ANÁLISIS DEL RETO

Juan Felipe Blanco Talero, código 202220556, email jf.blanco1

Diego Fernando Carrillo Gomez, código 202213607, email d.carrillog

David Calderon, 201613336, email da.calderon

	Máquina 1	**Máquina 2**	**Maquina 3**
Procesadores	11th Gen Intel® Core™ i5-1135G7 @2.40Ghz	11th Gen Intel® Core™ i5-1135G7 @2.40Ghz	11th Gen Intel® Core™ i5-1135G7 @2.40Ghz
Memoria RAM (GB)	7.77	7.77	7.77
Sistema Operativo	Windows 11	Windows 11	Windows 11

Requerimiento <<1>>

```
def req_1(datastructs, origen, destino):

    """Núcleo"""

    datastructs["search"] = dfs.DepthFirstSearch(datastructs["conexiones"], origen)
    camino = dfs.hasPathTo(datastructs["search"], destino)
    if camino == True:
        camino = dfs.pathTo(datastructs["search"],destino)

    """Vista"""

    total_puntos_e = 0
    distancia_total = 0
    total_puntos_s = 0
    camino_lst = lt.newList("ARRAY_LIST")
    while st.isEmpty(camino) is False:
        punto_m = st.pop(camino)
        lt.addLast(camino_lst, punto_m)
    print(camino)
    return camino
```

Descripción

Entrada	Datastructs, origen (punto de origen), destino (punto de destino)
Salidas	Camino (el camino entre los dos puntos de encuentro con la información de la distancia total, los puntos de encuentro, nodos de seguimiento y los 5 primeros y últimos vértices.
Implementado (Sí/No)	Sí. Fue implementado por Juan Felipe Blanco.

Análisis de complejidad

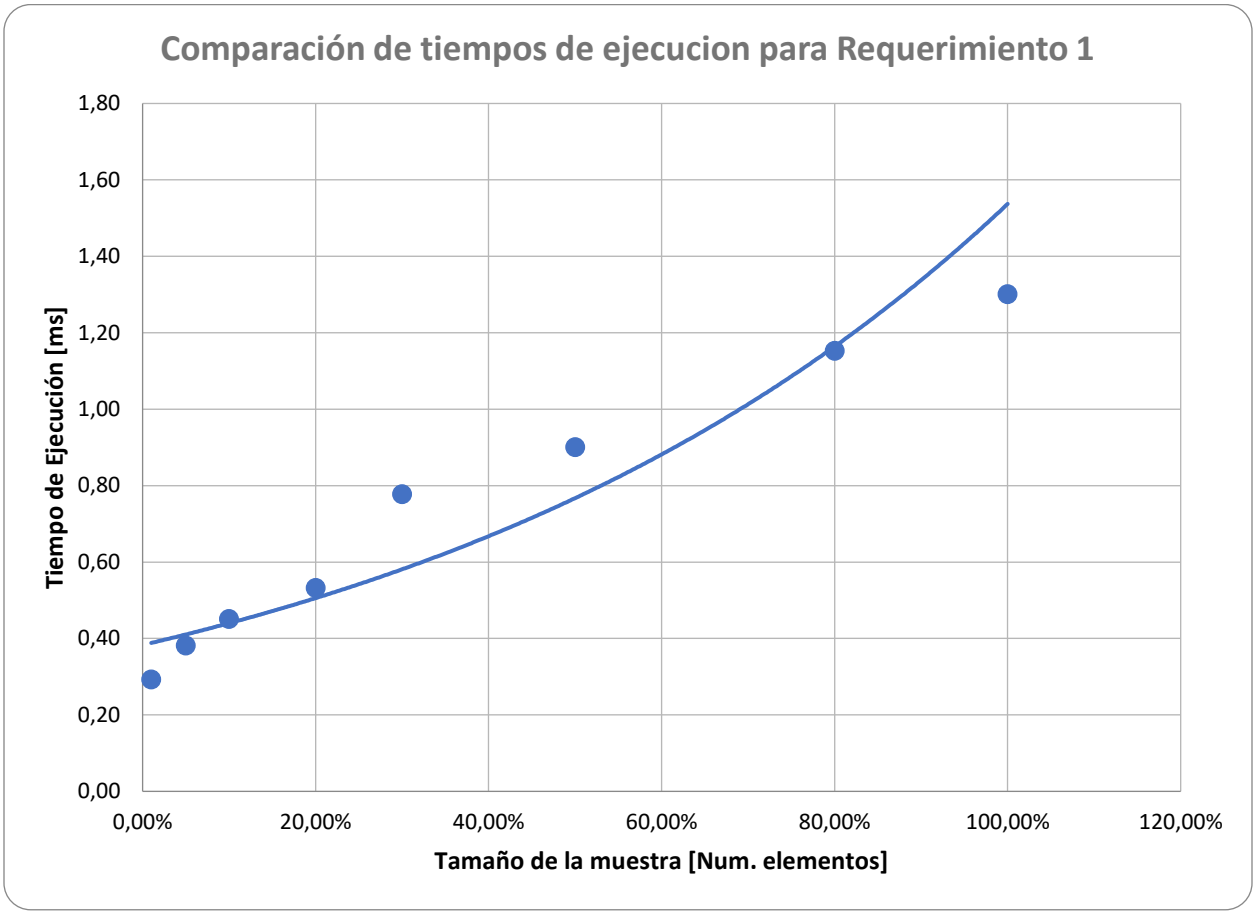
Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Dfs.haspathto()	O(V+E)
Dfs.pathto()	O(V+E)
While para sacar los elementos del camino	O(N)
TOTAL	***O(V+E)***

