Departamento de Ingeniería de Sistemas y Computación
Estructuras de Datos y Algoritmos
ISIS-1225

ANÁLISIS DEL RETO

Tomás Aguilera, 202321962, t.aguilera

Miguel Angel Velandia, 202312487, ma.velandia2

Juan Felipe Saenz, 202311148, jf.saenz

Requerimiento 1

Descripción

```python
def req_1(data_structs,pais,nivel,num_elem):
    """
    Función que soluciona el requerimiento 1
    """
    #Usamos el mapa con todos los paises y sus trabajos
    paises = data_structs["por_pais"]
    #Conseguimos solo las ofertas del pais buscado
    ofertas_pais = mp.get(paises,pais)
    #Usamos solo el valor de la llave, el cual es un dicc
    ofertas_filtradas = me.getValue(ofertas_pais)
    total_ofertas_pais = lt.size(ofertas_filtradas["all_jobs"])

    if lt.size(ofertas_filtradas[nivel]) > num_elem:
        lista = lt.subList(ofertas_filtradas[nivel],1,num_elem)
    else:
        lista = ofertas_filtradas[nivel]
    n = lt.size(lista)

    if n >10:
        respuesta = lt.newList("ARRAY_LIST")
        prim_5 = lt.subList(lista,1,5)
        ult_5 = lt.subList(lista,n-4,5)
        for job in lt.iterator(prim_5):
            lt.addLast(respuesta,job)
        for job in lt.iterator(ult_5):
            lt.addLast(respuesta,job)
        return respuesta,lt.size(ofertas_filtradas[nivel]),total_ofertas_pais
    else:
        return lista,lt.size(ofertas_filtradas[nivel]),total_ofertas_pais
```

Este requerimiento devuelve las ofertas con un determinado nivel de experticia que tiene un país y muestra las 5 primeras y 5 últimas de las N ofertas que pidió el usuario.

Entrada	Estructura de datos, el país a buscar, el nivel de las ofertas y la cantidad de ofertas que quiere ver.
Salidas	La cantidad de ofertas de un nivel de país y experiencia que el usuario pidió organizadas de más a menos recientes.
Implementado (Sí/No)	Si. Implementado por grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Obtener la tupla llave-valor con la información del país (mp.get())	O(1)
Obtener solo el valor (me.getValue())	O(1)
Reducir la lista a la cantidad de trabajos que quiere ver el usuario	O(C)
Reduce la lista si es necesario para solo mostrar las primeras y últimas 5 ofertas (if n >10…)	O(1)
TOTAL	***O(C)***

c = cantidad de trabajos solicitados por el usuario

El peor de los casos sería c = (total de trabajos en el país) - 1 en donde su complejidad sería O(c)

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una máquina con las siguientes especificaciones. Los datos de entrada fueron:

Cuántas ofertas quiere ver: 110.000

Escriba el código del país: PL

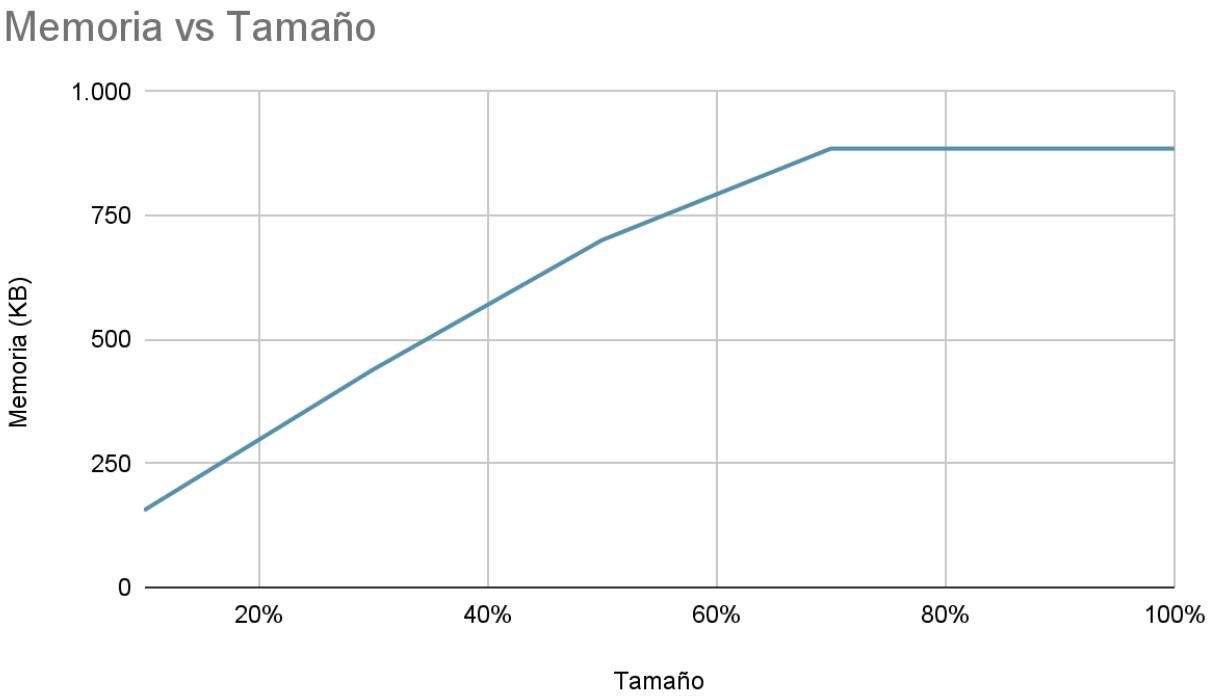
Escriba el nivel de experticia: mid

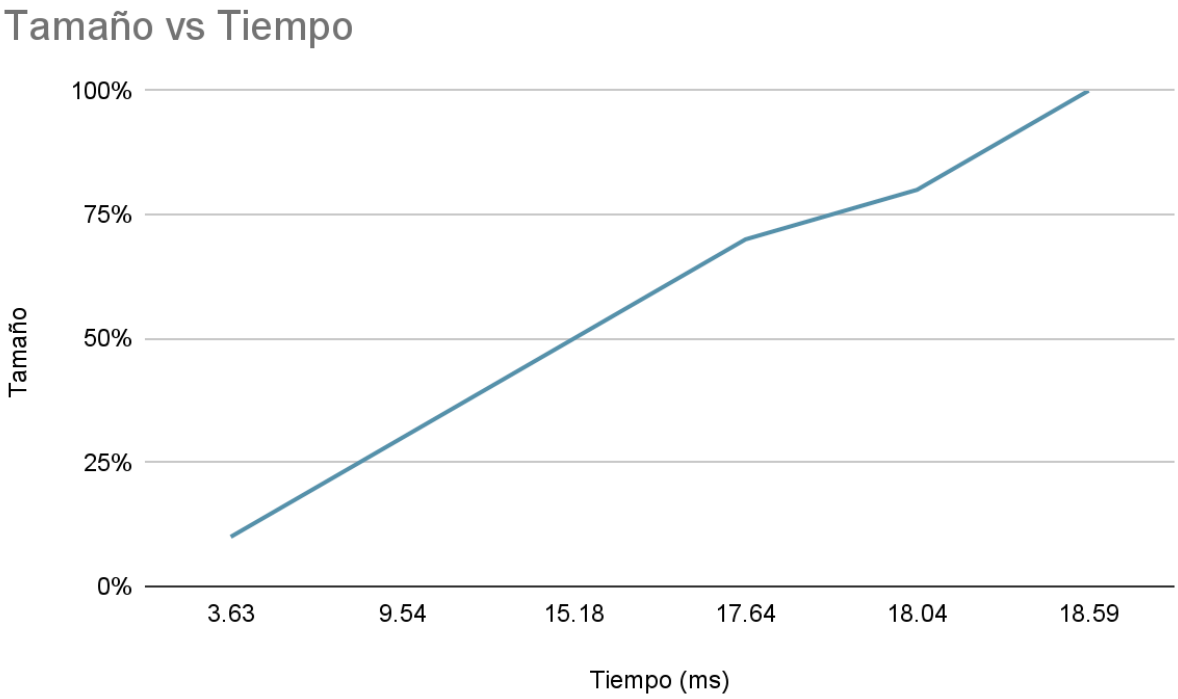
*(El cual es el peor de los casos)

