**HW #8 – Darin Ellis – 12/2**

Tables of comparison results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N = 100 | backwards | sorted | swapped pairs | random |
| Selection | 4998 | 4949 | 4998 | 5045 |
| Insertion | 4949 | 98 | 147 | 2670 |
| Quicksort (rand) | 670 | 593 | 657 | 597 |
| Quicksort (1st) | 4949 | 4949 | 2499 | 764 |
| Mergesort | 664 | 664 | 664 | 664 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N = 1000 | backwards | sorted | swapped pairs | random |
| Selection | 499998 | 499499 | 499998 | 500488 |
| Insertion | 499499 | 998 | 1497 | 245246 |
| Quicksort (rand) | 10884 | 10744 | 10942 | 11310 |
| Quicksort (1st) | 499499 | 499499 | 249999 | 13601 |
| Mergesort | 9965 | 9965 | 9965 | 9965 |

Observing the patterns above, I would argue that:

* Selection sort is (n2). It takes essentially the same regardless of input, and grows around a power of 2.
* Insertion sort is Ω(n) and O(n2). In the worst case (backwards), it takes as long as selection sort. In the best case (already sorted), it takes almost exactly n comparisons.
* Quicksort (with a random pivot) is approximately (nlogn). In all cases it grows slightly faster than linear time, but far from a square.
* Quicksort (with a pivot as the first element) is Ω(nlogn) and O(n2). In the worst cases (and even when fully sorted already!) it takes time identical to selection sort. In the other case (random) it performs nearly identical to the other Quicksort.
* Mergesort is (nlogn). It performs the same no matter the input and grows slightly faster than linear.

This behavior appears to match what we discussed in class exactly! Selection sort is generally very slow. Insertion sort can be very fast in perfect situations, but in general, as n increases, it performs like selection sort. Quicksort is generally rather fast, but has the potential to peform very poorly in some situations. Mergesort is universally fast and rather unaffected by the input.