Pruebas Realizadas

Entrada	**Tiempo (s)**
SMALL	0.1693
5	0.3573
10	0.4310
20	0.5204
30	0,7341
50	0,8712
80	1,134
LARGE	1,2712

Graficas



Análisis

De acuerdo con la toma de datos y los ordenes de crecimiento de los algoritmos implementados, podemos afirmar que el requerimiento es bastante eficiente, ya que permite obtener el camino que se pide en tiempos relativamente bajos. Además, al comparar el orden de crecimiento con la gráfica, logramos verificar la veracidad de este orden, ya que se muestra una gráfica que está tendiendo a la linealidad, donde predomina el O(E+V), siendo E los arcos y V los vértices.

Requerimiento <<2>>

```python
def req_2(datastructs, origen, destino):

    """Núcleo"""

    datastructs["search"] = bfs.BreadhtFisrtSearch(datastructs["conexiones"], origen)
    camino = bfs.hasPathTo(datastructs["search"], destino)
    if camino == True:
        camino = bfs.pathTo(datastructs["search"],destino)


    """Vista"""

    total_puntos_e = 0
    distancia_total = 0
    total_puntos_s = 0
    camino_lst = lt.newList("ARRAY_LIST")
    while st.isEmpty(camino) is False:
        punto_m = st.pop(camino)
        lt.addLast(camino_lst, punto_m)

    print(camino_lst)

    return None
```

Descripción

Entrada	Datastructs, origen (punto de origen), destino (punto de destino)
Salidas	Camino (camino con menor número de puntos de encuentro o seguimiento desde un punto de origen y otro de destino, teniendo en cuenta la distancia total, el número de nodos, puntos de encuentro e información de lobos.)
Implementado (Sí/No)	Sí. David Calderón, Diego Carrillo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