Procesadores	**AMD Ryzen 5 5600 X**
Memoria RAM	32 GB
Sistema Operativo	Windows 11 Pro

Entrada	Tiempo (ms)	Memoria (KB)
10 pct	3.63	155.38
30 pct	9.54	439.19
50 pct	15.18	700.50
70 pct	17.64	885.25
80 pct	18.04	885.25
large	18.59	885.25

Gráficas


Memoria vs Tamaño
1.000
750
500
250
0
Memoria (KB)
20%
40%
60%
80%
100%
Tamaño


Tamaño vs Tiempo
100%
75%
50%
25%
0%
Tamaño
3.63
9.54
15.18
17.64
18.04
18.59
Tiempo (ms)

Análisis

Los resultados si coinciden con la esperada función lineal, sin embargo, consideramos que esta función será constante a partir del peor de los casos y que en el caso promedio también lo será puesto a que las diferencias que hay son mínimas, sin embargo, si la base de datos aumentara entonces el comportamiento lineal se seguiría observando.

En cuanto a la memoría, tiene sentido que a partir de los peores casos esta se mantenga constante, ya que va a tener que seguir almacenando los mismos datos.

Requerimiento 2

```python
def req_2(data_structs,empresa,ciudad,num_ofertas):
    """
    Función que soluciona el requerimiento 2
    """
    empresas = data_structs["por_empresa"]

    empresa_dupla = mp.get(empresas,empresa)
    info_empresa = me.getValue(empresa_dupla)
    ofertas = info_empresa["ofertas"]

    ofertas_filtradas = lt.newList("ARRAY_LIST")

    for job in lt.iterator(ofertas):
        if job["city"] == ciudad:
            lt.addLast(ofertas_filtradas,job)
    if lt.size(ofertas_filtradas) > num_ofertas:
        ofertas_filtradas = lt.subList(ofertas_filtradas,1,num_ofertas)

    n = lt.size(ofertas_filtradas)
    if n >10:
        respuesta = lt.newList("ARRAY_LIST")
        prim_5 = lt.subList(ofertas_filtradas,1,5)
        ult_5 = lt.subList(ofertas_filtradas,n-4,5)
        for job in lt.iterator(prim_5):
            lt.addLast(respuesta,job)
        for job in lt.iterator(ult_5):
            lt.addLast(respuesta,job)
        return respuesta,n
    else:
        return ofertas_filtradas,n
```

Descripción

Busca los trabajos de cierta empresa en una ciudad específica. Devuelve una lista con los primeros y últimos 5 trabajos de la cantidad solicitada por el usuario.

Entrada	Las estructuras de datos, el nombre de la empresa, nombre de la ciudad, cantidad de trabajos a ver.
Salidas	La cantidad de trabajos que cumplen las condiciones

Implementado (Sí/No)	Si. Implementado por el grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Acceder a los trabajos de una empresa (mp.get())	O(1)
Recorrer la lista de esos trabajos	O(t)
Reducir la lista a la cantidad de trabajos que quiere el usuario (subList())	O(c)
TOTAL	***O(t + c)***

t = Trabajos de una empresa en la ciudad especificada

c = cantidad de trabajos que pide el usuario

En el peor de los casos que sería que c = t-1, se recorre prácticamente la misma lista dos veces lo que hace que ese sea el nivel de complejidad.

Si c >= t, la complejidad sería O(t)

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).

