Activity 2. Divide and Conquer by subtraction.

Subtraction1 and Subtraction2 time give an abort time of 8000. This is caused by a Stack Overflow Error. The complexity for the 1st one is O(n) and for the 2nd is O(n^2), as the stack waste is linear, O(n), they both overflow.

Subtraction3 would take 7,0571E+16 ms to execute a n=80 execution, which is 2.237.804.609,34 years.

The complexity is quadratic O(2^n), in this case the stack does not overflow because

long before the execution time is untreatable.

Table for Subtraction4. Complexity O(n^3)

|  |  |
| --- | --- |
| Subtraction4 |  |
| n | time |
| 100 | 8 |
| 200 | 45 |
| 400 | 314 |
| 800 | 2414 |
| 1600 | 19737 |
| 3200 | (OOT)157774 |

Table for subtraction5. Complexity O(3^n/2)

|  |  |
| --- | --- |
| Subtraction5 |  |
| n | time |
| 30 | 638 |
| 32 | 2262 |
| 34 | 6496 |
| 36 | 19093 |
| 38 | 57180 |
| 40 | OOT |

It will take 18966,3558 years to execute n=80 executions of Subtraction5

Activity 3. Divide and conquer by division.

Time complexity for Division1 and 3 is O(n) and for Division3 is O(n log n).

Division4 and 5 table. Complexity O(n^2)

|  |  |  |
| --- | --- | --- |
| n | Division4 | Division5 |
| 1000 | 15 | 54 |
| 2000 | 75 | 219 |
| 4000 | 184 | 858 |
| 8000 | 728 | 3418 |
| 16000 | 2705 | 12071 |
| 32000 | 12345 | 42291 |
| 64000 | 51927 | OOT |
| 128000 | (OOT)185794 | OOT |

Division4 is has better performance than Division5.

For both implementations it is faster on the smaller inputs and the as the inputs starts to get bigger, the time complexity starts to match more with the O(n^2) complexity.

Activity 4. Two basic examples.

Table for VectorSum1, 2 and 3 with n=1,000. Complexity O(n)

|  |  |  |  |
| --- | --- | --- | --- |
| n | sum1 | sum2 | sum3 |
| 3 | 11 | 19 | 25 |
| 6 | 15 | 29 | 67 |
| 12 | 31 | 55 | 88 |
| 24 | 58 | 95 | 202 |
| 48 | 78 | 162 | 348 |
| 96 | 143 | 332 | 674 |
| 192 | 312 | 646 | 1364 |
| 384 | 561 | 1282 | 2714 |
| 768 | 1143 | 2518 | 5026 |
| 1536 | 2260 | 4950 | 8261 |
| 3072 | 4392 | 8878 | 16354 |
| 6144 | 6641 | 15878 | 32458 |
| 12288 | 13747 | StackOverflow | (OOT)66169 |
| 24576 | 26653 | StackOverflow | OOT |
| 49152 | 54696 | StackOverflow | OOT |

As we can clearly see, even though they are all O(n), sum1 is the best approach, then sum3 and with worse performance sum2, that ends up giving an StackOverflow Error.

Fibonacci tables tested with n=100,000

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n | fib1 | fib2 | fib3 | fib4 |
| 10 | 28 | 42 | 54 | 478 |
| 15 | 41 | 50 | 58 | 4183 |
| 20 | 42 | 69 | 72 | 43526 |
| 25 | 46 | 90 | 94 | OOT |
| 30 | 57 | 98 | 103 | OOT |
| 35 | 120 | 118 | 114 | OOT |
| 40 | 72 | 133 | 125 | OOT |
| 45 | 83 | 175 | 144 | OOT |
| 50 | 99 | 167 | 167 | OOT |
| 55 | 103 | 186 | 166 | OOT |
| 59 | 114 | 194 | 188 | OOT |

Fibonacci 1,2 and 3 have a similar performance as they are all O(n), being Fibonacci1 the best among the three, and fibonacci3 slightly faster than the 2nd option. Howewer the worst performance is from Fibonacci4 as it has a complexity of O(1.6^n), as it is exponential It reaches the OoT very fast.

Activity 5. Another task.

TABLE MERGESORT.

|  |  |  |  |
| --- | --- | --- | --- |
| n | ordered | reverse | random |
| 31250 | 42 | 31 | 34 |
| 62500 | 85 | 67 | 71 |
| 125000 | 193 | 125 | 146 |
| 250000 | 333 | 218 | 276 |
| 500000 | 556 | 476 | 609 |
| 1000000 | 1072 | 974 | 1205 |
| 2000000 | 2075 | 1998 | 2493 |
| 4000000 | 4267 | 4157 | 4512 |
| 8000000 | 6825 | 6671 | 7901 |
| 16000000 | 13546 | 13063 | 16270 |
| 32000000 | 27890 | 27017 | 33409 |
| 64000000 | 57321 | 55271 | OOT(68498) |

TABLE MERGESORT vs QUICKSORT

|  |  |  |  |
| --- | --- | --- | --- |
| n | mergesort | quicksort | t1/t2 |
| 10000 | LOR | LOR | LOR |
| 2\*10000 | LOR | LOR | LOR |
| 2\*\*2\*10000 | 63 | LOR | >1 |
| 2\*\*3\*10000 | 94 | LOR | >1 |
| 2\*\*4\*10000 | 190 | 78 | 2,43589744 |
| 2\*\*5\*10000 | 373 | 160 | 2,33125 |
| 2\*\*6\*10000 | 801 | 356 | 2,25 |
| 2\*\*7\*10000 | 1582 | 767 | 2,06258149 |
| 2\*\*8\*10000 | 3287 | 2195 | 1,49749431 |
| 2\*\*9\*10000 | 5289 | 3627 | 1,45822994 |
| 2\*\*10\*10000 | 10265 | 9468 | 1,08417828 |
| 2\*\*11\*10000 | 21078 | 29372 | 0,71762223 |
| 2\*\*12\*10000 | 43394 | OOT | <1 |
| 2\*\*13\*10000 | OOT | OOT | OOT |
| 2\*\*14\*10000 | OOT | OOT | OOT |

What constant do you get as a comparison of both algorithms?

Quicksort is faster on smaller inputs, however, as It is on random arrays, we are on the worst case scenario, meaning complexity goes to O(n^2), this translates to Quicksort being outperformed by Mergesort from 2^10\*10000 and beyond.