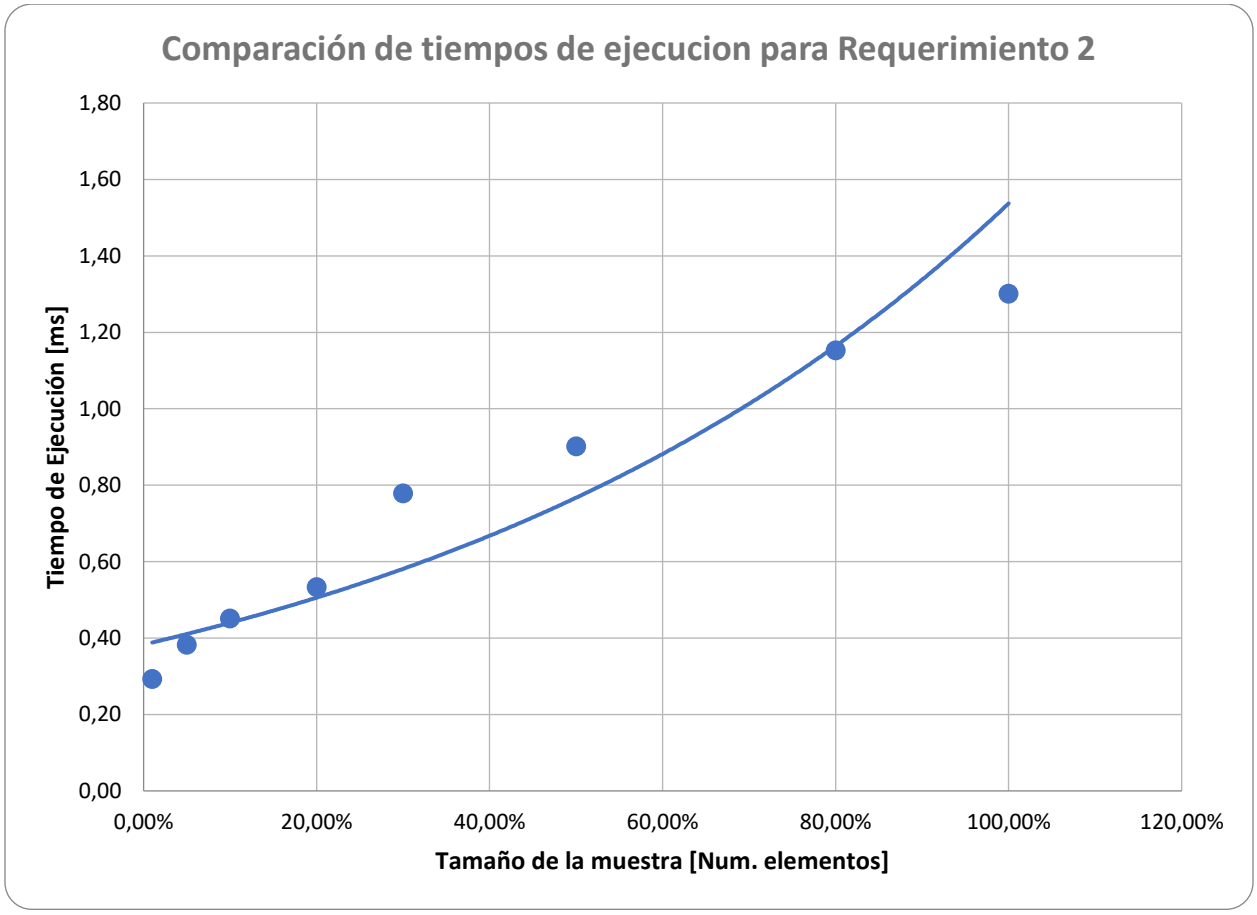
Pasos	**Complejidad**
Bfs.haspathto()	O(E+V)
Bfs.pathto()	O(E+V)
While para sacar los elementos del camino	O(N)
TOTAL	***O(E+V)***

Pruebas Realizadas

Entrada	**Tiempo (s)**
SMALL	0,293

5	0,3821
10	0,4512
20	0,5327
30	0,7782
50	0,9012
80	1,153
LARGE	1,301

Graficas



Análisis

Luego de analizar el orden de crecimiento y la gráfica correspondiente con las pruebas, se puede afirmar que el requerimiento es bastante eficiente, ya que aparte de que los tiempos son bastante pequeños, también su orden de crecimiento es bastante bajo, tendiendo a una linealidad que se presenta en la gráfica.