Cuántas ofertas quiere ver: 170

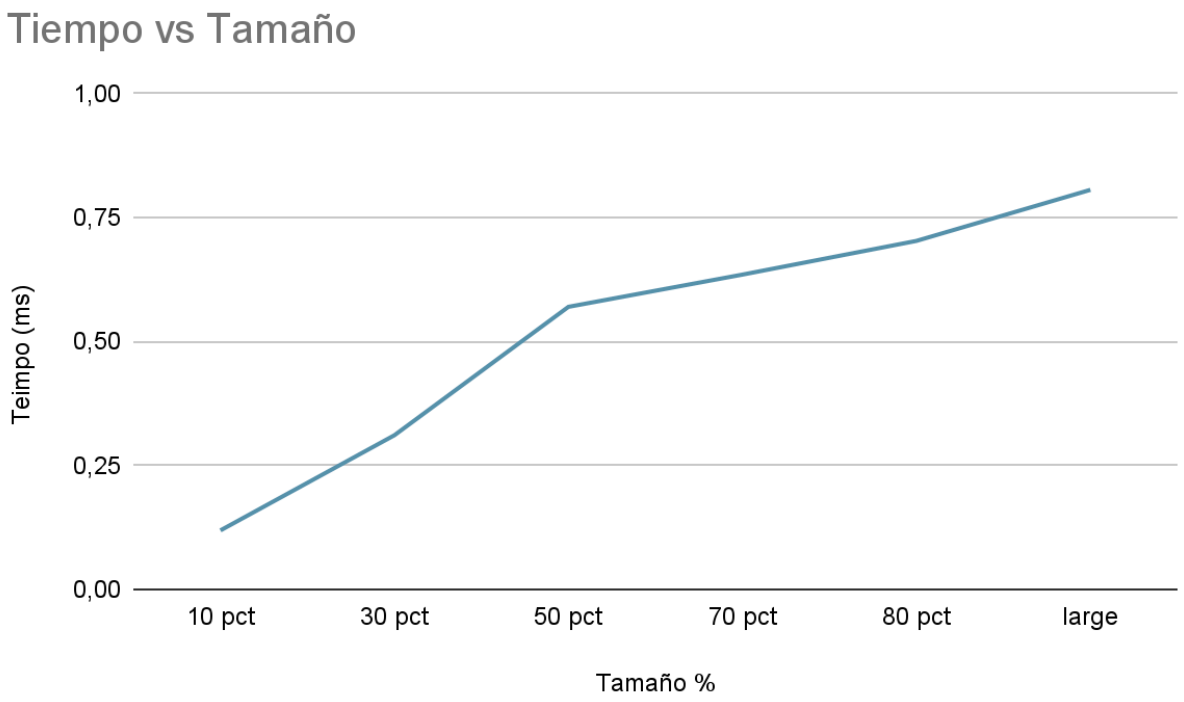
Escriba el nombre de la empresa: nokia

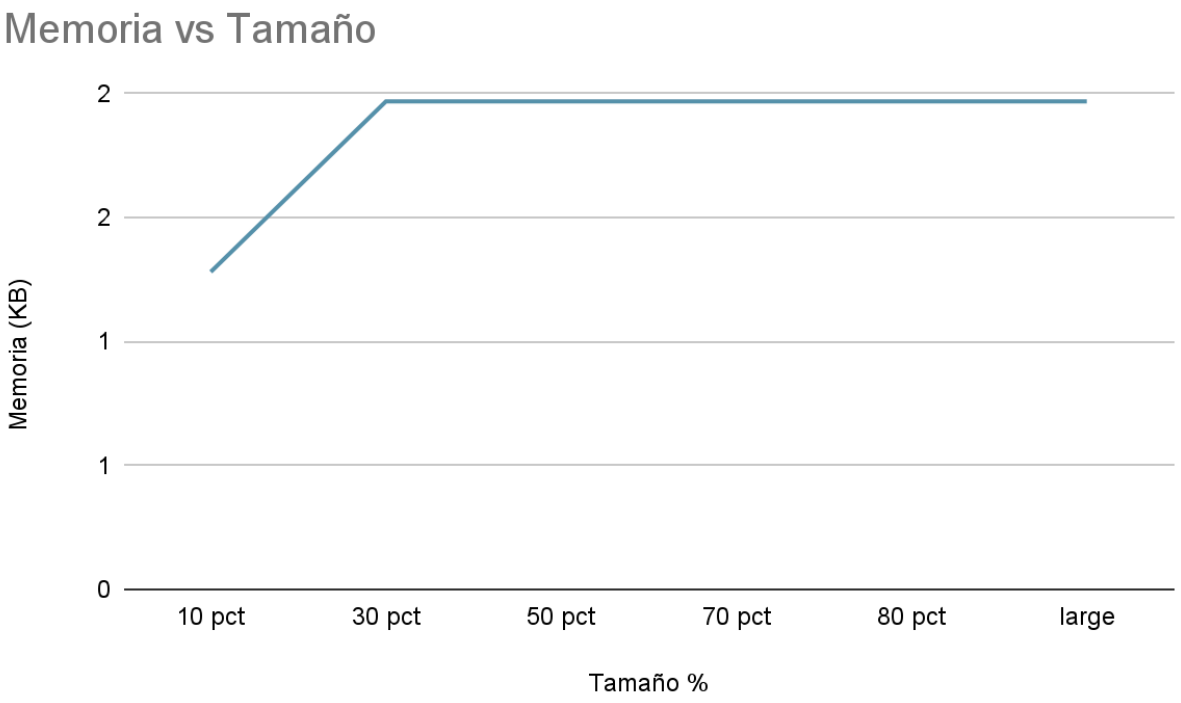
Escriba la ciudad de la empresa: Warszawa

Procesadores	**AMD Ryzen 5 5600 X**
Memoria RAM	32 GB
Sistema Operativo	Windows 11 Pro

Entrada	Tiempo (ms)	Memoria (KB)
10 pct	0.119	1.28
30 pct	0.311	1.96
50 pct	0.570	1.96
70 pct	0.635	1.96
80 pct	0.703	1.96
large	0.806	1.96

Gráficas





Análisis

Los datos coinciden con un comportamiento lineal en caso de que c >= t, que fue el caso de prueba que se usó.

Según el uso de la memoria, a partir del 30% ya se encontraban los datos que se necesitan para este requerimiento, a partir de hay, como no se necesitan más datos, la memoria permanece constante.

Requerimiento 3

```python
def req_3(data_structs,empresa,fecha_inicial,fecha_final):
    """
    Función que soluciona el requerimiento 3
    """
    try:
        fecha_inicial = d.strptime(fecha_inicial,"%Y-%m-%d")
        fecha_final = d.strptime(fecha_final,"%Y-%m-%d")
    except:
        return -1

    #Conseguimos los trabajos de una empresa
    dupla_empresa = mp.get(data_structs["por_empresa"],empresa)
    empresa = me.getValue(dupla_empresa)
    trabajos = empresa["ofertas"]

    #Creamos una lista para poner los trabajos que cumplan estén en el tiempo determinado
    lista = lt.newList("ARRAY_LIST")
    junior = 0
    mid = 0
    senior = 0

    for trabajo in lt.iterator(trabajos):
        fecha = trabajo["published_at"]
        if fecha >= fecha_inicial and fecha <= fecha_final:
            lt.addLast(lista,trabajo)
            if trabajo["experience_level"] == "junior":
                junior+=1
            elif trabajo["experience_level"] == "mid":
                mid+=1
            elif trabajo["experience_level"] == "senior":
                senior+=1
    #Si la lista tiene más de 10 elementos, se reduce y se toman los primeros 5 y últimos 5
    n = lt.size(lista)
    if n >10:
        respuesta = lt.newList("ARRAY_LIST")
        prim_5 = lt.subList(lista,1,5)
        ult_5 = lt.subList(lista,n-4,5)
        for job in lt.iterator(prim_5):
            lt.addLast(respuesta,job)
        for job in lt.iterator(ult_5):
            lt.addLast(respuesta,job)
        return respuesta, junior,mid,senior,n
    else:
        return lista,junior,mid,senior,n
```

Descripción

Devuelve una lista con los trabajos de una compañía que estén entre un rango de fechas. Además devuelve la cantidad de trabajos de cada nivel de experiencia.

Entrada	Estructuras de datos, nombre de la empresa, fecha inicial y fecha final
Salidas	Una lista con los trabajos que cumplen los requisitos y contadores
Implementado (Sí/No)	Si. Tomás Aguilera

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Conseguir los trabajos de una empresa (mp.get())	O(1)
Recorrer la lista con esos trabajos	O(t)
Sumar a los contadores	O(1)
Reducir la lista a 10 elementos	O(1)
TOTAL	***O(t)***

t = Total de trabajos de una empresa

Pruebas Realizadas

Escriba la empresa: **nokia**

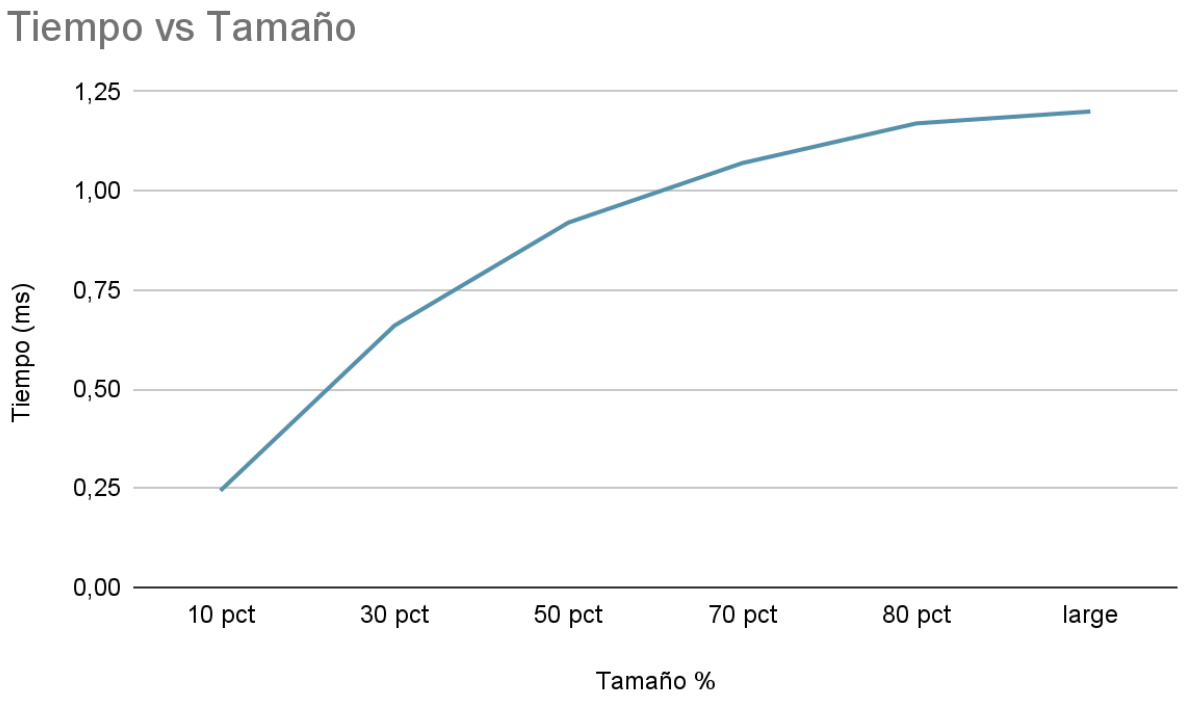
Escriba la fecha inicial (YY-MM-DD): **2000-01-01**

