Unión

Lenguaje natural:

Teniendo dos conjuntos A y B ordenados de menor a mayor, hay que retorna un nuevo conjunto que tenga todos los elementos que hay en A, en B y o en ambos sin repetir

Precondición: A y B son conjuntos no null, no tienen elementos repetidos y están ordenados.

Postcondición: Se devuelve un nuevo conjunto ordenado que contiene los elementos de tanto A como B sin repeticiones

Seudocódigo

```
i ← 0

j ← 0
C ← conjunto vacío
Mientras i < largo(A) Y j < largo(B) hacer
        Si A[i] < B[j] entonces
                Agregar A[i] a C
                i ← i + 1
        Sino si A[i] > B[j] entonces
                Agregar B[j] a C
                j ← j + 1
        Sino // A[i] == B[j]
                Agregar A[i] a C
                i ← i + 1
                j ← j + 1

Fin Mientras

Mientras i < largo (A) hacer

        Agregar A[i] a C

        i ← i + 1
Mientras j < largo(B) hacer
        Agregar B[j] a C
        j ← j + 1
Retornar C
FinAlgoritmo
```

Intersección

Lenguaje natural:
Teniendo dos conjuntos A y B ordenados de menor a mayor, hay que retornar un nuevo conjunto que contenga los elementos que están en ambos conjuntos a la vez.

Precondición:
A y B son conjuntos no null, no tienen elementos repetidos y están ordenados de mayor a menor.

Postcondición:
Se devuelve un nuevo conjunto ordenado que contiene únicamente los elementos que están en A y a la vez en B, sin repeticiones.

Seudocódigo

```
  i ← 0
  j ← 0
  C ← conjunto vacío
  Mientras i < largo(A) Y j < largo (B) hacer
    Si A[i] < B[j] entonces
        j ← j + 1
    Sino si A[i] > B[j] entonces
        i ← i + 1
    Sino //Esto es cuando son iguales, podría usar tanto a como b
        Agregar A[i] a C
        i ← i + 1
        j ← j + 1
  Fin Mientras
  Retornar C
FinAlgoritmo
```

Orden de ejecución del algoritmo:

Unión

Al principio, para resolver la unión de dos conjuntos, se me ocurrió recorrer por separado cada conjunto y, por cada elemento, revisar si ya estaba presente en el conjunto que voy a devolver antes de agregarlo. Esto implicaba hacer una búsqueda dentro del conjunto resultado para evitar duplicados, lo que llevaría recorrerlo nuevamente. En términos de complejidad, esta solución tendría un orden de tiempo de ejecución de aproximadamente $n^2 + m^2$, ya que recorrería los elementos de cada conjunto y tendría que asegurarse de que no sea un elemento repetido.

Luego, investigando una forma más eficiente de hacerlo, leí que si ambos conjuntos están ordenados, se podría mejorar el rendimiento del programa. La idea es utilizar dos variables, i y j(i para A y j para B), que funcionan como punteros para recorrer simultáneamente los conjuntos A y B. Mientras no se haya llegado al final de ninguno de los dos, se comparan los elementos actuales. Si el elemento de A es menor que el de B, se agrega el elemento de A al resultado y se avanza el puntero i, si el de B es menor, se agrega el de B y se avanza j. Si ambos elementos son iguales, se agrega uno solo (ya que tienen el mismo valor y podría ser tanto a como b) y se avanzan ambos punteros. Al final del recorrido, si alguno de los conjuntos todavía tiene elementos que no se tomaron en cuenta (si algún puntero es menor que el largo de su conjunto), se agregan todos esos elementos restantes al conjunto. De esta manera la complejidad pasa a ser O(n + m), ya que se recorre una sola vez cada conjunto.

Intersección

Para la intersección, al principio pensé en una solución similar a la de la primera unión, usando dos bucles anidados para comparar todos los pares posibles entre los conjuntos A y B. Esto implicaba una complejidad de O(n * m), ya que se evalúa cada combinación. Sin embargo, decidí aplicar la misma idea de los punteros que usé para la unión. Como los conjuntos están ordenados, nunca debemos que volver ya que sabemos que los valores más grandes a otros están más adelante. Entonces si el elemento de A es menor que el de B, se avanza i, y si el de B es menor, se avanza j. Solo cuando los elementos coinciden se agregan al conjunto resultado y ahí avanzamos ambos punteros. Este procedimiento también recorre cada conjunto solo una vez, teniendo un tiempo de orden O(n + m).

Fuentes que utilicé:

https://www.geeksforgeeks.org/union-and-intersection-of-two-sorted-arrays-2/