

Programación

PROBLEMA 1 (5 puntos)

Se tiene un array A con N elementos. El array contiene valores naturales (enteros positivos), pudiendo existir elementos repetidos. El intervalo de valores posibles es desconocido y N puede ser grande. Se quiere separar en el array A los números primos de los que no lo son, colocando todos los primos juntos al principio del array y el resto al final. No se conoce a priori cuantos primos hay.

Ejemplo:

4	25	14	1	13	62	18	2	13	52	11	22	14	16	7	9	10
---	----	----	---	----	----	----	---	----	----	----	----	----	----	---	---	----



7	11	13	1	13	2	18	62	14	52	25	22	14	16	4	9	10
---	----	----	---	----	---	----	----	----	----	----	----	----	----	---	---	----

Nota: Utilizar la función `EsPrimo(n)`, que devuelve 1 si n es primo, 0 si no es primo. Que habrás de programar.

Se pide:

- Realizar un programa que separe los números primos de los que no lo son en el menor tiempo posible.
- Indica el orden del tiempo de ejecución del algoritmo anterior en función del tamaño N, razonando la respuesta, ¿Es el modelo más eficiente? (Contesta a esta pregunta dentro del código como un apartado nuevo).

Utiliza este código para la función `EsPrimo`, para poder testar tu problema

Programa 2: (5 puntos)

Como sabes, el almacenamiento árbol montículizado es un vector indexado, donde (usando 0-indexing, al estilo Python) donde:

- raíz en el primer elemento $V[0]$
- nodo en $v[i]$: \Rightarrow Hijo izquierdo en $V[2*i+1]$
- nodo en $v[i]$: \Rightarrow Hijo derecho en $V[2*i+2]$

I) Considerando un vector V con n elementos que está correctamente monticulizado, escribe un algoritmo recursivo que construye y retorna el árbol binario representado por el vector dado.

Puedes utilizar el léxico habitual de creación de Arbol, asignando el nodo raíz, la creación de nodo, y la asignación de los atributos `node.parent`, `node.left` y `node.right`, para estructurar el árbol, según el ejemplo a continuación

```
class Node:
    def __init__(self, key):
        self.key    = key
        self.left   = None
        self.right  = None
        self.parent = None

class Mont:
    def __init__(self, aRootNode: Node):
        self.rootNode = aRootNode
```

II) Escribe un método de la clase Mont que recorra el árbol generado en el orden que elijas (indica cuál !) y retorne una lista con todas las claves.
(Nota: los cinco puntos no requieren este punto opcional)