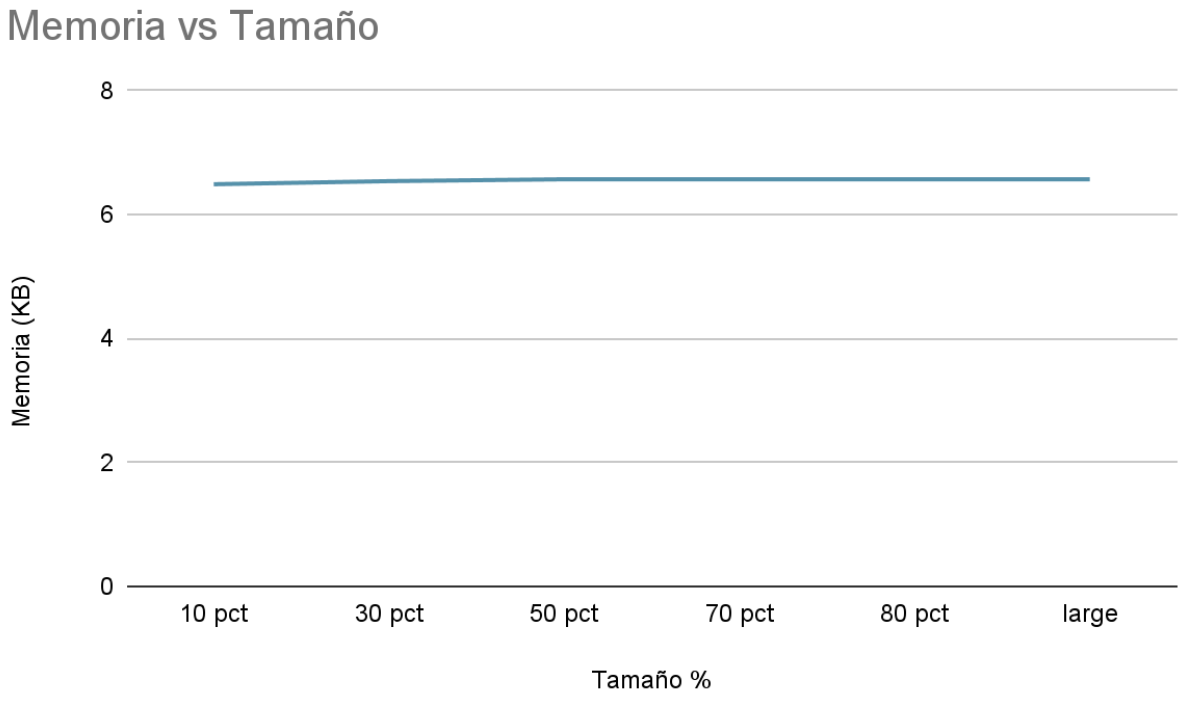
Escriba la fecha final (YY-MM-DD): **2024-01-01**

Procesadores	**AMD Ryzen 5 5600 X**
Memoria RAM	32 GB
Sistema Operativo	Windows 11 Pro

Entrada	**Tiempo (ms)**	**Memoria (KB)**
10 pct	0.244	6.49
30 pct	0.66	6.54
50 pct	0.92	6.57
70 pct	1.07	6.57
80 pct	1.17	6.57
large	1.20	6.57

Gráficas


Tiempo vs Tamaño
1,25
1,00
0,75
0,50
0,25
0,00
Tiempo (ms)
10 pct
30 pct
50 pct
70 pct
80 pct
large
Tamaño %


Memoria vs Tamaño
8
6
4
2
0
Memoria (KB)
10 pct
30 pct
50 pct
70 pct
80 pct
large
Tamaño %

Análisis

Las gráficas y la información coinciden con una función lineal. En este caso los valores tanto de memoria como tiempo son mínimos, pero si se manejara una base de datos más grande, se notaría más fácilmente este comportamiento lineal.

Requerimiento 4

```python
def req_4(data_structs,codigo,fechaInicio,fechaFin):
    # TODO: Realizar el requerimiento 4
    """
    Función que soluciona el requerimiento 4
    """
    try:
        fecha_inicial = d.strptime(fechaInicio,"%Y-%m-%d")
        fecha_final = d.strptime(fechaFin,"%Y-%m-%d")
    except:
        return -1

    paises = data_structs["por_pais"]

    info_pais_dupla = mp.get(paises,codigo)
    info_pais = me.getValue(info_pais_dupla)
    trabajos_pais = info_pais["all_jobs"]

    ofertas_en_pais = lt.newList("ARRAY_LIST")
    empresas_en_pais = set()
    ciudades_en_pais = set()
    ciudades_ofertas = {}
    total_ofertas = 0

    for trabajo in lt.iterator(trabajos_pais):
            if fecha_inicial <= trabajo["published_at"] <= fecha_final:
                lt.addLast(ofertas_en_pais, trabajo)
                empresas_en_pais.add(trabajo["company_name"])
                ciudades_en_pais.add(trabajo["city"])
                if trabajo["city"] not in ciudades_ofertas:
                    ciudades_ofertas[trabajo["city"]] = 1
                else:
                    ciudades_ofertas[trabajo["city"]] +=1
                total_ofertas += 1

    total_ofertas_final=lt.size(ofertas_en_pais)
    total_empresas=len(empresas_en_pais)
    total_ciudades=len(ciudades_en_pais)
    ciudad_mayor_ofertas = llave_max_valor(ciudades_ofertas)
    ciudad_menor_ofertas = llave_min_valor(ciudades_ofertas)
    n = lt.size(ofertas_en_pais)
    if n >10:
        respuesta = lt.newList("ARRAY_LIST")
        prim_5 = lt.subList(ofertas_en_pais,1,5)
        ult_5 = lt.subList(ofertas_en_pais,n-4,5)
        for job in lt.iterator(ult_5):
            lt.addLast(respuesta,job)
        for job in lt.iterator(prim_5):
            lt.addLast(respuesta,job)
        return respuesta,total_ofertas_final,total_empresas,total_ciudades,ciudad_mayor_ofertas,ciudad_menor_ofertas
    else:
        return ofertas_en_pais,total_ofertas_final,total_empresas,total_ciudades,ciudad_mayor_ofertas,ciudad_menor_ofertas
```

Descripción

Entrada	Estructura de datos,código del país, fecha de inicio, fecha de fin
Salidas	Las primeras 5 y últimas 5 ofertas en dicha ciudad, representados en una tabla, la ciudad donde se hicieron mas ofertas con su contador, la ciudad donde se hicieron menos ofertas y su contador, el total de empresas que ofertó, el total de ciudades donde se ofertó
Implementado (Sí/No)	Si, Implementado por Juan Felipe Sáenz

