|  | Student information | Date | Number of session |
|---|---|---|---|
| **Algorithmics** | UO: 300047 | 03/02/2025 | 1 |
|  | Surname: Rodríguez Torres |  |  |
|  | Name: Luisa Natalia |  |  |

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
*Universidá d'Uviéu*
*University of Oviedo*

# Activity 1. Divide and Conquer by subtraction

**Subtraction1.java:**

- **A = 1**
- **B = 1**
- **K = 0**
- **TOTAL COMPLEXITY: O(n)**

**Subtraction2.java:**

- **A = 1**
- **B = 1**
- **K = 1**
- **TOTAL COMPLEXITY: O(n^2)**

**Subtraction3.java:**

- **A = 2**
- **B = 1**
- **K = 0**
- **TOTAL COMPLEXITY: O(2^n)**

**After running the algorithms we can see that they all have their expected complexities.**

**For Substraction1.java we stop computing at 16.384 due to a stackOverflow.**

**For Substraction2. java we stop computing at n = 32.768 due to a stackOverflow.**

**This happens due to how the recursions are programmed. These methods will use recursion n times, which at high numbers such as n = 16384, we will repeat the operation that amount of times. The Java stack cannot handle that much information.**

| Algorithmics | Student information | Date | Number of session |
|---|---|---|---|
| | UO: 300047 | 03/02/2025 | 1 |
| | Surname: Rodríguez Torres | | |
| | Name: Luisa Natalia | | |

**To calculate how long it would take for Substraction3 to compute at n=80, we can take a random measurement, in this example I will use 1788ms at n = 30. Since the complexity is O(2^n), the time will be calculated as: 1788×2^(n−30)ms. In this case 1788×2^(50)ms. We get approximately 2.01 × 10^18 milliseconds, or approximately 63,693,056 years.**

**Subtraction4.java:**

- **A = 1**
- **B = 1**
- **K = 2**
- **TOTAL COMPLEXITY: O(n^3)**
-

**For small values (100–800), the execution time is negligible, however after 1600, the time grows exponentially as expected, following a cubic pattern and quickly getting OoT.**

| n | Time (ms) |
|---|---|
| 100 | 2 |
| 200 | 2 |
| 400 | 0 |
| 800 | 0 |
| 1600 | 33 |
| 3200 | 279 |
| 6400 | 2104 |
| 12800 | 15314 |
| 25000 | OoT |

**Subtraction5.java:**

- **A = 2**
- **B = 2**
- **K = 1**
- **TOTAL COMPLEXITY: O(3^(n/2))**

**We can clearly see that the complexity of the problem translates to its real time. Every two sizes however, the time barely changes, due to perhaps, not having to divide the problem more from the previous iteration of the problem, so the same amount of invocations are needed.**

| n | Time (ms) |
|---|---|
| 30 | 18 |
| 32 | 62 |
| 34 | 61 |
| 36 | 548 |
| 38 | 539 |
| 40 | 4819 |
| 42 | 4886 |
| 44 | OoT |

## Activity 2. Divide and Conquer by division

### Division1.java:

- **A = 1**
- **B = 3**
- **K = 1**

| Algorithmics | Student information | Date | Number of session |
|---|---|---|---|
| | UO: 300047 | 03/02/2025 | 1 |
| | Surname: Rodríguez Torres | | |
| | Name: Luisa Natalia | | |

- **TOTAL COMPLEXITY: O(n)**

## Division2.java:

- **A = 2**
- **B = 2**
- **K = 1**
- **TOTAL COMPLEXITY: O(n·logn)**

## Division3.java:

- **A = 2**
- **B = 2**
- K = 0
- TOTAL COMPLEXITY: O(n)

**Division4.java:**

- A = 1
- B = 2
- K = 2
- TOTAL COMPLEXITY: O(n^2)

**Division5.java:**

- A = 5
- B = 2
- K = 2
- TOTAL COMPLEXITY: O(n^2)

| Algorithmics | Student information | Date | Number of session |
|---|---|---|---|
| | UO: 300047 | 03/02/2025 | 1 |
| | Surname: Rodríguez Torres | | |
| | Name: Luisa Natalia | | |

## Activity 4. Two basic examples

| Order | Fibonacci 1 time (ms) | Fibonacci 2 time (ms) | Fibonacci 3 time (ms) | Fibonacci 4 time (ms) |
|---|---|---|---|---|
| 10 | 279 | 293 | 680 | 13652 |
| 20 | 337 | 635 | 1327 | OoT |
| 30 | 457 | 944 | 1919 | OoT |
| 40 | 598 | 1407 | 2608 | OoT |
| 50 | 579 | 2004 | 3233 | OoT |
| 60 | 563 | 2518 | 3802 | OoT |

**Fibonacci 1 is an iterative solution with complexity O(n). Fibonacci 2 is also an iterative solution with complexity O(n) but it utilizes a vector, making it slower when having to deal with bigger sizes. Fibonnacci 3 is a recursive method with complexity O(n), this method is slower even at smaller orders. Finally, Fibonacci 4 has exponential complexity O(1.6^n), making it by far the worst version.**

| Size | VectorSum 1 time (ms) | VectorSum 2 time (ms) | VectorSum 3 time (ms) |
|---|---|---|---|
| 3 | 2 | 3 | 2 |
| 24 | 1 | 2 | 3 |
| 192 | 5 | 13 | 37 |
| 1536 | 38 | 230 | 254 |
| 12288 | 309 | 2069 | 2448 |
| 98304 | 2392 | 4243 | 16467 |

**VectorSum 1 presents a complexity of O(n) due to iterating through the array once. VectorSum 2 presents also a complexity of O(n), using recursion. VectorSum 3 also presents a complexity of O(n) by using divide and conquer. As we can observe, despite them all having the same complexxity, VectorSum 1 is much faster than the others since it uses simpler operations. By using a simple for-loop with a sum it computes a very simple operation which reduces its overhead. Vector2 uses recursion, so it must allocate memory for every iteration. Vector3 also has the same problem as Vector 2, except with even more overhead, since we must calculate the middle point.**

## Activity 2. Petanque championship organization

- **The complexity of the algorithm I have implemented is: O(n log(n)). It uses divide and conquer by division with the following parameters: A=2, B=2, K=1.**

| n | Time |
|---|---|
| 2 | 1 |
| 4 | 0 |
| 8 | 0 |
| 16 | 0 |
| 32 | 0 |
| 64 | 0 |
| 128 | 0 |
| 256 | 0 |
| 512 | 1 |
| 1024 | 5 |
| 2048 | 46 |
| 4096 | 176 |
| 8192 | 608 |

| Algorithmics | Student information | Date | Number of session |
|---|---|---|---|
| | UO: 300047 | 03/02/2025 | 1 |
| | Surname: Rodríguez Torres | | |
| | Name: Luisa Natalia | | |

| | |
|---|---|
| **16384** | **3712** |
| **32768** | **OutOfMemory** |

**We can see that this algorithm works very well for smaller numbers, and does indeed grow with a n log(n) complexity. However, for bigger numbers, the amount of function calls becomes too much for Java, since it must allocate memory in each of them, so we get and OutOfMemory exception.**