0.1 Resolución:

Para la resolución se cuenta inicialmente con una K-partición con sus elementos (nodos) distribuidos de alguna forma, la llamaremos *solucion*. Lo que se va a proceder a hacer es tratar de mejorar el peso total de esta K-partición (con peso total nos referimos a la suma de los pesos de todos los conjuntos de dicha partición). Reubicando sus nodos de manera que se minimice dicho peso. Para esto y siguiendo la idea de búsqueda local se construirán varias posibles soluciones, vecinos, y nos quedaremos con el mejor de ellos (el mejor es aquel, cuyo peso total sea el mínimo de entre los vecinos que constituyen la vecindad). Para la resolución, empezaremos tomando al primero de los conjuntos que constituyen a *solucion*. Una vez tomado, se trabajara con los nodos dentro del mismo y de a par, siempre que estos sean adyacentes. Si no hay nodos adyacentes en el conjunto, entonces se devolverá como solucion y vecino, a *solucion* (no se producen cambios). En caso de existir al menos un par de nodos adyacentes. Evaluaremos el peso que se genera entre la arista que une a este par y el peso de las aristas a los nodos adyacentes dentro del conjunto actual, *peso_combinacion*. Calculado este último peso, nos quedaremos con aquel cuyo valor es máximo, *mayor_peso*. Es decir, podemos pensarlo como encontrar la componente conexa con mas peso dentro del conjunto. Una vez hallado este par de nodos, a los que llamaremos a uno *nodo_A* y otro *nodo_B*, crearemos otra K-partición, *nuevo_vecino*. El mismo sera una copia de la *solucion* actual, solo que no contara con los nodos denominados *nodo_A* y *nodo_B*. Y lo que se va a proceder a hacer es a reubicar a ambos nodos en aquel o aquellos conjuntos tal que el peso final de *nuevo_vecino* sea mínimo. Y tendremos una posible solución. Finalizado esto, se procede a operar de la misma manera pero, ahora trabajando con el siguiente conjunto. De esta forma vamos a obtener un nuevo vecino. Luego de haber trabajado con los k conjuntos elegiremos a aquel vecino de todos los formados cuyo peso total sea el mínimo. Como *solucion* es hasta el momento la solución para el problema, se comparara si el peso del vecino elegido es menor que este. Si no lo es, no se pudo obtener una mejor ubicación que la que ya se tenia para los nodos y se finaliza el algoritmo siendo *solucion* la solucion a nuestro problema. En caso de serlo *solucion* pasara a ser el vecino seleccionado. Y se comenzara a ejecutar nuevamente lo propuesto inicialmente, pero ahora con la K-partición *solucion* modificada. Para poder seguirla optimizando.

```
procedure REACOMODAR(..solucion, total de conjuntos,..)
    suma_solucion ← sumar el peso de la solucion
    if (solucion no usa los k conjuntos) then
        agregar conjuntos restantes vacios a solucion
    end if
    vecino_solucion ← crear una nueva k_particion
    posible_solucion ← −1
    while (se pueda mejorar suma_solucion) do
        for i= 1,...,total de conjuntos do
            if (solucion[i] tiene mas de un nodo) then
                nuevo_vecino ← crear una nueva k_particion
                peso_vecino ← crear_vecino(...,i, ,solucion, vecino_actual,..)
            end if
            if (peso_vecino < posible_solucion o posible_solucion == −1) then
                posible_solucion ← peso_vecino
                vecino_solucion ← nuevo_vecino
            end if
        end for
        if (posible_solucion < suma_solucion) then
            suma_solucion ← posible_solucion
            solucion ← vecino_solucion
            posible_solucion ← −1
        else
            return solucion
        end if
    end while
    Return solucion
end procedure
```

```
procedure CREAR_VECINO(..i, solucion, vecino_actual..)
    for cada combinacion entre par de nodos adyacente dentro de solucion[i] do
        nodo_A ← tomar un nodo de solucion[i]
        nodo_B ← tomar un nodo adyacente a nodo_A en solucion[i]
        peso_combinacion ← sumar el peso de la arista entre el nodo_A y el nodo_B y el peso de las aristas de los nodos adyacentes a nodo_A y nodo_B dentro de solucion[i]
        mayor_peso ← el mayor peso_combinacion
        separo_A ← nodo_A con mayor peso_combinacion
        separo_B ← nodo_B con mayor peso_combinacion
    end for
    nuevo_vecino ← solucion
    sacar a los nodos separo_A y separo_B del conjunto nuevo_vecino[i]
    ubicar a los nodos separo_A y separo_B entre los conjuntos de nuevo_vecino de manera de minimizar el peso total
    return peso total de nuevo_vecino
end procedure
```

tanto solucion[i] como nuevo_vecino[i] hacen referencia al i-esimo conjunto de cada K-partición. Con los conjuntos enumerados de 1 al total de conjuntos.