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Conseguir las ofertas de un país	O(1)
Recorrer la lista de los trabajos de la ciudad	O(t)
Ordenar las ofertas y filtrar segun lo requerido	O(1)
TOTAL	***O(t)***

Pruebas Realizadas

Procesadores	**12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz**
Memoria RAM	8 GB
Sistema Operativo	Windows 11 Home

Caso de prueba:

código: PL

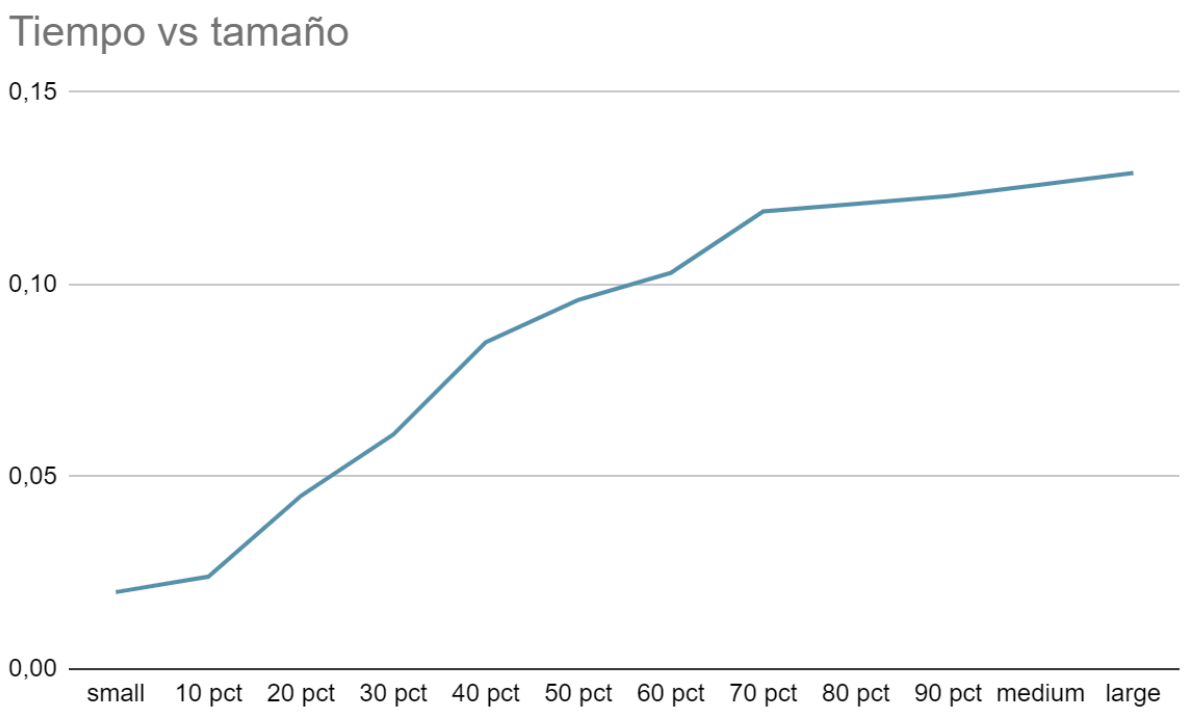
Fecha inicio: 2020-01-01

Fecha fin: 2022-12-31

Entrada	**Tiempo (s)**
small	0.020
10 pct	0.024
20 pct	0.045
30 pct	0.061
40 pct	0.085
50 pct	0.096
60 pct	0.103
70 pct	0.119
80 pct	0.121
90 pct	0.123

medium	0.126
large	0.129

Graficas



Análisis

Vemos que la gráfica se comporta como una función lineal, esto va de la mano con el análisis de complejidades realizado arriba, donde se pudo concluir que la complejidad de este requerimiento es o(t), por lo que coinciden tanto los valores teóricos como los experimentales

Requerimiento 5

```python
def req_5(data_structs, ciudad,fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5
    try:
        fecha_inicial = d.strptime(fecha_inicial,"%Y-%m-%d")
        fecha_final = d.strptime(fecha_final,"%Y-%m-%d")
    except:
        return -1

    #Conseguimos los trabajos de una ciudad
    dupla_ciudad = mp.get(data_structs["por_ciudad"], ciudad)
    ciudad = me.getValue(dupla_ciudad)
    trabajos = ciudad["all_jobs"]

    #Creamos una lista para poner los trabajos que cumplan estén en el tiempo determinado
    lista = lt.newList("ARRAY_LIST")
    empresas_ofertas = {}
    for trabajo in lt.iterator(trabajos):
        fecha = trabajo["published_at"]
        if fecha >= fecha_inicial and fecha <= fecha_final:
            lt.addLast(lista,trabajo)
            if trabajo["company_name"] not in empresas_ofertas:
                empresas_ofertas[trabajo["company_name"]] = 1
            else:
                empresas_ofertas[trabajo["company_name"]] +=1

    empresa_menor, mayor = llave_max_valor(empresas_ofertas)
    empresa_mayor, menor = llave_min_valor(empresas_ofertas)
    n_empresas = len(empresas_ofertas)


    #Si la lista tiene más de 10 elementos, se reduce y se toman los primeros 5 y últimos 5
    n = lt.size(lista)
    if n > 10:
        respuesta = lt.newList("ARRAY_LIST")
        prim_5 = lt.subList(lista,1,5)
        ult_5 = lt.subList(lista,n-4,5)
        for job in lt.iterator(ult_5):
            lt.addLast(respuesta,job)
        for job in lt.iterator(prim_5):
            lt.addLast(respuesta,job)

        return respuesta, n_empresas, empresa_mayor, mayor, empresa_menor, menor,n
    else:
        return lista, n_empresas, empresa_mayor, mayor, empresa_menor, menor,n
```

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Estructuras de datos, ciudad, fecha inicial y fecha final
Salidas	Una lista con los trabajos que cumplen los requisitos y contadores
Implementado (Sí/No)	Sí, lo implementó Miguel Velandia

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Conseguir los trabajos de una ciudad (get)	O(1)
Recorrer la lista con esos trabajos	O(n)
Sumar a los contadores	O(1)
Reducir la lista a 10 elementos	O(1)
TOTAL	***O(n)***

n = trabajos de una ciudad

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).

