Activity 1. Divide and Conquer by subtraction

In both Subtraction1.java and Subtraction2.java, we have a StackOverflowError for values over n=8000, because we exceed the size of the computer’s stack with a big amount of recursive calls.

In Subtraction3.java, we have a = 2, b = 1 and k = 0. As it’s subtraction, and a > 1, we can obtain the time complexity O(an/b), therefore the time complexity will be O(2n). Then, knowing that the time it takes to manage n­1 = 29 is t1 = 37076 ms, we can obtain the time that it will take for n2 = 80 with:

t2 = (2n2/2n1)\*t1 = 8.348772989 \* 1019 ms.

Converted to years:

8.348772989 \* 1019 ms -> 8.348772989 \* 1016 s -> 1.391462165 \* 1015 min -> 2.319103608 \* 1013 hours -> 9.6629317 \* 1011 days -> **2647378584** **years**

The Subtraction4.java algorithm was created using two nested loops and a recursive call that give us a= 1, b=1, k=2, so our O(nk+1) is O(n3).

We obtain the following time measurements:

|  |  |
| --- | --- |
| **n** | **nTimes** |
| 100 | 6.2 ms |
| 200 | 40.6 ms |
| 400 | 302 ms |
| 800 | 2339 ms |
| 1600 | 18544 ms |
| 3200 | Oot |

The Subtraction5.java algorithm was created using two nested loops and three recursive call that give us a= 3, b=2, k=0, so our O(an/b) is O(3n/2).

We obtain the following time measurements:

|  |  |
| --- | --- |
| **n** | **nTimes** |
| 30 | 953 ms |
| 32 | 2189 ms |
| 34 | 6867 ms |
| 36 | 19477 ms |
| 38 | Oot |

If we want to know how many years would it last for our algorithm for a problem size of 80, we can use:

t2 = (3n2/2/3n1/2)\*t1 = (340/318)\* 19477 = 6.11208898 \* 1014 ms.

Converted to years:

6.11208898 \* 1014 ms -> 6.11208898 \* 1011 s -> 1.018681497 \* 1010 min -> 169780249.4 hours -> 7074177.06 days -> **19381.3** **years**

Activity 2. Divide and conquer by division

As it can be seen, Division1 and Division3 have the same time complexity O(n), so they have similar time measurements. On the other hand, Division2 has a time complexity of O(nlogn), which is worse than O(n), therefore it has larger time measures.

Division4.java algorithm was created using two nested loops and a recursive call that give us a= 1, b=2, k=2, so our O(nk) is O(n2) and a < bk.

|  |  |
| --- | --- |
| **n** | **nTimes** |
| 1000 | 9.5 ms |
| 2000 | 37.4 ms |
| 4000 | 150 ms |
| 8000 | 591 ms |
| 16000 | 2328 ms |
| 32000 | 9359 ms |
| 64000 | 37632 ms |

Division5.java algorithm was created using two nested loops and a recursive call that give us a= 1, b=2, k=2, so our O(nk) is O(n2) and a < bk.

|  |  |
| --- | --- |
| **n** | **nTimes** |
| 1000 | 9.5 ms |
| 2000 | 37.4 ms |
| 4000 | 150 ms |
| 8000 | 591 ms |
| 16000 | 2328 ms |
| 32000 | 9359 ms |
| 64000 | 37632 ms |