```
Primes (A,P,r)
    if P==r:
      a= CheckPrimes(A,P)
      return 1 if prime, 0 if not prime
    else;
        q = ⌊(P+r)/2⌋
        a= Primes (A,P,q) → Calculate amount of primes of first and
        b = Primes (A,q+1,r) ↗ second split
        return a+b

Check primes(A,P)
    asign n the value of A[P]
    cont =0
    IF n in P:  ~ P is an array of Cache, we add the prime numbers we
       return 1   already found to optimize

    else:
        if n is either 1 or 0:
           return 0
        for i in range(2, √n +1):
           if n is divisible by i at any point:
                return 0
        P. apperd (n) ~ Store n in P
        return 1
```

(b)

Se plantea una ecuación de recurrencia muy similar (prácticamente igual) a la función Merge-Sort, ya que tiene una estructura similar.

$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 2T(n/2) + \Theta(n) & n > 1 \end{cases}$$

Esta parte surge de la función llamándose a sí misma cada vez 2 veces pero reduciendo a la mitad la longitud del arreglo que se va a analizar.

Esta parte surge de la función que analiza si un número es primo o no, y como contiene un for que involucra a $\sqrt{n}$, ya que se busca si tiene divisores.

(c) Se usa el teorema maestro de la siguiente forma:

$$T(n) = 2T(n/2) + \sqrt{n}$$

$$a = 2,\ b = 2,\ f(n) = \sqrt{n}$$

$$n^{\log_2 2} = n$$

$$f(n) = O(\sqrt{n}) = O(n^{1-\frac{1}{2}}) = O(n^{\log_b a - \epsilon}) \quad \epsilon = \frac{1}{2}$$

$$T(n) = \Theta(n)$$