Escriba el nombre de la ciudad: Warszawa

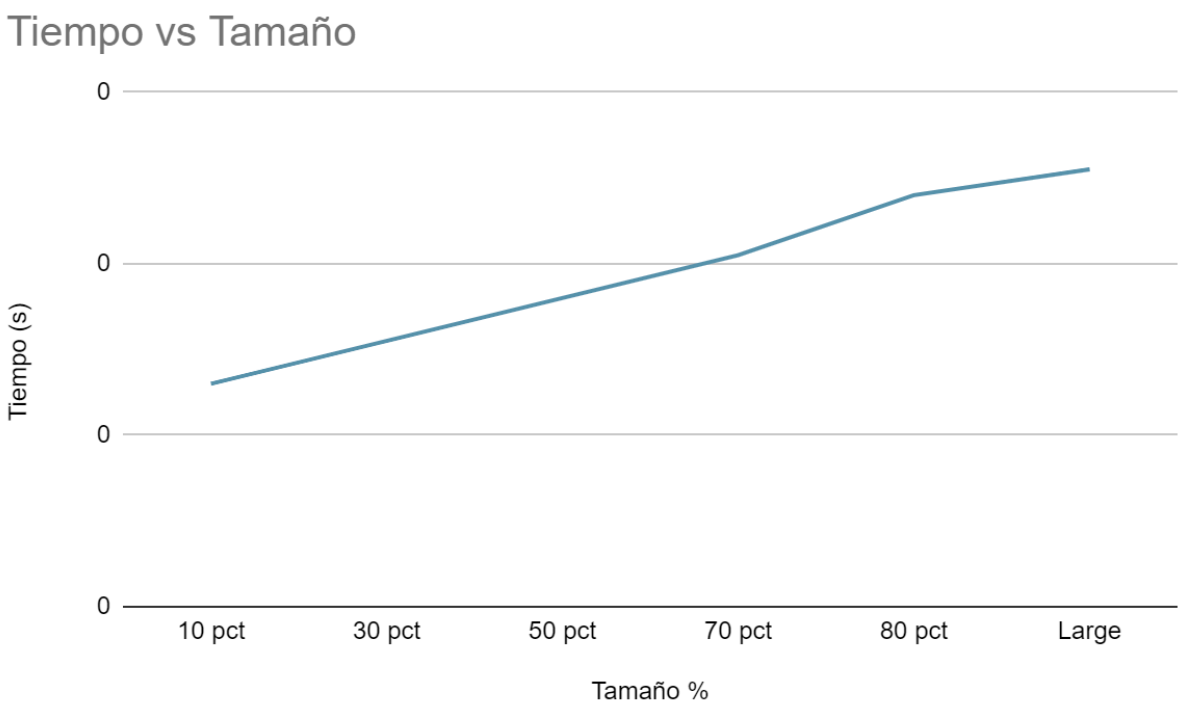
Escriba la fecha inicial (YY-MM-DD): **2000-01-01**

Escriba la fecha final (YY-MM-DD): **2024-01-01**

Entrada	Tiempo (s)
10 pct	0.026
30 pct	0.031
50 pct	0.036
70 pct	0.041
80 pct	0.048
large	0.059

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La gráfica muestra lo esperado, teniendo en cuenta los datos que manejamos parece una constante, pero al aumentar los datos la gráfica se va convirtiendo a una gráfica lineal.

Requerimiento 6

```python
def req_6(data_structs,nivel,anio,num_ciudades):
    """
    Función que soluciona el requerimiento 6
    """

    ciudades = mp.keySet(data_structs["por_ciudad"])
    ciudades_que_cumplen = lt.newList("ARRAY_LIST")


    if nivel == "indiferente":
        for ciudad in lt.iterator(ciudades):
            info_ciudad_dupla = mp.get(data_structs["por_ciudad"],ciudad)
            info_ciudad = me.getValue(info_ciudad_dupla)

            #verificamos si una ciudad tiene ofertas en el año
            existe_anio = mp.contains(info_ciudad["years"],anio)
            if existe_anio:
                ofertas_por_year = mp.get(info_ciudad["years"],anio)
                info_ofertas_por_year = me.getValue(ofertas_por_year)
                if lt.size(info_ofertas_por_year["all_jobs"]) != 0:
                    lt.addLast(ciudades_que_cumplen,{"ofertas":lt.size(info_ofertas_por_year["all_jobs"]),"ciudad":info_ciudad["name"],"lista":info_ofertas_por_year["all_jobs"]})

    else:
        #conseguimos la info de la ciudad
        for ciudad in lt.iterator(ciudades):
            info_ciudad_dupla = mp.get(data_structs["por_ciudad"],ciudad)
            info_ciudad = me.getValue(info_ciudad_dupla)

            #verificamos si una ciudad tiene ofertas en el año
            existe_anio = mp.contains(info_ciudad["years"],anio)
            if existe_anio:
                ofertas_por_year = mp.get(info_ciudad["years"],anio)
                info_ofertas_por_year = me.getValue(ofertas_por_year)
                if lt.size(info_ofertas_por_year[nivel]) != 0:
                    lt.addLast(ciudades_que_cumplen,{"ofertas":lt.size(info_ofertas_por_year[nivel]),"ciudad":info_ciudad["name"],"lista":info_ofertas_por_year[nivel]})

    sort(ciudades_que_cumplen,compSize)
    ward = False
    if lt.size(ciudades_que_cumplen)>0:
        ciudad_mayor = lt.firstElement(ciudades_que_cumplen)
        ciudad_menor = lt.lastElement(ciudades_que_cumplen)

        ciudad_mayor= {"ciudad":ciudad_mayor["ciudad"],"ofertas":ciudad_mayor["ofertas"]}
        ciudad_menor= {"ciudad":ciudad_menor["ciudad"],"ofertas":ciudad_menor["ofertas"]}
        ward = True

    if lt.size(ciudades_que_cumplen) < num_ciudades:
        lista_final = ciudades_que_cumplen

    else:
        lista_final = lt.subList(ciudades_que_cumplen,1,num_ciudades)

    total_ciudades_que_cumplen = lt.size(lista_final)
    total_ofertas_que_cumplen = 0
    total_empresas_que_cumplen = 0

    final = lt.newList("ARRAY_LIST")

    for key in lt.iterator(lista_final):
        lista = key["lista"]
        promedio_total = 0
        total_emp = {}
        mejor_oferta = {"info":None,"cantidad":0}
        peor_oferta = {"info":None,"cantidad":float("inf")}

        for job in lt.iterator(lista):
            promedio_salario = (job["salary_from"]+job["salary_to"])/2
            promedio_total += (promedio_salario)
            #Sacamos la mejor oferta
            if job["salary_to"] > mejor_oferta["cantidad"] and job["salary_to"] != 0:
                mejor_oferta["cantidad"] = job["salary_to"]
                mejor_oferta["info"] = job
            #Sacamos la peor oferta
            if job["salary_from"] < peor_oferta["cantidad"] and job["salary_from"] != 0:
                peor_oferta["cantidad"] = job["salary_from"]
                peor_oferta["info"] = job
```

```
        #Añadimos las empresas
        if job["company_name"] not in total_emp:
            total_emp[job["company_name"]] = 1
        else:
            total_emp[job["company_name"]] += 1

    total_ofertas_que_cumplen += lt.size(lista)
    total_empresas_que_cumplen += len(total_emp)

    filtro = ["title","country_code","company_name","experience_level","salary_from","salary_to","id"]

    mejor_oferta["info"] = filtrar_llaves_req6(mejor_oferta["info"],filtro)
    peor_oferta["info"] = filtrar_llaves_req6(peor_oferta["info"],filtro)

    max_empresa = llave_max_valor(total_emp)

    lt.addLast(final,{"ciudad":key["ciudad"],"total_ofertas":lt.size(lista),"promedio_total":promedio_total/lt.size(lista),"total_empresas":len(total_emp),
                      "mayor_empresa":max_empresa,"mejor_oferta":mejor_oferta["info"], "peor_oferta":peor_oferta["info"]})

if ward:
    return final, total_ciudades_que_cumplen, total_empresas_que_cumplen, total_ofertas_que_cumplen, ciudad_mayor, ciudad_menor
else:
    return final, total_ciudades_que_cumplen, total_empresas_que_cumplen, total_ofertas_que_cumplen
```

Descripción

Primero se recorren los trabajos con un nivel de experiencia y en un año de todas la ciudades y se añaden a una lista. Después esta lista es reducida para solo tener en cuenta solo las ciudades que el usuario quiere. Finalmente, usamos los trabajos en cada ciudad para obtener la información que necesitamos.

