Activity 1. [Divide and Conquer by subtraction]

For substraction1 we have that the theoretical time complexity is O(n), we can reach that conclusion applying the divide and conquer theorem.

In substraction1 we can’t compare the time complexity with the theoretical one as it gives stack overflow error when the time is so low.

For substraction2 we have that the theoretical time complexity is O(n^2), we can reach that conclusion applying the divide and conquer theorem.

We can see that the theoretical complexity matches with the real one as, before it gives stack overflow error, we can see that the cont is growing exponentially.

It aborts at n = 8192 but not because of the time, because of an stack overflow error as it is charging in the stack too much calls to the function so when the stack is full it throws the error.

For substraction3 we have that the theoretical time complexity is O(2^n), we can reach that conclusion applying the divide and conquer theorem.

The theoretical complexity matches with the real one as it grows exponentially.

It would take for n=80 will take 2715640871 years.

For substraction4:

|  |  |
| --- | --- |
| N | Substraction4 (seconds) |
| 100 | 0,054 |
| 200 | 0,372 |
| 400 | 3,074 |
| 800 | 23,803 |

For substraction5:

|  |  |
| --- | --- |
| N | Substraction5 (seconds) |
| 30 | 0,726 |
| 32 | 2,022 |
| 34 | 6,092 |
| 36 | 17,642 |
| 38 | 52,729 |

It will take 17555 years for n = 80.

Activity 2. [Divide and conquer by division]

The complexity of division1 is O(n^(log1(3)). Executing the code, we can see that the time complexity is the same as the expected.

The complexity of division2 is O(n\*log(n)). It matches the theoretical time complexity as it have 2 calls to the same function, it reduces by half every call and it has a loop with a O(n) complexity inside the function.

The complexity of division3 is O(log(n)). It matches the theoretical complexity as it has two recursive call, it reduces by half in each of them and it doesn’t contain any loop inside.

For division4:

|  |  |
| --- | --- |
| N | Time (s) |
| 1000 | 0,059 |
| 2000 | 0,203 |
| 4000 | 0,793 |
| 8000 | 3,138 |
| 16000 | 12,458 |
| 32000 | 49,824 |
| 64000 | Oot |

The times matches the expected time as O (2000) is expected to have a time of 0,236 and the result time is similar.

For division5:

|  |  |
| --- | --- |
| N | Time (s) |
| 1000 | 0,255 |
| 2000 | 0,997 |
| 4000 | 4,004 |
| 8000 | 15,448 |
| 16000 | 61,502 |
| 32000 | Oot |

The times matches the expected time as O (2000) is expected to have a time of 1,020 and the result time is similar.

Activity 3. [Two basic examples]

Times for VectorSum1 O(n):

|  |  |
| --- | --- |
| N | Time(s) |
| 3 | 0,059 |
| 6 | 0,078 |
| 12 | 0,128 |
| 24 | 0,231 |
| 48 | 0,422 |
| 96 | 0,801 |
| 192 | 1,558 |
| 384 | 3,081 |
| 768 | 6,159 |
| 1536 | 12,202 |
| 3072 | 24,758 |
| 6144 | Oot |

Times for VectorSum2 O(n):

|  |  |
| --- | --- |
| N | Time(s) |
| 3 | 0,103 |
| 6 | 0,167 |
| 12 | 0,294 |
| 24 | 0,538 |
| 48 | 1,015 |
| 96 | 1,975 |
| 192 | 3,938 |
| 384 | 7,843 |
| 768 | 15,735 |
| 1536 | 50,538 |
| 3072 | Oot |

Times for VectorSum3 O(n\*log(n)):

|  |  |
| --- | --- |
| N | Time(s) |
| 3 | 0,118 |
| 6 | 0,244 |
| 12 | 0,512 |
| 24 | 1,060 |
| 48 | 2,214 |
| 96 | 6,004 |
| 192 | 15,611 |
| 384 | 17,259 |
| 768 | 49,256 |
| 1536 | Oot |

Comparing the three sums, we can see that the worst is the last one with the worst complexity of the three. We can see that comparing the ones with O(n) complexity, is worst the one that uses recursive calls.

Fibonacci1 O(n):

|  |  |
| --- | --- |
| N | Time(s) |
| 10 | 0,126 |
| 15 | 0,18 |
| 20 | 0,219 |
| 25 | 0,267 |
| 30 | 0,313 |
| 35 | 0,358 |
| 40 | 0,409 |
| 45 | 0,501 |
| 50 | 0,53 |
| 55 | 0,597 |
| 59 | 0,638 |

Fibonacci2 O(n):

|  |  |
| --- | --- |
| N | Time(s) |
| 10 | 0,163 |
| 15 | 0,254 |
| 20 | 0,342 |
| 25 | 0,384 |
| 30 | 0,456 |
| 35 | 0,520 |
| 40 | 0,588 |
| 45 | 0,659 |
| 50 | 0,730 |
| 55 | 0,795 |
| 59 | 0,848 |

Fibonacci3 O(n):

|  |  |
| --- | --- |
| N | Time(s) |
| 10 | 0,275 |
| 15 | 0,352 |
| 20 | 0,438 |
| 25 | 0,566 |
| 30 | 0,636 |
| 35 | 0,748 |
| 40 | 0,860 |
| 45 | 0,931 |
| 50 | 1,064 |
| 55 | 1,210 |
| 59 | 1,284 |

Fibonacci4 O(1.6^n):

|  |  |
| --- | --- |
| N | Time(s) |
| 10 | Lor |
| 12 | 0,086 |
| 14 | 0,249 |
| 16 | 0,630 |
| 18 | 1,609 |
| 20 | 4,069 |
| 22 | 10,464 |
| 24 | 27,871 |

Comparing the four Fibonacci, we can see that the worst one is the one with the worst complexity as expected. But taking in account the others, the fastest is the one simpler, the second fastest is the one that uses an array and the slowest one of these three, is the one that uses recursive calls.

Activity 4. [Another task]

|  |  |  |  |
| --- | --- | --- | --- |
| N | Time(s) ordered | Time(s) reverse | Time(s) random |
| 31250 | Lor | Lor | Lor |
| 62500 | Lor | Lor | 0,054 |
| 125000 | 0,078 | 0,087 | 0,121 |
| 250000 | 0,149 | 0,156 | 0,186 |
| 500000 | 0,323 | 0,328 | 0,387 |
| 1000000 | 0,672 | 0,668 | 0,848 |
| 2000000 | 1,413 | 1,327 | 1,754 |
| 4000000 | 2,838 | 2,646 | 4,290 |
| 8000000 | 5,627 | 5,532 | 6,978 |
| 16000000 | 11,525 | 12,400 | 14,066 |
| 32000000 | 23,997 | 26,626 | 30,061 |
| 64000000 | 51,268 | 58,497 | Oot |

|  |  |  |  |
| --- | --- | --- | --- |
| N | t Mergesort (t1) | t Quicksort (t2) | t1/t2 |
| 250000 | 0,186 | 0,139 | 1,33 |
| 500000 | 0,381 | 0,253 | 1,50 |
| 1000000 | 0,803 | 0,540 | 1,48 |
| 2000000 | 1,578 | 1,253 | 1,25 |
| 4000000 | 3,269 | 2,754 | 1,18 |
| 8000000 | 6,548 | 6,464 | 1,01 |

As we can see, as the complexity is the same, the times are practically the same. The one advantage that mergesort has is that it doesn’t have that case that quicksort has when it reaches n2 complexity.