Retos de Programación en Python

1. Fernando y los primos

Dado un número N, determinar si es primo.
Complejidad: O(√n)

```
def es_primo(n: int) -> bool:
    if n < 2:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True
```

2. Isaac y los intervalos mágicos

Contar la cantidad de primos en cada intervalo (a,b).
Complejidad: Construcción O(n log log n), consultas O(1)

```
def criba_eratostenes(limite: int):
    es_primo = [True] * (limite + 1)
    es_primo[0] = es_primo[1] = False
    for i in range(2, int(math.sqrt(limite)) + 1):
        if es_primo[i]:
            for j in range(i*i, limite + 1, i):
                es_primo[j] = False
    return es_primo

def conteo_primos_intervalos(intervalos, limite=1000000):
    es_primo = criba_eratostenes(limite)
    prefijo = [0] * (limite + 1)
    for i in range(1, limite + 1):
        prefijo[i] = prefijo[i-1] + (1 if es_primo[i] else 0)
    return [prefijo[b] - prefijo[a-1] for a, b in intervalos]
```

3. Ana y los números palíndromos

Determinar si un número es palíndromo.
Complejidad: O(d) (d = número de dígitos)

```
def es_palindromo(n: int) -> bool:
    s = str(n)
    return s == s[::-1]
```

4. Bruno y los números perfectos

Determinar si un número es perfecto.
Complejidad: O(√n)

```
def es_perfecto(n: int) -> bool:
    if n < 2:
        return False
    suma = 1
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            suma += i
```

```
            if i != n // i:
                suma += n // i
    return suma == n
```

5. Camila y los cuadrados ocultos

Contar los cuadrados perfectos en un intervalo.
Complejidad: O(1) por consulta

```
def cuadrados_perfectos(intervalos):
    resultados = []
    for a, b in intervalos:
        start = math.ceil(math.sqrt(a))
        end = math.floor(math.sqrt(b))
        resultados.append(max(0, end - start + 1))
    return resultados
```

6. Diego y la suma misteriosa

Calcular la suma de una lista de enteros.
Complejidad: O(n)

```
def suma_lista(lista):
    return sum(lista)
```

7. El laberinto de Laura

Distancia mínima en grafo no ponderado con BFS.
Complejidad: O(V + E)

```
def bfs(grafo, s):
    n = len(grafo)
    dist = [-1] * n
    dist[s] = 0
    cola = deque([s])
    while cola:
        u = cola.popleft()
        for v in grafo[u]:
            if dist[v] == -1:
                dist[v] = dist[u] + 1
                cola.append(v)
    return dist
```

8. El árbol genealógico de Sofía

Recorrido inorden de un árbol binario.
Complejidad: O(n)

```
class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.izq = None
        self.der = None

def inorden(raiz):
    if raiz:
        return inorden(raiz.izq) + [raiz.valor] + inorden(raiz.der)
    return []
```

9. La biblioteca de Daniel

Ordenar un arreglo con Merge Sort.
Complejidad: O(n log n)

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr)//2
    izquierda = merge_sort(arr[:mid])
    derecha = merge_sort(arr[mid:])
    return merge(izquierda, derecha)

def merge(izq, der):
    resultado = []
    i = j = 0
    while i < len(izq) and j < len(der):
        if izq[i] < der[j]:
            resultado.append(izq[i]); i += 1
        else:
            resultado.append(der[j]); j += 1
    resultado.extend(izq[i:])
    resultado.extend(der[j:])
    return resultado
```

10. La misión de Mariana

Búsqueda binaria en una lista ordenada.
Complejidad: O(log n)

```
def busqueda_binaria(arr, x):
    izq, der = 0, len(arr)-1
    while izq <= der:
        mid = (izq + der)//2
        if arr[mid] == x:
            return True
        elif arr[mid] < x:
            izq = mid + 1
        else:
            der = mid - 1
    return False
```