Entrada	Estructuras de datos, nivel de experiencia, año, número de ciudades sobre las que consultar
Salidas	Las ciudades con más ofertas del nivel de experiencia buscado y datos adicionales sobre la información de las ofertas en cada ciudad.
Implementado (Sí/No)	Si. Implementado por grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Recorrer total de ciudades (81) (keySet)	O(1)
Recorrer las ofertas que hay en una ciudad en determinado año y determinado nivel de experiencia	O(e)
(Recorrer las ofertas que hay en una ciudad en determinado año)(si no se especifica el nivel de experiencia)	O(t)
merge sort de las lista de ofertas que cumplen los requisitos	O(t log t)
Reducir la lista de trabajos al ciudades solicitado por el usuario (subList())	O(c)
Se recorren las ciudades c ciudades que el usuario dijo	O(c)
Se recorren los trabajos dentro de de cada ciudad	O(j)
TOTAL	***O(t log t) ó O(e log e)***

e = trabajos con un nivel de experiencia de el año escogido

t = todos los trabajos del año escogido

*Si es t no puede ser e y viceversa (peor caso siendo t)

c = número de ciudades a mostrar

j = todos los trabajos de las c ciudades

En el peor de los casos, c = 1.033 y t= 107.500 y la ciudad con más trabajos = 20.334 (consultando en el año 2022). Con estos datos empíricos, podemos decir que para el peor caso el tamaño c podría ser despreciado teniendo en cuenta el tamaño de t y pese a que 20.334 no es un valor despreciable, puesto a que es considerablemente grande, el promedio de trabajos por ciudad es por debajo de los 1000 por lo que se podría despreciar. Para un t lo suficientemente grande, t log t dominaría la complejidad total por lo que tomamos esa como la complejidad.

Esta complejidad sirve en este caso dado que c, t y j no adquieren valores tan grandes, sin embargo si la base de datos cambiara y se añadieran más ciudades y trabajos, entonces para calcular la complejidad deberíamos incluir a estas respectivamente.

Pruebas Realizadas

Ingrese el nivel de experticia: indiferente

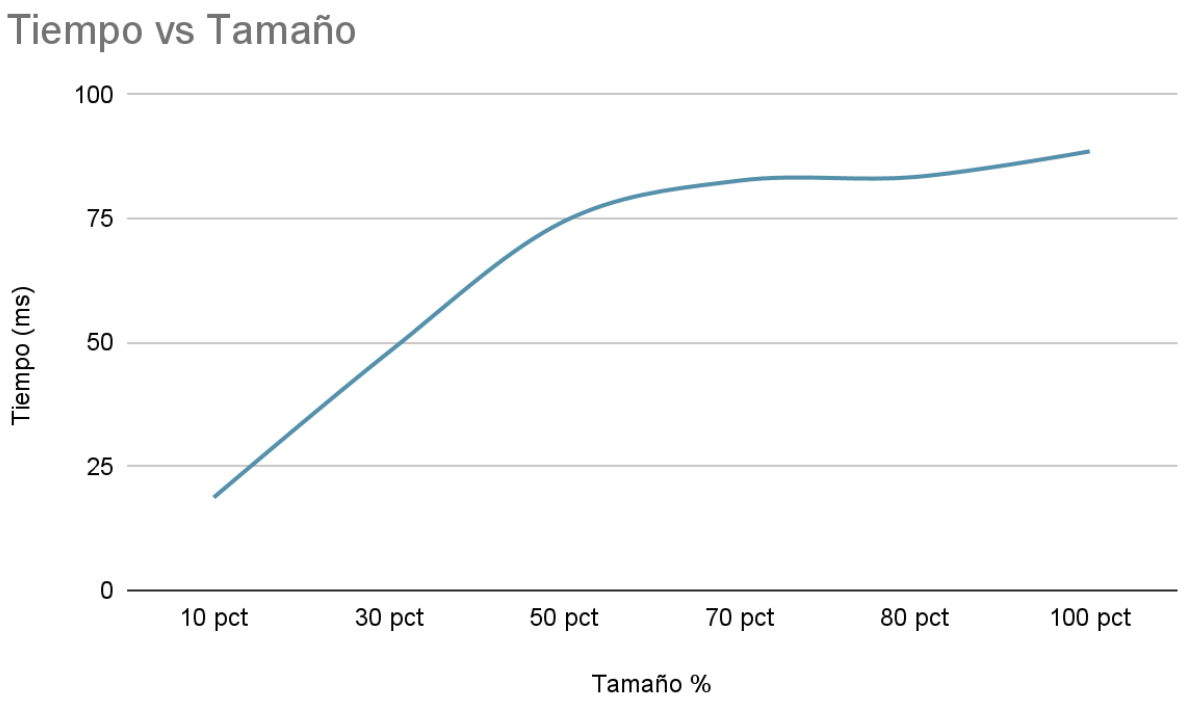
Ingrese el año: 2022

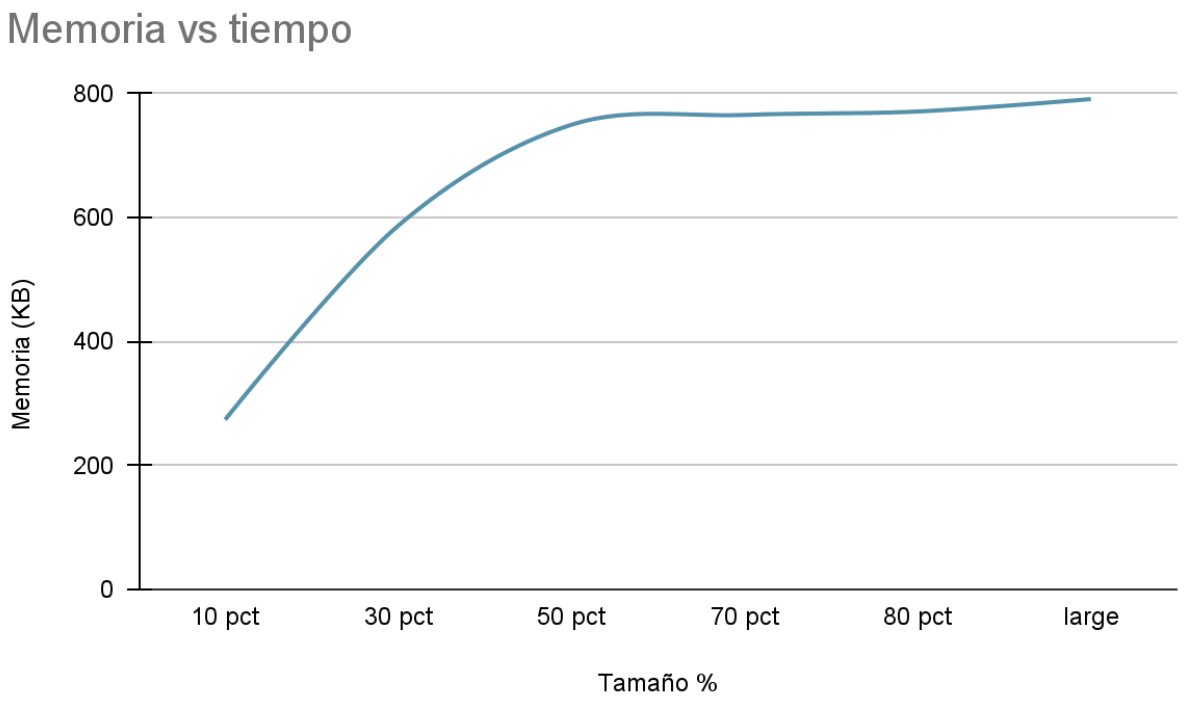
Cuántas ciudades quiere consultar: 1033

Procesadores	**AMD Ryzen 5 5600 X**
Memoria RAM	32 GB
Sistema Operativo	Windows 11 Pro

Entrada	Tiempo (ms)	Memoria (KB)
10 pct	18.74	274.02
30 pct	48.08	587.10
50 pct	74.47	749.84
70 pct	82.66	765.65
80 pct	83.39	771.23
large	88.57	791.27

Gráficas


Tiempo vs Tamaño
100
75
50
25
0
Tiempo (ms)
10 pct
30 pct
50 pct
70 pct
80 pct
100 pct
Tamaño %



Análisis

La gráfica si parece estar en el espacio de una función n log n. Como los datos usados fueron los del peor caso, es posible que en los casos promedios se comporte mucho mejor, sin embargo si se amplía la base de datos este comportamiento lo deberíamos seguir observando.