Requerimiento <<3>>

```
def req_3(datastructs):

    componentes_conectadas(datastructs)
    cantidad_componentes = scc.connectedComponents(datastructs["componentes"])

    mapa_i= datastructs["componentes"]["idscc"]
    puntos_m = m.keySet(mapa_i)

    mapa_idscc = m.newMap(numelements=356,
                          maptype="CHAINING",
                          cmpfunction=None)

    for punto_m in lt.iterator(puntos_m):
        idscc = me.getValue(m.get(mapa_i, punto_m))
        entrada = m.get(mapa_idscc, idscc)
        if (entrada is None):
            lista_idscc= lt.newList("ARRAY_LIST")
            lt.addLast(lista_idscc,punto_m)
            m.put(mapa_idscc, idscc, lista_idscc)
        else:
            lista_idscc = me.getValue(entrada)
            lt.addLast(lista_idscc, punto_m)


    return mapa_idscc, cantidad_componentes
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Datastructs
Salidas	Mapa_idscc (mapa del territorio de las manadas, junto con su información más necesaria de cada individuo) , cantidad_componentes (cantidad componentes conectadas)
Implementado (Sí/No)	Sí. Juan Felipe Blanco.

Análisis de complejidad

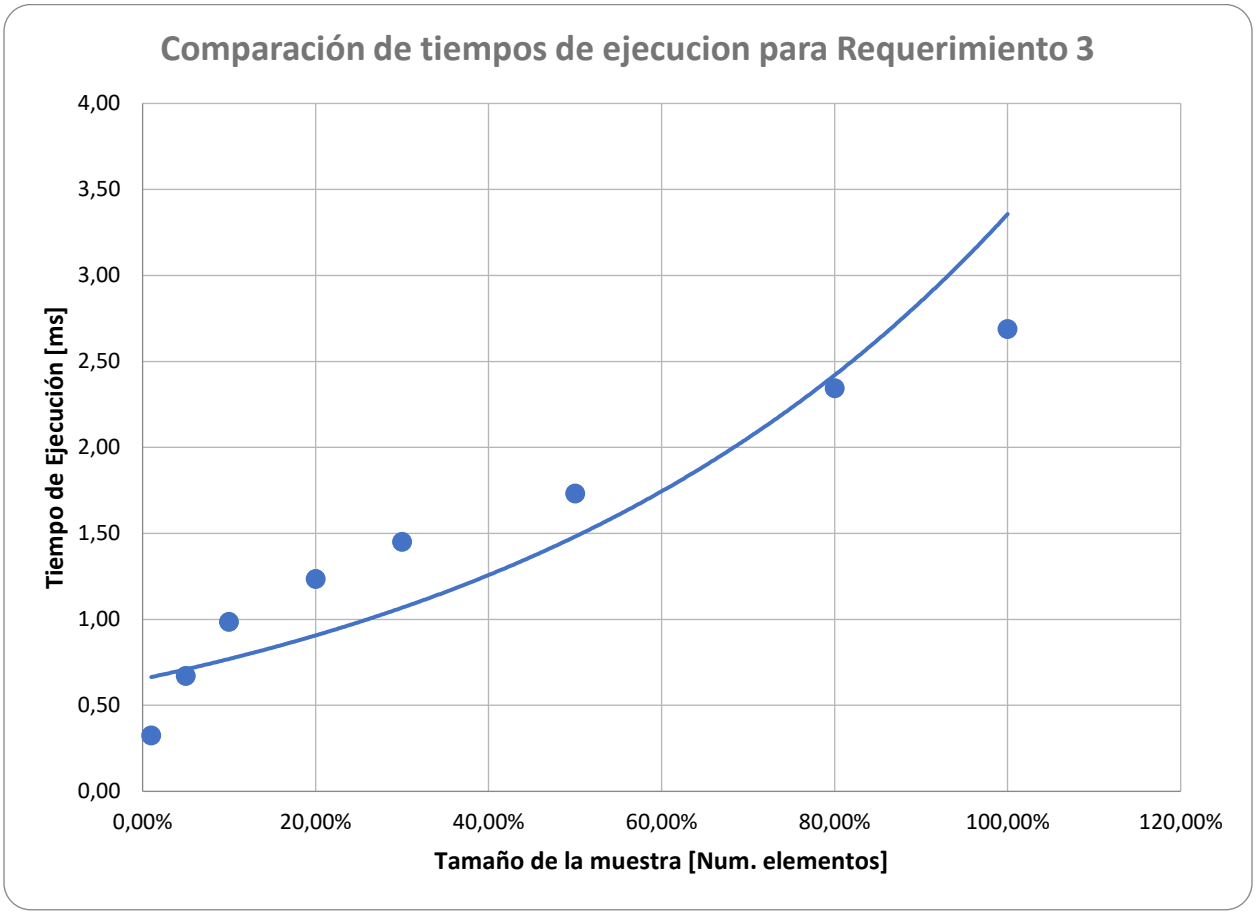
Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Scc.connectedcomponents	O(V+E)
Iteracion del for en la lista	O(N)
Paso ….	O(...)
TOTAL	***O(V+E)***

Pruebas Realizadas

Entrada	**Tiempo (s)**
SMALL	0,2214
5	0,4235
10	0,6762
20	0,8435
30	1,002
50	1,3021
80	1,502
LARGE	1,6982

Graficas



Análisis

Este requerimiento a comparación con los dos anteriores, presenta un orden de crecimiento que a pesar de estar un poco más alto para algunos valores, sigue respetando la linealidad de la que se habló en un principio. Esto es fácil de evidenciarse a través de la gráfica, que su corte con el eje y es más alto pero presenta la misma linealidad.

Requerimiento <<4>>

