Activity 1. Divide and Conquer by subtraction

**Subtraction 1:** a = 1 / b = 1 / k = 0

Complexity: O(nk+1) => O(n0+1) => O(n)

I cannot prove if the results fit or not with the theoretical complexity.

Due to low runtimes (<50mS) until the overflow arise before time > 50mS.

**Subtraction 2:** a = 1 / b = 1 / k = 1

Complexity: O(nk+1) => O(n1+1) => O(n2)

In the only >50mS values, the time grows by 4 (22) while the n grows by 2, so yes, it is quadratic as the theoretical value.

The algorithm aborts due to the lack of space as the stack is +- 30000 cells of size, so for a big n and given a waste of O(n), the stack soon, will be full, for a “low” n.

**Subtraction 3:** a = 2 / b = 1 / k = 0

Complexity: O(an/b) => O(2n/1) => O(2n)

The theoretical results matches the final result as the n increases by 1, the time multiplies by 2 so for a given n, the next expected time is: 2n+1

Given that: n=25\*\*TIME=1172\*\*

N2 = 80, N1 = 25, t1 = 1172 mS, ¿t2 in years?

1 year = 31 556 952 S

Then t2 = 2N2/2N1 \* t1 = 2N2-N1 \* t1 = 255 \* 1.172 S ≈ ***1 338 080 753.37 years for n = 80***

**Subtraction 4:** a = 1 / b = 1 / k = 2 -> Complexity: O(nk+1) => O(n2+1) => O(n3)

|  |  |
| --- | --- |
| n (102) | Time (mS) |
| 1 | 1.334 |
| 2 | 10.074 |
| 4 | 7.94 |
| 8 | 619.7 |
| 16 | 4939 |
| 32 | 38968 |
| 64 | OoT |

**Subtraction 5:** a = 3 / b = 2 / k = 0 -> Complexity: O(an/b) => O(3n/2)

|  |  |
| --- | --- |
| n | Time (mS) |
| 30 | 367 |
| 32 | 1088 |
| 34 | 3266 |
| 36 | 9764 |
| 38 | 29183 |
| 40 | OoT |

The time grows by a **factor of 3 by the increase in 2 of n**

Given that: n=38\*\*TIME=29183\*\*

N2 = 80, N1 = 38, t1 = 29183 mS, ¿t2 in years?

1 year = 31 556 952 S

Then t2 = 3N2/2/3N1/2 \* t1 = 3(N2-N1)/2 \* t1 = 321 \* 29.183 S ≈ ***9673.44652054 years for n = 80***

Activity 2. Divide and Conquer by division

**Division 4:** a = 2 / b = 2 / k = 2 -> Complexity: O(nk) => O(n2)

|  |  |
| --- | --- |
| n (103) | Time (mS) |
| 1 | 7.420 |
| 2 | 29.1 |
| 4 | 115.5 |
| 8 | 456.8 |
| 16 | 1847 |
| 32 | 7243 |
| 64 | 28951 |
| 128 | OoT |

**Division 5:** a = 4 / b = 2 / k = 0 -> Complexity: O(nlogb(a)) => O(n2)

|  |  |
| --- | --- |
| n (103) | Time (mS) |
| 1 | 23.5 |
| 2 | 93.4 |
| 4 | 372.3 |
| 8 | 1478.1 |
| 16 | 5907 |
| 32 | 23596 |
| 64 | OoT |
| 128 | OoT |

Activity 3. Two basic examples

**VectorSum.java:**

|  |  |  |  |
| --- | --- | --- | --- |
| n (103) | Time sum1 (mS) | Time sum2 (mS) | Time sum3 (mS) |
| 1 | 0.00386 | 0.0171 | 0. 0309 |
| 2 | 0.007637 | 0.0329 | 0.0608 |
| 4 | 0.015316 | 0.0658 | 0.1218 |
| 8 | 0.0304 | 0.1312 | 0.2471 |
| 16 | 0.0618 | StackOverFlow | 0.4951 |
| 32 | 0.1223 | StackOverFlow | 0.9929 |
| 64 | 0.2464 | StackOverFlow | 1.9614 |

**Fibonacci.java:** a = 4 / b = 2 / k = 0 -> Complexity: O(nlogb(a)) => O(n2)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| n (10) | Time fib1 (mS) | Time fib2 (mS) | Time fib3 (mS) | Time fib4 (mS) |
| 1 | 9.0E-5 | 1.13E-4 | 1.85E-4 | 0.002243 |
| 2 | 1.29E-4 | 1.79E-4 | 3.49E-4 | 0.275 |
| 3 | 1.68E-4 | 2.44E-4 | 5.04E-4 | 34.09 |
| 4 | 2.1E-4 | 3.11E-4 | 6.55E-4 | 4186 |
| 5 | 2.47E-4 | 3.81E-4 | 8.09E-4 | OoT |

Activity 5. Calendar

|  |  |
| --- | --- |
| n | t Calendar (mS) |
| 2^1 | 1.187E-4 |
| 2^2 | 2.392E-4 |
| 2^3 | 5.77E-4 |
| 2^4 | 0.0018452 |
| 2^5 | 0.006646 |
| 2^6 | 0.020016 |
| 2^7 | 0.07114 |
| 2^8 | 0.2697 |
| 2^9 | 1.0704 |
| 2^10 | 4.333 |
| 2^11 | 17.141 |
| 2^12 | 70.79 |
| 2^13 | 609.0 |
| 2^14 | 3407.5 |

They meet the the theoretical complexity as it is O(n2) as a=2 – b=2 – k=2 -> a < bk => O(nk)

The time should grow by terms of 4 for each time the n doubles. And it does for nearly all.