**Ejercicio 2.** 4 puntos

1. [2 puntos] Programar en Haskell una función que satisfaga la especificación del problema a del Ejercicio 1. Recordá escribir los tipos de los parámetros.

   Pueden asumir como existentes las siguientes funciones sobre listas: cantidadDeApariciones, esPermutacion, estaOrdenada y mínimo y máximo.

2. [2 puntos] Programar en Python una función que implemente el enunciado del Ejercicio 1.2. Recordá escribir los tipos de los parámetros y variables que uses en tu implementación. Dado que el Ejercicio 1.2 utiliza el Ejercicio 1.1, asumir que ya existe una implementación del Ejercicio 1.1.

**Ejercicio 3.** 2 puntos

Sea la siguiente especificación del problema **sonIguales**, una posible implementación en lenguaje imperativo y el test suite:

```
problema sonIguales (in uno: seq⟨Char⟩,in otro: seq⟨Char⟩) : Bool {
    requiere: {|uno| > 0 ∧ |otro| > 0}
    asegura: {result = true ↔ (|uno| = |otro| ∧ (∀i : Z)(0 ≤ i < |uno| → uno[i] = otro[i]))}
}
```

```
def sonIguales(uno: str, otro: str) -> bool:
L1:     if len(uno) < len(otro):
L2:         return False
L3:     else:
L4:         i: int = 0
L5:         mitad: int = int(len(uno)/2) # ej: int(5/2) -> int(2.5) -> 2
L6:         while i <= mitad:
L7:             if (uno[i] != otro[i]) or (uno[len(uno)-i-1] != otro[len(otro)-i-1]):
L8:                 return False
L9:             i = i + 1
L10:        return True
```

| | Test 1 | Test 2 | Test 3 |
|---|---|---|---|
| Entrada | $uno = "abc"$<br>$otro = "abc"$ | $uno = "abc"$<br>$otro = "axy"$ | $uno = "ab"$<br>$otro = "abc"$ |
| Salida Esperada | Verdadero | Falso | Falso |

1. [0.25 puntos] Dar el diagrama de control de flujo (control-flow graph) del programa sonIguales.
2. [0.5 puntos] ¿La ejecución del test suite resulta en la ejecución de todas las líneas del programa sonIguales? Justifique.
3. [0.5 puntos] ¿La ejecución del test suite resulta en la ejecución de todas las decisiones (branches) del programa? Justifique.
4. [0.75 puntos] Detalle **todos** los errores de la implementación. Agregar nuevos casos de tests y/o modificar casos de tests existentes para que el test suite detecte el/los defecto/s.

**Ejercicio 4.** 2 puntos

1. [1 punto] Dada la siguiente especificación:

   ```
   problema raizCuadrada (in x: Float) : Float {
       requiere: {x ≥ 0}
       asegura: {res² = x}
   }
   ```

   Indique y justifique cuáles de los siguientes algoritmos cumplen con la especificación:

   a) Si $x \geq 0$ devuelve $\sqrt{x}$, si no devuelve 0
   b) Si $x > 0$ devuelve $\sqrt{x}$, si no devuelve 0
   c) Si $x > 1$ devuelve $\sqrt{x}$, si no devuelve 0
   d) Si $x \geq 0$ devuelve $-\sqrt{x}$, si no se indefine

2. [1 punto] ¿Qué relación hay entre una especificación y un algoritmo? Dar un ejemplo de ambos conceptos.

# Justifique todas sus respuestas

**ercicio 1.** 2 puntos

1. [1 punto] Dada la siguiente especificación relacionada a un juego de mesa completar nombres adecuados para el problema $a$, el parámetro $b$, las etiquetas $x$, $y$, $s$, $t$, $u$, $v$ y $w$ y los predicados $p1$, $p2$, $p3$, $p4$ y $p5$. Justifique los nombres elegidos describiendo brevemente el problema.

```
problema a (in b: seq(Z) ) : Z {
    requiere x: {(∀i : Z)(0 ≤ i < |b| → 0 < b[i] < 7)}
    requiere y: { p1(b) }
    requiere z: { |b| = 5 }
    asegura s: { p2(b) ⟷ (resultado = 55) }
    asegura t: { p3(b) ⟷ (resultado = 45) }
    asegura u: { p4(b) ⟷ (resultado = 35) }
    asegura v: { p5(b) ⟷ (resultado = 25) }
    asegura w: { (¬p2(b) ∧ ¬p3(b) ∧ ¬p4(b) ∧ ¬p5(b)) ⟷ (resultado = 0)}
}

pred p1 (b:seq(Z)) {
    (∀i, j : Z)((0 ≤ i < |b| ∧ 0 ≤ j < |b| ∧ i <= j) ⟹ (b[i] <= b[j]))
}
pred p2 (b:seq(Z)) {
    (∀i, j : Z)((0 ≤ i < |b| ∧ 0 ≤ j < |b|) ⟹ (b[i] = b[j]))
}
pred p3 (b:seq(Z)) {
    (∃n, m : Z)(m ≠ n ∧ n ∈ b ∧ m ∈ b ∧ cantAp(b, n) = 1 ∧ cantAp(b, m) = 4)
}
pred p4 (b:seq(Z)) {
    (∃n, m : Z)(m ≠ n ∧ n ∈ b ∧ m ∈ b ∧ cantAp(b, n) = 2 ∧ cantAp(b, m) = 3)
}
pred p5 (b:seq(Z)) {
    (∃n, m, o, p, q : Z)(n ∈ b ∧ m ∈ b ∧ o ∈ b ∧ p ∈ b ∧ q ∈ b ∧ m = n + 1 ∧ o = m + 1 ∧ p = o + 1 ∧ q = p + 1)
}
```

2. [1 punto] Especificar el siguiente problema (se puede especificar de manera formal o semi-formal):

   Dados los inputs *jugadasJugador1*: $seq(seq(\mathbb{Z}))$, *jugadasJugador2*:$seq(seq(\mathbb{Z}$ ... retornar 0, 1 o 2 según quién sea el jugador que obtiene más puntos. Los parámetros *jugadasJugador1* y *jugadasJugador2* representan jugadas. El puntaje de cada jugador será el resultado de aplicar el `problema a` (del punto 1.1) a cada elemento de la secuencias *jugadasJugador1* y *jugadasJugador2* y luego sumar todos estos valores. En caso de empate, el resultado será 0. En este juego, todos los jugadores tienen siempre la misma cantidad de jugadas.