**Table 1:** (Prototype for integers 1-16):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Ordered Data | | Reverse Order | | Random Order | |
|  | Comparisons | Exchanges | Comparisons | Exchanges | Comparisons | Exchanges |
| Bubble Sort | 15 | 0 | 120 | 120 | 120 | 66 |
| Selection Sort | 120 | 15 | 120 | 15 | 120 | 15 |
| Insertion Sort | 0 | 0 | 120 | 120 | 66 | 66 |

**Table 2:** (Large Data – Integers 1-2000):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Ordered Data | | Reverse Order | | Random Order | |
|  | Comparisons | Exchanges | Comparisons | Exchanges | Comparisons | Exchanges |
| Bubble Sort | 1999 | 0 | 1999000 | 1999000 | 1998724 | 974006 |
| Selection Sort | 1999000 | 1999 | 1999000 | 1999 | 1999000 | 1999 |
| Insertion Sort | 0 | 0 | 1999000 | 1999000 | 974006 | 974006 |

**Report:**

Since bubble sort is sensitive to data, when running against the best case (ordered data), the algorithm makes n-1 comparisons and no exchanges (since the Boolean condition determined that no exchanges need to be made in an already ordered data set). This gives bubble sort a best running efficiency of O(n). Also, in the worst case, bubble sort will exchange n-1 times for the first iteration of the outer loop, n -2 times for the second loop, and so on, to get a total of (n\*(n-1)/2) exchanges. The same goes for the number of comparisons in the worst case, with the algorithm achieving (n\*(n-1)/2) comparisons – in all, this means a worst runtime efficiency of O(n2). In the average case, bubble sort makes < n^2 swaps and <n^2 number of comparisons, landing at an average efficiency of O(n2).

For selection sort, the algorithm is **not** sensitive to input. This means that regardless of the case, selection sort will always execute n-1 exchanges (O(n) efficiency) and (n\*(n-1)/2) comparisons (O(n2) efficiency). Overall, the algorithm runs at an O(n2) efficiency for the best, worst, and average sorting scenarios.

Since insertion sort is sensitive to data, when running against the best case (ordered data), the algorithm makes no comparisons and no exchanges. This is because in the best case, the inner loop will never run, which represents zero comparisons. If no comparisons are made, this means no swaps happened either (again, the data is already in order). Overall, this equates to a best time complexity of O(n). In the worst case, insertion sort makes (n\*(n-1)/2) comparisons (the maximum because the data is in reverse order) and (n\*(n-1)/2) exchanges, which equates to an O(N)^2 efficiency. The average time complexity of insertion sort is O(N)^2. This is because on average, the bad and good cases tend to cancel each other out, which leaves the algorithm to examine about half the number of items in the list. From my data, is shown by ~ n2/4 comparisons (about half as many as selection sort) and exchanges.