Valentina Jiménez:

Código: 201924116

Correo: lv.jimenez@uniandes.edu.co

Mario Ricaurte:

Código: 201922994

Correo: m.ricaurte@uniandes.edu.co

Análisis de complejidad

Requerimiento 1:

La complejidad del primer requerimiento es de O(log(n)), esto se debe a que menos de la mitad de los elementos que entran a la función son descartados y no se opera sobre ellos ya que no cumplen la condición.

Requerimiento 2:

Este algoritmo tiene una complejidad de O(nlog), siendo n la cantidad de vértices en el grafo. Esto es así porque el algoritmo recorre cada vértice, pregunta por la cantidad de arcos que tiene, recopila la cantidad de arcos en una lista y ordena la lista con el algoritmo *Quicksort* de mayor a menor. El recorrido de cada vértice tiene una complejidad de O(n). No obstante, n > nlogn para n>10, mientras que n < nlogn para n <10, y como se espera que el grafo tendrá más de 10 vértices, la complejidad final del algoritmo es la del proceso de ordenamiento, pues es mayor.

Requerimiento 3:

La función tiene una complejidad de O(log(n)) ya que divide los términos usados en menos de la mitad. Sin embargo, utiliza Dijkstra para encontrar la ruta con menor peso. Por tanto, teniendo en cuenta el uso de este algoritmo, la complejidad es O(n).

Requerimiento 4:

Este algoritmo tiene una complejidad de O((V + E)logV), siendo V la cantidad de vértices y E la cantidad de arcos. Esto se debe al algoritmo *Prim* que encuentra la red de expansión mínima. No obstante, dentro del algoritmo nos encontramos con un recorrido de todos los vértices, por lo tanto esta sección tiene una complejidad de O(V) y también nos encontramos con una función que ordena con *Quicksort* estos vértices según el peso de los arcos, así que tenemos una complejidad de O(VlogV). Finalmente, la complejidad total del algoritmo es la del algoritmo de *Prim*, pues (V + E)logV > VlogV cuando E > 0, y previamente ya se expuso que VlogV > V cuando V>10, así que finalmente obtenemos que (V + E)logV > VlogV > V.

Requerimiento 5:

Este algoritmo tiene una complejidad de O(nlogn), puesto que se utiliza ordena una lista de n vértices con el algoritmo *Quicksort*. No obstante, para obtener esta lista hay un ciclo con complejidad O(n), pues se recorren todos los vértices, en el peor caso. En general, este ciclo será más corto, pues es la

cantidad de arcos de un vértice determinado, y si bien es posible que un vértice esté conectado a todos los demás, no es el caso en este grafo y quizá esa ocurrencia no es muy común. Finalmente, hay otro ciclo y otra función de ordenamiento con *Quicksort* dentro, pero esta parte solo ordena los arcos de cada vértices, y solo recorre los vértices afectados, así que tiene una complejidad muy baja, pero asumiendo en un peor caso en el que todo está conectado a todo, su complejidad sería la de la multiplicación del ciclo por el algoritmo de ordenamiento: $O(n^2 \log n)$. No obstante, en este grafo nunca se presenta esa complejidad y es por esto que la complejidad final del algoritmo es de $O(n \log n)$, por el *Quicksort* grande.