Jeffrey McDonald

CS 3310

Project 1 report

Overall the implementation of all these algorithms were a smooth process as the algorithms themselves were simple to understand. However, one difficulty that I had encountered was the quick sort was on average slower than my merge sort by a factor 4 times. I had evaluated the possible problems with this was that my partition part of the code was entering a loop that evaluate the entire array from the same pivot again effectively redoing the loop once more for no reason. I had discovered this as a joint problem as my quicksort would throw an exception very quickly at a low input size of merely 20,000 ints. Along with the sorting algorithms were the multiplication of the matrixes returning a matrix with only zeroes held in the matrix. It was a simple syntax error that had shifted all the values to the right as I had established the end value for the for loop as length and not length-1.

Merge sort and Quick sort Analysis

|  |  |
| --- | --- |
| input | Time(ms) |
| 10,000 | 3 |
| 20,000 | 3 |
| 50,000 | 5 |
| 100,000 | 11 |
| 1,000,000 | 83 |

Time complexity of merge sort

All cases: O(nlogn)

|  |  |
| --- | --- |
| Input | Time(ms) |
| 10,000 | 7 |
| 20,000 | 5 |
| 50,000 | 30 |
| 100,000 | 72 |
| 1,000,000 | 7002 |

Time complexity of Quick sort:

Best case: O(nLogn)

Average Case: O(nLogn)

Worst Case: O(n2)

For my merge sort it seems that it ran in the average case scenario as there is no other scenario for the sort to change as it unconditionally breaks the array into halves and the recombines them. However for my quicksort when it had hit 1,000,000 my time had increased from 72 ms to 7002 ms. What this could’ve meant is that my pivot might have been a very low value and had very little to sort through to. On average it seems that my algorithms had

Matrix Multiplication

|  |  |
| --- | --- |
| Input | Time(ms) |
| 2 | 0 |
| 4 | 0 |
| 8 | 0 |
| 16 | 1 |
| 32 | 3 |
| 64 | 4 |
| 128 | 3 |
| 256 | 40 |
| 512 | 330 |
| 1024 | 3811 |
| 2048 | 46097 |

Time Complexity: O(n3)

|  |  |
| --- | --- |
| Input | Time(ms) |
| 2 | 0 |
| 4 | 0 |
| 8 | 0 |
| 16 | 0 |
| 32 | 2 |
| 64 | 2 |
| 128 | 6 |
| 256 | 44 |
| 512 | 313 |
| 1024 | 2457 |
| 2048 | 28,068 |

Time complexity: O(n2.87)

In comparison between the two algorithms they seem to be even hitting at around 0 ms or a really small value in nanoseconds. After the value 1024 the normal method would be longer by about 1400 more compared to the Strassen method. By 2048 the Strassen method is significantly faster than the normal method. It as difference of around 20,000 at the input of 2048.

Tower of Hanoi

|  |  |
| --- | --- |
| Input | Time(ms) |
| 2 | 1 |
| 4 | 1 |
| 8 | 1 |
| 16 | 374 |

Time Complexity:2n

For the tower of Hanoi the recurrence relation is 2M(n-1)+1. This is because first you must move every single disc on top of the bottom one to the middle peg and then moving that to the third peg. After that one must move the n-1 disc back the first and then repeating the process.

As using master theorem A=2,B=1,D=0

2>10

Which means O(nlog12) or 2n as the time complexity