0.2 Análisis de complejidad:

Para representar la K-partición *solucion*, se utilizo un vector de tuplas. donde la segunda componente emula un conjunto, por medio de un vector de int, y la primera a la suma del peso de dicho conjunto. El algoritmo cuenta con dos funciones principales que son las mencionadas *reacomodar* y *crear vecino*, como el mismo se ejecuta hasta que ya no se pueda mejorar al vector *solucion*. Analizaremos la complejidad de cada iteración.

1) Complejidad de *reacomodar*:
a) Como se trabaja a partir de una soluciónón dada, la misma quizás no utiliza la cantidad total de conjuntos con que dispone. Entonces se procede a completar al vector *solucion* con las tuplas restantes. O(k)
b) Se crear un vector vacío, que almacenara al mejor de los vecinos. O(1) y se comienza con las mencionadas iteraciones.
c)Por cada conjunto de *solucion* se llama a la función crear vecino. Es decir se la llama k veces (siendo k la cantidad de conjuntos).
d) Cada vez que se obtiene un *nuevo_vecino* y su peso, se verifica si es menor que el peso de los anteriores vecinos o si es el primer vecino creado. En caso de serlo se guarda este peso 0(1) y tambien a *nuevo_vecino* en *vecino_solucion* O(k+n) (n cantidad de nodos en total). Como esto se realiza a lo sumo para todos los conjuntos tenemos O(k*(k+n))
e) Finalizada estas iteraciones, se verifica si el peso de *vecino_solucion* es menor que el peso de *solucion* en caso de serlo se guarda el peso y se reemplaza a *solucion* O(k+n). Y se vuelve a realizar una nueva iteración.

2) Complejidad de *Crear vecino*:
a) Esta función comienza por buscar a los pares de nodos que sean adyacentes y calculando su *peso_combinacion*. Para obtener todas las combinaciones se emplearon dos for anidados, el primero de ellos va desde 1 al total de nodos en el conjunto que se esta utilizando, denominaremos t a esta cantidad con $0 <= t < n$. El segundo comienza desde el siguiente nodo que se este utilizando en el primero de los for hasta t. De esta forma obtenemos todas las combinaciones. Pero, por cada una hay que sumar el peso de la arista que los une y el peso de sus adyacentes. Para calcular este último, se vuelve a recorrer a todos los nodos y si son adyacentes se suma el peso de la arista. El mismo se encuentra en una matriz a la que podemos acceder en O(1). De esta manera se realizan, para generalizar, dos for de 1 hasta t. y en cada iteracion una suma de t elementos en total O(t^3). Comparando cada *peso_combinacion* realizado para guardarnos el máximo O(1).
b) Procedemos a copiar al vector *solucion* a *nuevo_vecino* O(k + n), y a retirar del i-esimo conjunto (i-esima tupla) de este último vector a *nodo_A* y *nodo_B* O(t). Y se actualiza la primer componente con la suma correspondiente. Que es la cantidad de aristas, en el peor de los casos O(t^2).
c) Para encontrar la mejor ubicación del par de nodos es necesario ver todos las combinaciones con los k

conjuntos y ubicarlo donde el peso total generado sea el mínimo que con alguna otra combinación. Para representar esto, se crearan dos vectores de longitud k (cada posición hace referencia a un conjunto de *nuevo_vecino*). El primero posee en cada posición el peso que se adiciona al meter al *nodo_A* en cada conjunto y el segundo al adicionado por insertar a *nodo_B*. De esta manera, se busca la combinacion de estos vectores donde el valor de la posición de uno mas la del otro sea la mínima O(k^2). Y se guarda las posiciones elegidas.
d) Por último se inserta al par de nodos en las posiciones elegidas dentro de *nuevo_vecino* y se calcula el peso total de este vector O(k)

El costo total de esta función es por lo tanto O(t^3) + O(k + n) + O(t) + O(t^2)+ O(k^2) + O(k) = O(t^3) + O(k^2) + O(n)
t es como máximo igual a n. Acotamos t por n y obtenemos:
O(n^3 + k^2)
Pero esta función como se menciono en el punto 1.c es llamada k veces entonces tenemos:
O(k*(n^3 + k^2))
Agregando los costos de la función *reacomodar* tenemos en total:
O(k) + O(k*(n^3 + k^2)) + O(k*(k + n)) + O(k + n) = O(k*n^3 + k^3)