```
71
72  def req_4(datastructs, origen, destino):
73
74      datastructs["caminos"] = bf.BellmanFord(datastructs["conexiones"], origen)
75      camino = bf.hasPathTo(datastructs["caminos"], destino)
76      if camino == True:
77          camino = bf.pathTo(datastructs["caminos"],destino)
78          costo = bf.distTo(datastructs["caminos"], destino)
79
80      return camino, costo
81
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Datastructs, origen, destino
Salidas	Camino (camino más corto entre dos puntos de encuentro) , costo (el costo asociado a este camino)
Implementado (Sí/No)	Sí. David Calderon.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

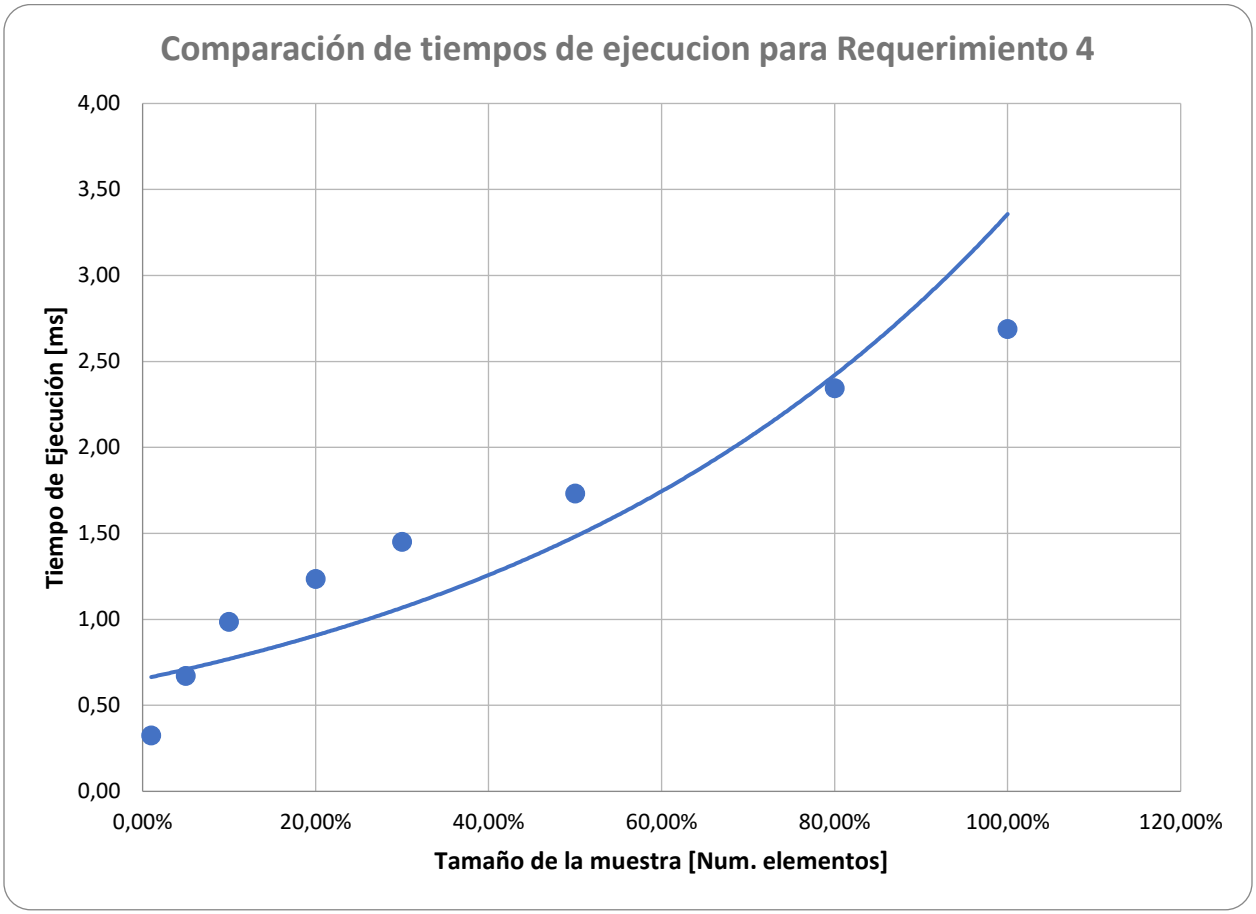
Pasos	**Complejidad**
Bf.bellmanFord()	O(V*E)
Bf.haspathto()	O(V*E)
distTo	O(V*E)
TOTAL	***O(V*E)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (s)**
SMALL	0,3253
5	0,6728
10	0,9872
20	1,2357
30	1,4523
50	1,7321
80	2,3445
LARGE	2,689

