Theoretical discussion of algorithm efficiency

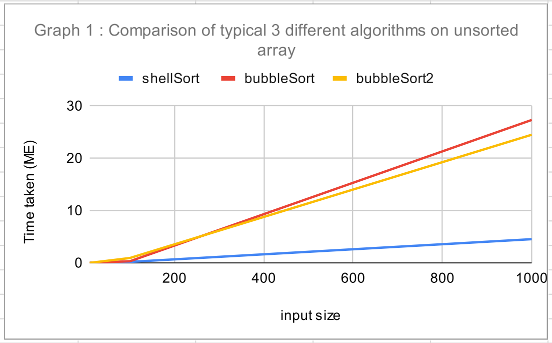
Section 1 : number of comparisons & swaps, and time taken to sort an unsorted/sorted array on 3 different algorithms

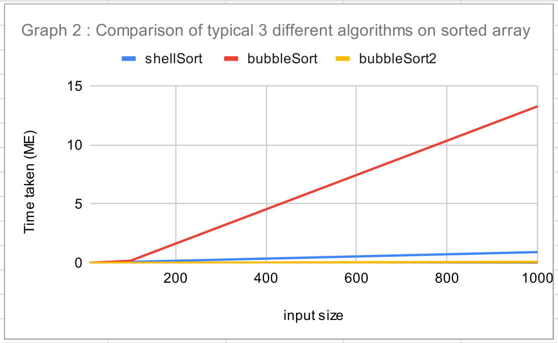
features /algorithm	size of 10				size of 100				size of 1000			
array size	shellSort	bubbleSort	bubbleSort2	min value	shellSort	bubbleSort	bubbleSort2	min value	shellSort	bubbleSort	bubbleSort2	min value
UNSORTED												
number of comparisons	53	45	45	45	2808	4950	4905	2808	55727	499500	499065	55727
number of swaps made	12	30	30	12	395	2382	2382	395	7455	260612	260612	7455
total time taken to sort (ME)	0.0208	0.0078	0.0076	0.0076	0.1978	0.3185	0.9358	0.1978	4.5337	27.3134	24.49	4.5337
				0				0				0
SORTED				0				0				0
number of comparisons	22	45	9	9	503	4950	99	99	8006	499500	999	999
number of swaps made	0	0	0	0	0	0	0	0	0	0	0	0
total time taken to sort	0.0036	0.0051	0.0019	0.0019	0.0893	0.184	0.0067	0.0067	0.9166	13.2808	0.0939	0.0939

Section 2 : An emphasis algorithm efficiency on 3 different algorithms

	time taken (ME)		
input size	shellSort	bubbleSort	bubbleSort2
10	0.0208	0.0078	0.0076
100	0.1978	0.3185	0.9358
1000	4.5337	27.3134	24.49

	time taken (ME)		
input size	shellSort	bubbleSort	bubbleSort2
10	0.0036	0.0051	0.0019
100	0.0893	0.184	0.0067
1000	0.9166	13.2808	0.0939


Graph 1 : Comparison of typical 3 different algorithms on unsorted array


Graph 2 : Comparison of typical 3 different algorithms on sorted array

Section 3 : Theoretical discussion of algorithm efficiency

Discussion :

We can make a few observations from the data obtained. Our analysis will focus on unsorted arrays.

- When input size increases enormously, Shell Sort does not increase its time largely to sort the array (Graph 1). However, original and modified bubble sort 2 increases largely. This is due to the fact that Shell Sort swaps indexes that are far apart, while Bubble sort swaps 2 items at once that are adjacent together. This feature allowed elements in the array to get into its valid position quicker than adjacent comparisons. We can verify this by checking the data from section 1. As the size of the array increases, Shell sort's number of comparisons and swaps does not increase largely compared to BubbleSort 1 and 2.
- Suppose we compare graph 1 with a graph of a typical growth function with large values of n. From graph 1, shellSort's time complexity is approximately O(nlogn), while BubbleSort 1 and 2 is approximately $O(n^2)$.
- From graph 1, since the lines that represent original and modified bubbles are almost similar, we can conclude their time complexity is also similar.
- Thus, the efficiency of these algorithms can be ranked as follows : shellSort > bubbleSort ≈ bubbleSort2.