division by 1.3(G.P). Space Complexity = $O(N)$ & Time Complexity of Sequence = $O(\log_{1.3} N)$.

Sequence 2= $\{N_1 = N/1.3, N_2 = N_1/1.3, N_3 = N_2/1.3, \ldots, 1\}$

Using the 'gap' values we generate sub-arrays. When there are no comparisons made, it could be assumed that the sub-array is sorted.

Input	Shell_Insertion_Sort			Improved_Bubble_Sort		
(N)	Moves	Comparisons	Runtime(sec)	Moves	Comparisons	Runtime(sec)
1000	35266	35266	0.0 sec	23197	13455	0.0 sec
10000	615529	615529	0.01 sec	339913	189528	0.01 sec
100000	9484124	9484124	0.02 sec	4482925	2448540	0.02 sec
1000000	135697411	135697411	0.32 sec	55746098	30237996	0.14 sec

Analysis	Shell_Insertion_Sort	Improved_Bubble_Sort
Time Complexity	O(N*log(N))	O(N*log(N))
Space Complexity	O(1)	O(1)