CS3310 Project 1

Chris Saba

Recurrence Relations

Tower of Hanoi recurrence relation:

T1 = 1

T2 = 2(T1) +1 = 3

T3 = 2(T2) + 1 = 7

T4= 2(T3) + 1 = 15

Previous iteration is multiplied by 2, starting by t1 = 1 so next iteration is -1

2, 4, 8, 16…. 2^n

… Claim: Tn = 2^n -1

Tn = 2(Tn-1) + 1

Tn+1 = 2(Tn) + 1

= 2(2^n – 1) + 1

= 2^(n+1) – 1

T(64) = 2^64 – 1 = 18446744073709551616 − 1

T(4) = 2^4 – 1 = 16-1 = 15

Time complexity = O(2^n)

//CODE IMPLEMENTATIONS//

All code is ran in Java, using an Intel Core i7 9700K at 4.9GHz with 3000Mhz 32GB Ram

1) Merge Sort

Merge sort algorithm implemented and timed over array sizes 1000 up to 1 Billion, larger unallowed for max heap JVM limit. Arrays are created and random entries up to 10000 are created and written inside. Java JDK 8 features the Instant and Duration class, which allows to find differences in time and print it during the program. This was used to find the time scale of each iteration and will be used throughout the entire project.

Looking at the graph, you can see that it is a very fast sorting algorithm, and that the code doesn't change its time complexity by much on the contents of the array, which means it is very stable. Later I will explain why this is important.

2) Quick Sort

The quick sort algorithm is scaled similar to Merge sort, however the problem size stops at 125 Million, because of stack overflow errors. Included was an educated guess for a 1Billion size array, for the sake of comparison to Merge sort.

According to the graph, there might be some discrepancies you will find. The 125 Million sized array was sorted faster than a 100 million array, and the explanation here is a good market to show what happens when you have a worst case scenario vs a best case scenario. The result of time is based on the content of the array in Quick sort, so you can easily see that the 125Million got a better case scenario than the 100Million array. In the comparison chart between both, you can see that the quick sort is faster than the merge sort, but only up until 1Million in size, which is different with every iteration. This is the rough breakpoint between the algorithms, where switching to Merge sort after size 1Mil is faster than continuing with Quicksort. This process is used widely and especially in a case later on with Strassen's Method.

3) Tower of Hanoi

An easy implementation of Tower of Hanoi is written where nothing needs to be printed but its time to complete. All the algorithms and programs utilize the Instant and Duration, but the rest use it inside of a for loop that allows the program to keep iterating until the time to complete exceeds 600 seconds, or 10 Mins. ToH is an example of an algorithm that always has the same time complexity regardless of content, since there is no randomness or change of data, only the ring size amount. Written above is the recurrence relation and Time complexity for the ToH, and it accurately shows in accordance to the graph provided in the Excel sheet.

4) Classical Matrix Multiplication

Provided is the Classical IJK algorithm for multiplication, which is said to be slow, being O(n^3) time complexity… However that isn't always the case, since a fully recursive Strassen method is much slower than IJK. I did not provide the fully recursive Strassen method, since it takes too long, however there are 2 graphs of Strassen hybrids, which when cases of n array size is smaller than 50 and 100. Those array sizes are then calculated using regular IJK which is the same method as the provided algorithm. Therefore, neither algorithm by itself is considered fast, but a combination between the two are always faster.

5) Strassen Hybrid method.

As said before, the Strassen hybrid method is faster than the Strassen fully recursive method, and even then it isn't the fastest matrix multiplication algorithm available. However, the performance in comparison is much faster, allowing for Strassen to complete a 8192 size array multiplication in less than 3 minutes whereas the Classical method took over 7 mins for a 4096 sized array. There even is a noticeable difference between n <= 100 and n <= 50, where the 100 case does 8192 in 149 seconds, versus 50's 172 seconds. This is machine specific, and it depends on the processor of what size array the program switches to classical for performance, and there experiences a drop-off in performance after 100 eventually.