**Analysis**

\*Testing was done using eclipse\*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sorted  Test | N = 100 | N = 1000 | N = 10000 | N = 100000 | N = 1000000 |
| Heap Sort | 370,049ns | 16,324,841ns | 25,764,759ns | 76,530,356ns | 631,819,357ns |
| Insertion Sort | 35,353ns | 83,001ns | 883,047ns | 10,677,644ns | 62,831,215ns |
| Merge Sort | 170,230ns | 1,517,473ns | 41,490,528ns | 117,686,955ns | 589,642,527ns |
| Quick Sort | 311,257ns | 800,429ns | 15,706,939ns | 46,831,848ns | 343,580,982ns |

Looking at the data for sorting the files that are already sorted, insertion sort was the fastest

out of all the sorting methods. Comparing this results to the theoretical analysis, which has insertion to

have a best case of O(n) compared to the other three's best case which is O(nlogn), it is true that

insertion should be the fastest since it is just linearly going through the array and just comparing each

data and no swapping has to be done. The data for heap, merge and quick sort looks a little bit weird

since looking at their theoretical analysis they should be about to same for their best case. Maybe this is

due to the machine being use, but I am unable to test this because I only have one machine. It could

also maybe due to the way the methods are written that is causing the discrepancies. There could be a

step in each of those methods that could cause a slower performance, or cause it to run better.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unsorted Test | N = 100 | N = 1000 | N = 10000 | N = 100000 | N = 1000000 |
| Heap Sort | 382,731ns | 16,085,058ns | 24,908,612ns | 100,887,158ns | 1,725,577,583ns |
| Insertion Sort | 1,415,258ns | 17,210,578ns | 348,651,394ns | 31,759,398,513ns | 9,825,866,651,834ns |
| Merge Sort | 207,504ns | 1,499,413ns | 43,591,320ns | 127,619,888ns | 1,096,780,106ns |
| Quick Sort | 360,827ns | 2,028,933ns | 23,250,881ns | 83,773,031ns | 900,535,400ns |

With the unsorted files from elements of 100 to 1,000,000, Insertion Sort was constantly the

slowest of all 4 sorting methods from all different number of elements test. This is true by looking at the

theoretical analysis which has insertion sort to have an average and worst case of O(n^2). With the heap,

merge and quick sort the theoretical analysis says that their average cases are O(nlogn). Merge and

heap sorts worst case is at most O(nlogn) while Quick sort is O(n^2). Looking at the table the fastest was

Quick Sort, followed by the heap sort and then the merge sort.

At the test with 1,000,000 elements the heap sort came out to be slower. It is said that the heap

sort is not a stable method and this shows that it seems to be true, while merge sort is stable and quick

sort is somewhat stable. I ran the methods a couple more times on each number of elements and it

seems like merge sort is pretty stable because the running time it was getting was pretty close to the

previous test, while heap and quick sort was getting different running times and sometimes heap sort

would be faster than quick sort.