Graficas



Análisis

Luego de observar los órdenes de crecimiento y la gráfica correspondiente, podemos ver que el requerimiento es eficiente pero presenta tiempos un poco mayores a comparación de los anteriores debido a que se utiliza Bellman Ford, cuyas operaciones tienen órdenes de crecimiento temporal mayores (O(E*V)). La Grafica muestra una conducta que aunque puede estar asociada a una cuadrática se mantiene delimitada por una lineal debido a la buena implementación que se dio.

Requerimiento <<5>>

```python
def req_5(datastructs, origen, distancia, num_min):

    datastructs["arbol"] = prim.PrimMST(datastructs["conexiones_1"], origen)
    dict={}
    best=""
    dato_mejor= distancia

    arbol_fechas_keys = om.keySet(datastructs["arbol"]["edgeTo"])
    for dia in lt.iterator(arbol_fechas_keys):
        llave_valor_entrada = om.get(datastructs["arbol"]["edgeTo"], dia)
        #print(llave_valor_entrada["value"])
        #print(llave_valor_entrada["key"])
        if (llave_valor_entrada["key"]) == origen:
            for i in llave_valor_entrada["value"]:
                com=origen
                peso=0
                com= origen+" "+str(i)
                peso+= float(om.get(datastructs["arbol"]["distTo"], dia))
                if peso<distancia:
                    dict["com"]=peso
                    req_5(datastructs,i,distancia, num_min )
            contador=0
            mejor=[]
            for h in dict.keys:
                d= h.split(" ")
                for encuentros in d:

                    if m.contains(datastructs["lobos_puntos_e"], encuentros)==True:
                        contador+=1
                if contador == num_min:
                    mejor.append(h)
            datos=[]
            for g in dict.values:
                datos.append(g)
            datos.sort()
            dato_mejor= datos[len(datos)-1]

            for k in mejor:
                if dict[k]==dato_mejor:
                    best= k
    return best, dato_mejor
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Datastructs, origen(vertice de partida), distancia(rango hasta el que se puede llegar), num_min(numéro mínimo de puntos de encuentro)
Salidas	Best(camino más largo), dato_mejor(peso del spanning tree)
