Jhakon Pappoe

IT334

Big-O Analysis

Algorithm A also known as Timsort is derived from merge sort and insertions sort. The graph shows that it is growing in a linear form. This means that it is adaptiple, the line does not dip or jump at all. The notation for this algorithm is O(n log (n)) which is what the algorithm will cost. This sort is impressively fast, the algorithm is close to Algorithm C runtime. I sorted the integer array from the start and looped until the index matches the index one slot before it.

Algorithm B uses ArrayList which is a dynamic array that allows to move through indexes. This algorithm costs O(n^2) time. To add/remove has O(n) for such operations. Compared to the previous algorithm it is much slower. As more complexity is added the algorithm is overloaded and becomes slow with large amounts of data. The nested loop has the contains method checks for same indices.

Algorithm C uses TreeSet which is used for linearithmic and has sorted elements that is a great advantage. For add, remove, contains operations the algorithm costs O(log(n)). This is a very efficient algorithm after reporting from my data. This would be closely comparable with algorithm D. The foreach loop adds elements.

Algorithm D uses Hashset, the operations add/remove/contains of HashSet costs **O(n) linear** time. This algorithm is comparatively faster than any of the other algorithms. My data represents the slight increase in load overtime making it the fastest algorithm. As the previous algorithm there is a foreach loop that adds elements.

Algorithm E is relatively a slow algorithm just as Algorithm B this costs O(n^2) quadratic which is faster than the Arraylist but slower than the others. I looped through and compared the indices to the other indices.