

## Punto 2:

### Array 2:

CodingBat code practice [about](#) | [help](#) | [code help+videos](#) | [done](#) | [report](#) | [prefs](#) aospinap1  
@eafit.edu.co  
[log out]

[Java](#) [Python](#)

Array-2 ★★☆☆☆☆ chance

Medium array problems -- 1 loop. See the [Java Arrays and Loops](#) document for help.

|              |                |                   |
|--------------|----------------|-------------------|
| ✓ countEvens | ✓ bigDiff      | ✓ centeredAverage |
| ✓ sum13      | ✓ sum67        | ✓ has22           |
| ✓ lucky13    | ✓ sum28        | ✓ more14          |
| ✓ fizzArray  | ✓ only14       | ✓ fizzArray2      |
| ✓ no14       | ✓ isEverywhere | ✓ either24        |
| ✓ matchUp    | ✓ has77        | ✓ has12           |
| ✓ modThree   | ✓ haveThree    | ✓ twoTwo          |
| ✓ sameEnds   | ✓ tripleUp     | ✓ fizzArray3      |
| ✓ shiftLeft  | ✓ tenRun       | ✓ pre4            |
| ✓ post4      | ✓ notAlone     | ✓ zeroFront       |
| ✓ withoutTen | ✓ zeroMax      | ✓ evenOdd         |
| ✓ fizzBuzz   |                |                   |

- countEvents:  $O(n)$ .
- bigDiff:  $O(n)$ .
- centeredAverage:  $O(n)$ .
- sum13:  $O(n)$ .
- fizzBuzz:  $O(n)$ .

### Array 3:

CodingBat code practice [about](#) | [help](#) | [code help+videos](#) | [done](#) | [report](#) | [prefs](#) aospinap1  
@eafit.edu.co  
[log out]

[Java](#) [Python](#)

Array-3 ★☆☆☆☆ chance

Harder array problems -- 2 loops, more complex logic. See the [Java Arrays and Loops](#) document for help.

|              |             |               |
|--------------|-------------|---------------|
| ✓ maxSpan    | ✓ fix34     | ✓ fix45       |
| ✓ canBalance | ✓ linearIn  | ✓ squareUp    |
| ✓ seriesUp   | ✓ maxMirror | ✓ countClumps |

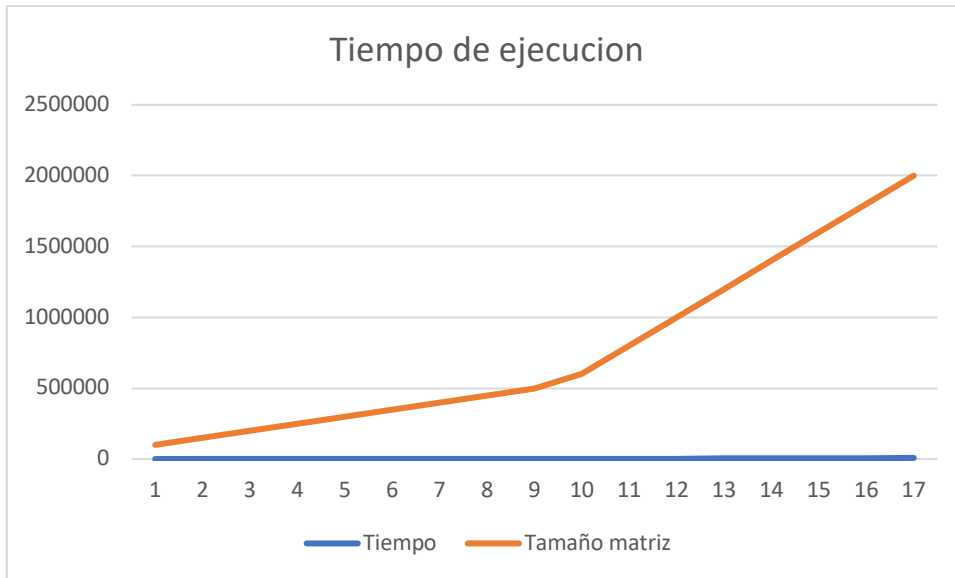
- maxSpan:  $O(1)$ .
- fix34:  $O(n)$ .
- fix45:  $O(n^2)$ .
- canBalance:  $O(n^2)$ .
- seriesUp:  $O(n^2)$ .

## Punto 4:

1. D
2. D
3. B
4. B
5. D
6. A
7.  $T(n) = T(n-1) \text{ ----- } O(n)$
8. D
9. D
- 10.
11. C
12. A
13. C
14. B

**Punto 3:**

| Tiempo | Tamaño matriz |
|--------|---------------|
| 0      | 100000        |
| 100    | 150000        |
| 220    | 200000        |
| 250    | 250000        |
| 500    | 300000        |
| 600    | 350000        |
| 800    | 400000        |
| 890    | 450000        |
| 900    | 500000        |
| 1100   | 600000        |
| 1800   | 800000        |
| 2900   | 1000000       |
| 4000   | 1200000       |
| 5600   | 1400000       |
| 6500   | 1600000       |
| 7000   | 1800000       |
| 8230   | 2000000       |



### 3.3

A mi modo de ver, el metodo que es mas optimo para utilizar en los video juegos es el margesort debido a que maneja gran cantidad de numeros y recurre a procesos mas optimos haciendo que le metodo se demore menos en ser ejecutado, tambien se puede ver esto en su complejidad debido a la que la complejidad de margesort es( $O(n \log n)$ ) y la de insrtion( $O(2^n)$ ), se puede ver claramente que en su complejidad se puede ver que el insrtion toma porcesos mas demorados debido a que tiene que procesar mas cantidad de numeros

### 3.4

No, no es el mas eficiente debido a su complejidad, el mejor es el margesort