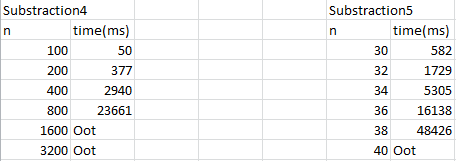
Activity 1. Divide and conquer by subtraction

In subtraction 1 we have a complexity of O(n) as a == 1 so it is O(n^(k+1)) and k is equal to 0, while in subtraction 2 we have a complexity O(n^3) as a == 1 and k == 2 and we have the same formula as in subtraction 1 as they both are divide and conquer by substraction.

They both stop giving times after n = 8192 because of an exception java.lang.StackOverflowError which is caused because the counter get too big so we get stack overflow.

To calculate how many years will take to execute Substraction3 with n = 80 we have to take a look at the complexity that is O(2n) as a > 1 so we have to apply O(a^(n/b)), and b is 1. Now We use the formula t2 = a^(n2-n1)\*t1 where k = n2/n1:

-> t = (2^80)/(2^30) \* 39527= 4,45 x 10^19 ms = 1,4 x 10^9 years



To calculate how many years will take to execute Substraction5 (O(3n/2)) with n = 80 we follow the same procedure as in Subtraction 3:

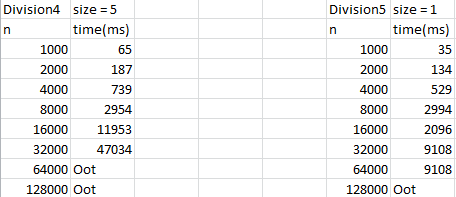
-> t = (3^(80/2))/(2^(30/2)) \* 582= 4,93 x 10^14 ms = 1,56 x 10^4 years

Activity 2. Divide and conquer by division

We can see that the fastest algorithm is Division1, which has complexity is O(n).

Division2 has a complexity O(nlogn) this is because in this case a is 2, b is 2 and k is 1 so a=b^k and we use the formula O(n^k logn). Division3 has the same complexity as Division1 but in this case k = 0 so a > b^k so we get the complexity in this way: O(n^(logba) and also another difference is that Division 1 divides the size problem by 3 and Division3 by 2 so with the same size problems and same repetitions, Division1 will always be faster.

Now I get the times of Division4 and Division4:

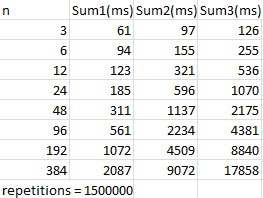


We can see that if both programs had the same size and despite having the same complexity, Division 4 is faster than Division5. This is probably because Division5 have many more recursive calls than Division4..

Activity 3. Basic Examples

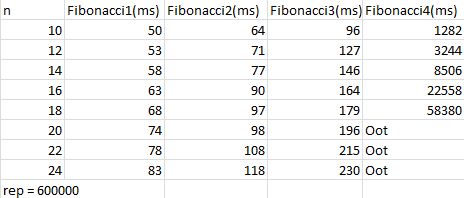
Sum1 has a linear complexity O(n), sum2 has the same complexity as Sum 1 but using a recursive call, and sum3 has also the same complexity as a > b^k and a = 2, b = 2 and k = 0 so O(n^(logba)) is also O(n)

From this point the times are measured with my personal computer



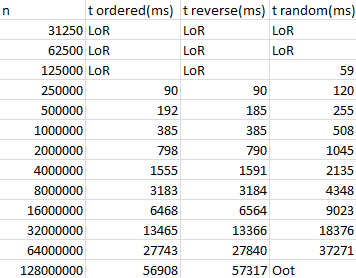
Sum1 is clearly the best as it does not use recursion

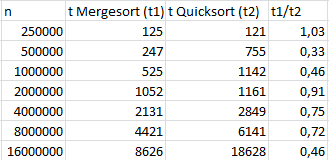
With the Fibonacci problems happens practically the same, Fibonacci1 and 2 do not use recursion and have both O(n) complexity, Fibonacci3 has a recursive call so O(n) and in Fibonacci 4 we get the complexity knowing that a = 2, b = 1 or 2 and k = 0 in both cases of b a > b^k and T(n)=T(n-1)+T(n-2)+O(1), that is an exponential solution O(1.6^n)



Clearly, Fibonacci1 and Fibonacci2 are the fastest ones. Fibonacci4 has the worst times as it has the worst complexity O(1.6n) .

Activity 4. Mergesort





With small problem size, quicksort seems to be a little bit faster, but as the problem size increases, Mergesort starts getting faster times than Quicksort.