Activity 1. Divide and conquer by subtraction

The subtraction1 has a complexity of O (n) (a = 1, b = 1, k = 0).

The subtraction2 has a complexity of O (n^2) (a = 1, b = 1, k = 1).

The subtraction3 has a complexity of O (2^n) (a = 2, b = 1, k = 0).

The subtraction1 and 2 stop measuring with n = 8192 due to a stack overflow.

The subtraction3 with n = 80: we know that for n = 25 it takes 1180ms, now we compute (2^25) / (2^80) = 3.6028797e+16. Therefore the time for would be 1180 \* 3.6028797e+16, that is 1348109462201928 years.

Table with times of Subtraction4 and 5 without optimization in milliseconds.

|  |  |  |  |
| --- | --- | --- | --- |
| **N** | **S1** | **N** | **S2** |
| **100** | 1 | **30** | 363 |
| **200** | 11 | **32** | 1084 |
| **400** | 80 | **34** | 3356 |
| **800** | 619 | **36** | 10072 |
| **1600** | 4870 | **38** | 29624 |
| **3200** | 38698 | **40** | OoT(90K+) |

Using the same method used in calculating S3 and taking in account that 3^(40/2) = 89743ms, the time in years would be, 3.1291449e+14.

Activity 2. Divide and conquer by division

Division1 has a complexity of O (n) (a = 1, b = 3, k = 1).

Division2 has a complexity of O (n \* logn) (a = 2, b = 2, k = 1).

Division3 has a complexity of O (n) (a = 2, b = 2, k = 0).

With the timings we can see that D1’s are the fastest, D2’s are in the slowest and D3’s are in the middle.

Table with times of Division4 and 5 without optimization in milliseconds.

|  |  |  |
| --- | --- | --- |
| **N** | **S1** | **S2** |
| **1000** | 4 | 25 |
| **2000** | 17 | 94 |
| **4000** | 65 | 372 |
| **8000** | 255 | 1508 |
| **16000** | 1020 | 6116 |
| **32000** | 4067 | 24081 |
| **64000** | 16404 | OoT |
| **128000** | OoT | OoT |

Activity 3. Two basic examples (VectorSum and Fibonacci)

From the VectorSum1 class we can obtain that the complexities are:

-sum1: O(n) just a simple loop.

-sum2: O(n) (a = 1, b = 1, k = 0) (subtraction).

-sum3: O(n) (a = 2, b = 2, k = 0) (division).

Table with times VectorSum1, 2 and 3 without optimization in milliseconds.

|  |  |  |  |
| --- | --- | --- | --- |
| **Size** | **VS1** | **VS2** | **VS3** |
| **3** | 43 / 10^7 = 0,000043 | 75 / 10^6 = 0,000075 | 90 / 10^6 = 0,00009 |
| **6** | 63 / 10^7 = 0,000063 | 119 / 10^6 = 0,000119 | 160/ 10^6 = 0,00016 |
| **12** | 87 / 10^7 = 0,000087 | 233 / 10^6 = 0,000233 | 360 / 10^6 = 0,00036 |
| **24** | 131/ 10^7 = 0,000131 | 429 / 10^6 = 0,000429 | 720 / 10^6 = 0,00072 |
| **48** | 226 / 10^7 = 0,000226 | 815 / 10^6 =  0,000815 | 146 / 10^6 = 0,00146 |
| **96** | 407 / 10^7 = 0,000407 | 1581 / 10^6 = 0,001581 | 2870 / 10^6 = 0,00287 |
| **192** | 773 / 10^7 = 0,000773 | 3140/ 10^6 = 0,003140 | 5860 / 10^6 = 0,00586 |
| **384** | 1506 / 10^7 = 0,001506 | 6333/ 10^6 = 0,006333 | 11580/ 10^6 =0,01158 |
| **768** | 2974 / 10^7 = 0,002974 | 12565/ 10^6 = 0,012565 | 23040/ 10^6 = 0,02304 |
| **1536** | 11680 / 10^7 = 0,011680 | 25229/ 10^6 = 0,025229 | 46680/ 10^6 = 0,04668 |
| **3072** | 23396 / 10^7 = 0,023396 | 50456/ 10^6 = 0,050456 | 93470/ 10^6 = 0,09347 |

All three methods have the same complexity, their difference in the timings comes from using iteration or recursion. At the beginning their timing are similar because there are not needed a lot of recursion calls but as the size grows large the recursion calls are more and the timing escalate for the methods using recursion.

Table with times Fibonacci1, 2 and 3 without optimization in milliseconds.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Order** | **F1** | **F2** | **F3** | **F4** |
| **10** | 92 / 10^6 = 0,000092 | 115 / 10^6 = 0,000115 | 190 / 10^6 = 0,000190 | 230 / 10^5 =0,0023 |
| **11** | 95 / 10^6 = 0,000095 | 119 / 10^6 = 0,000119 | 222 / 10^6 = 0,000222 | 363 / 10^5 =0,00363 |
| **12** | 101 / 10^6 = 0,000101 | 121 / 10^6 = 0,000121 | 233/ 10^6 = 0,000233 | 595 / 10^5 = 0,00595 |
| **13** | 105 / 10^6 = 0,000105 | 131 / 10^6 = 0,000131 | 251 / 10^6 = 0,000251 | 957 / 10^5 = 0,00957 |
| **14** | 110 / 10^6 = 0,000110 | 137 / 10^6 = 0,000137 | 270 / 10^6 = 0,000270 | 1533 / 10^5 = 0,01533 |
| **15** | 112 / 10^6 = 0,000112 | 146 / 10^6 = 0,000146 | 292 / 10^6 = 0,000292 | 2481 / 10^5 = 0,02481 |
| **20** | 136 / 10^6 = 0,000136 | 183 / 10^6 = 0,000183 | 366 / 10^6 = 0,000366 | 2767 / 10^4 = 0,2767 |
| **30** | 183 / 10^6 = 0,000183 | 257 / 10^6 = 0,000257 | 525 / 10^6 = 0,000525 | 3385 / 10^2 =33,85 |
| **40** | 232 / 10^6 = 0,000232 | 323 / 10^6 = 0,000323 | 696 / 10^6 = 0,000696 | 4151 |
| **50** | 282 / 10^6 = 0,000282 | 395 / 10^6 =0,000395 | 854 / 10^6 = 0,000854 | OoT |
| **59** | 318 / 10^6 = 0,000318 | 470 / 10^6 = 0,000470 | 998 / 10^6 = 0,000998 | OoT |

Here we can see the something similar to the first case. The ones with recursion take more time to compute, even more with the fib4 because it has two recursion calls. The difference between f1 and f2 is that f2 is using vectors and has to iterate throw them.