**Raw Sort Run Time Data:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Run Times | |  |  |  |  |  |  |  |
|  | Bubble |  |  | Insertion |  |  | Selection |  |  |
| Array Size | Ascending | Descending | Random | Ascending | Descending | Random | Ascending | Descending | Random |
| 25 | 0 | 19.67 | 11.33 | 0 | 4.67 | 3.67 | 4.3 | 5.3 | 5 |
| 50 | 0.33 | 95.33 | 72.67 | 1 | 18 | 13.67 | 15 | 16.33 | 17.33 |
| 100 | 1 | 247.33 | 187 | 2 | 72 | 47 | 60.33 | 76.33 | 63.33 |
| 200 | 2.33 | 386.67 | 440.33 | 4 | 335.67 | 180 | 229.33 | 248.33 | 255 |
| 1000 | 12.33 | 1323 | 1219.67 | 21 | 818 | 217 | 831.67 | 863.33 | 262.67 |

Bubble Sort:

Bubble Sort Takes much less time for an ascending list, and about the same for a descending and randomly sorted list. This is because when doing an ascending sort it iterates through the list checking each element, making no swaps, then reaches the end of the list and realises it's done, so ascending is very efficient. For descending it always takes N passes, and for random it normally takes pretty close to N passes, so it generally takes roughly the same time for random and descending, although in the long run random will take slightly less time.

Insertion Sort

Insertion sort takes very little time for an ascending list, this is because it only takes one comparison to sort each list, discovering each new element is now the largest and inserting at the end of the list. Random sorting is much slower than ascending, but much faster than descending, since it saves a lot of operations when the next entry is nearer the end of the list, while in descending the item to be sorted is compared against all previously sorted entries.

Selection Sort:

Selection Sort takes roughly the same time for ascending, descending and nearly sorted list, this is because it always searches through the entire list for the smallest item. For some unknown reason, sorting a 1000 element randomly sorted list takes much less time than an ascending or descending 1000 element list, and roughly the same as a 200 element randomly sorted list.

Comparison:

All three algorithms have a pair of nested for loops that iterate through the list, hence a time complexity of O(n^2) and the time they take will grow quadratically.

Data from 200 element lists were analysed due to the outlier in 1000 element random selection

Bubble sort and insertion sort are very efficient at sorting ascending lists, both we're however less efficient than selection at sorting descending lists.

Of these algorithms bubble is the worst, insertion sort should (pretty much) always be used instead of bubble since it performs as well or better in all categories. Insertion sort should probably be used almost all the time, unless you're expecting to sort mostly backwards lists, at which point it may make sense to just invert the list to start with, or you desire the runtime consistency of selection.