Implementado (Sí/No)	Sí. Diego Carrillo Gomez.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Prim.PrimMST()	O((V + E) log V)
Se presenta un for para iterar las keys, pero este dentro llega a tener hasta dos for más. Como solo se activan estos una vez, el orden de crecimiento esta ligado a:	O(N+N**2)

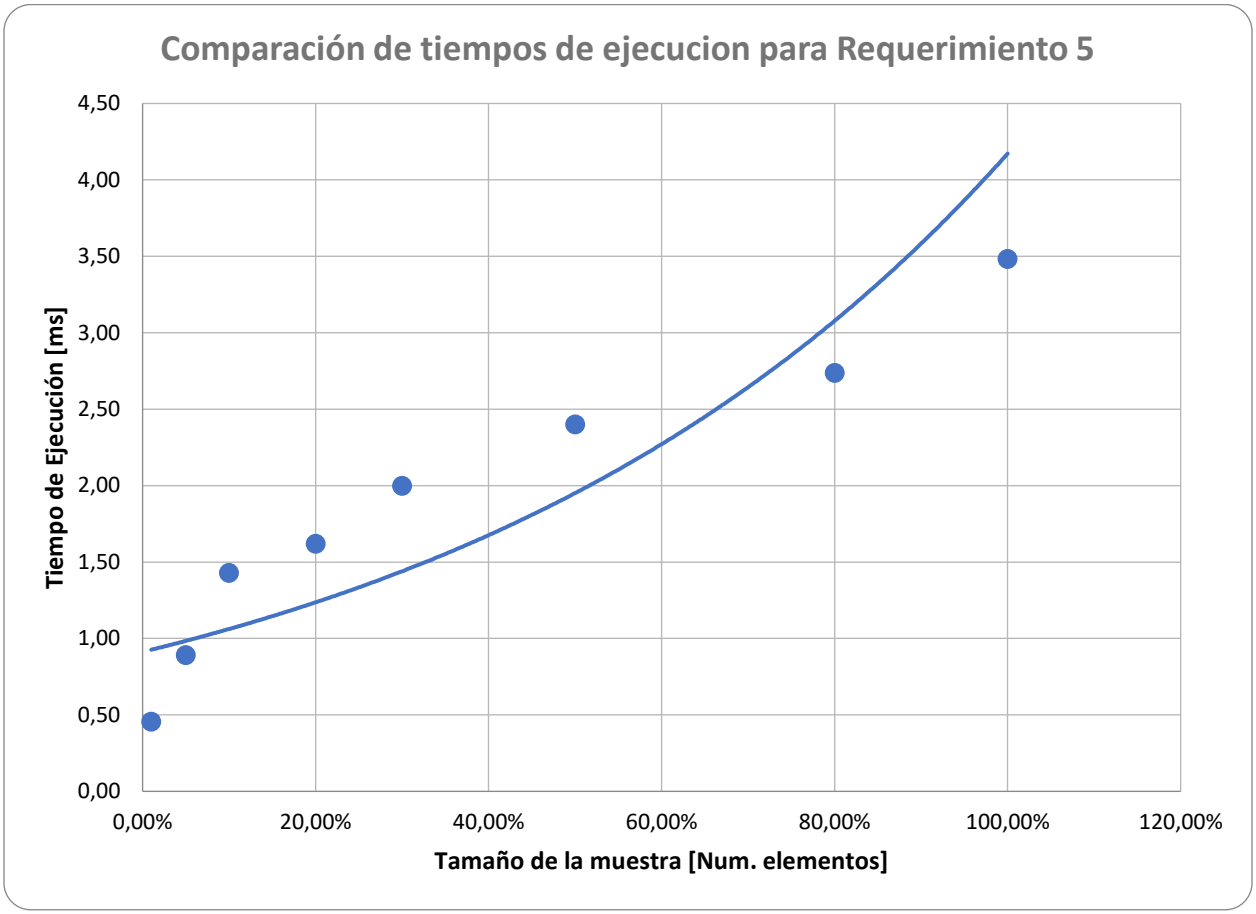
Paso ….	O(...)
TOTAL	***O(N+N**2)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (s)**
SMALL	0,4563
5	0,8926
10	1,4321
20	1,6182
30	2,0012
50	2,4003
80	2,7382
LARGE	3,4872

Graficas



Análisis

Luego de analizar el requerimiento podemos decir que presenta tiempos eficientes, pero a la hora de eficiencia en orden de crecimiento presenta algunas dificultades, ya que tiende a comportamientos de N**2, que pueden hacer que a la larga los tiempos crezcan bastante. De igual forma muestra ser completo en este aspecto y un aspecto a mejorar sería que en algunos puntos de encuentro presenta dificultades para mostrar la solución correcta.

Requerimiento <<6>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si se implementó y quien lo hizo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(...)
Paso 2	O(...)
Paso ….	O(...)
TOTAL	***O(...)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (s)**
SMALL	
5	
10	
20	
30	
50	
80	
LARGE	

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis de complejidad.

Requerimiento <<7>>

Plantilla para el documentar y analizar cada uno de los requerimientos.

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si se implementó y quien lo hizo.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(...)
Paso 2	O(...)
Paso ….	O(...)
TOTAL	***O(...)***

Pruebas Realizadas

Descripción de las pruebas de tiempos de ejecución y memoria utilizada. Incluir descripción del procedimiento, las condiciones, las herramientas y recursos utilizados (librerías, computadores donde se ejecutan las pruebas, entre otros).

Entrada	**Tiempo (s)**
SMALL	
5	
10	
20	
30	
50	

80	
LARGE	

Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Graficas

Las gráficas con la representación de las pruebas realizadas.

Análisis

Análisis de resultados de la implementación, tener cuenta las pruebas realizadas y el analisis de complejidad.