Requerimiento 7

```python
def req_7(data_structs,anio,mes,num_paises):
    paises = data_structs["por_pais"]
    nombre_paises = mp.keySet(data_structs["por_pais"])

    mapa_paises =  mp.newMap(87,
                             maptype='PROBING',
                             loadfactor=0.9,
                             )
    conteo_paises = {}

    #Recorremos todos los paises
    for pais in lt.iterator(nombre_paises):
        info_pais_dupla = mp.get(paises,pais)
        info_pais = me.getValue(info_pais_dupla)
        trabajos_pais = info_pais["all_jobs"]
        trabajos_cumplen =lt.newList("ARRAY_LIST")
        conteo_ciudades = {}
        #Si los trabajos de una pais están en el rango de fechas se añaden
        for job in lt.iterator(trabajos_pais):
            if job["year"] == anio and job["month"] == mes:
                lt.addLast(trabajos_cumplen,job)
                #Hacemos un conteo de los paises que cumplen las condiciones
                if job["country_code"] not in conteo_paises:
                    conteo_paises[job["country_code"]] = 1
                else:
                    conteo_paises[job["country_code"]] +=1
                #Hacemos un conteo de las ciudades que cumplen las condiciones dentro de cada país
                if job["city"] not in conteo_ciudades:
                    conteo_ciudades[job["city"]] = 1
                else:
                    conteo_ciudades[job["city"]] +=1

        total_ciudades = len(conteo_ciudades)
        ciudad_mayor = llave_max_valor(conteo_ciudades)

        #Agregamos al mapa cada país con su información correspondiente
        mp.put(mapa_paises,info_pais["name"],{"trabajos":trabajos_cumplen, "total_ciudades":total_ciudades,"ciudad_mayor":ciudad_mayor})

    conteo_paises = sortReq7(conteo_paises)

    primeras_n_llaves = {}
    contador = 1
    #Usamos solo los n primeros paises que nos piden
    for llave, valor in conteo_paises.items():
        if contador <= num_paises:
            primeras_n_llaves[llave] = valor
            contador += 1
        else:
            break

    lista_n_paises_cumplen = lt.newList("ARRAY_LIST")
    lista_info_todos_jobs = lt.newList("ARRAY_LIST")

    mayor_pais = {"conteo":0}
    total_ofertas = 0

    junior = {}
    mid = {}
    senior = {}

    cantidad_junior = 0
    junior_promedio_nivel = 0
    junior_ability = {}
    junior_emp = {}
    cantidad_mid = 0
    mid_promedio_nivel = 0
    mid_ability = {}
```

```
mid_emp = {}
cantidad_senior = 0
senior_promedio_nivel = 0
senior_ability = {}
senior_emp = {}

#Añadimos los n países que nos piden con su información correspondiente, esta se consigue con el mapa.
for pais in primeras_n_llaves:
    trabajos_del_pais_dupla = mp.get(mapa_paises,pais)
    info_trabajos_del_pais = me.getValue(trabajos_del_pais_dupla)
    trabajos_del_pais = info_trabajos_del_pais["trabajos"]
    filtro = ["experience_level","ability","level","company_name"]
    trabajos_del_pais = filtrar_llaves(trabajos_del_pais,filtro)

    lt.addLast(lista_n_paises_cumplen,{"pais":pais,"total_ofertas":lt.size(trabajos_del_pais),"total_ciudades":info_trabajos_del_pais["total_ciudades"],
                                       "ciudad_mayor":info_trabajos_del_pais["ciudad_mayor"]})
    lt.addLast(lista_info_todos_jobs,trabajos_del_pais)

    total_ofertas += lt.size(trabajos_del_pais)
    if lt.size(trabajos_del_pais) > mayor_pais["conteo"]:
        mayor_pais["pais"] = pais
        mayor_pais["conteo"] = lt.size(trabajos_del_pais)

    for job in lt.iterator(trabajos_del_pais):
    #job = ['mid', 'FIGMA', '4', 'tp servglobal ltd']
    #job[0] = experience level, job[1]= ability, job[2] = level, job[3] = company name
        if job[0] == "junior":
            cantidad_junior +=1
            junior_promedio_nivel+= job[2]
            if job[1] not in junior_ability:
                junior_ability[job[1]] = 1
            else:
                junior_ability[job[1]] += 1
            if job[3] not in junior_emp:
                junior_emp[job[3]] = 1
            else:
                junior_emp[job[3]] += 1

        elif job[0] == "mid":
            cantidad_mid +=1
            mid_promedio_nivel+= job[2]
            if job[1] not in mid_ability:
                mid_ability[job[1]] = 1
            else:
                mid_ability[job[1]] += 1

            if job[3] not in mid_emp:
                mid_emp[job[3]] = 1
            else:
                mid_emp[job[3]] += 1

        elif job[0] == "senior":
            cantidad_senior +=1
            senior_promedio_nivel+= job[2]
            if job[1] not in senior_ability:
                senior_ability[job[1]] = 1
            else:
                senior_ability[job[1]] += 1

            if job[3] not in senior_emp:
                senior_emp[job[3]] = 1
            else:
                senior_emp[job[3]] += 1

junior["total_ofertas"] = cantidad_junior
junior["promedio_nivel"] = junior_promedio_nivel/cantidad_junior
```

```
junior["mayor_habilidad"] = llave_max_valor(junior_ability)
junior["menor_habilidad"] = llave_min_valor(junior_ability)
junior["total_empresas"] = len(junior_emp)
junior["mayor_empresa"] = llave_max_valor(junior_emp)
junior["menor_empresa"] = llave_min_valor(junior_emp)

mid["total_ofertas"] = cantidad_mid
mid["promedio_nivel"] = mid_promedio_nivel/cantidad_mid
mid["mayor_habilidad"] = llave_max_valor(mid_ability)
mid["menor_habilidad"] = llave_min_valor(mid_ability)
mid["total_empresas"] = len(mid_emp)
mid["mayor_empresa"] = llave_max_valor(mid_emp)
mid["menor_empresa"] = llave_min_valor(mid_emp)

senior["total_ofertas"] = cantidad_senior
senior["promedio_nivel"] = senior_promedio_nivel/cantidad_senior
senior["mayor_habilidad"] = llave_max_valor(senior_ability)
senior["menor_habilidad"] = llave_min_valor(senior_ability)
senior["total_empresas"] = len(senior_emp)
senior["mayor_empresa"] = llave_max_valor(senior_emp)
senior["menor_empresa"] = llave_min_valor(senior_emp)

n = lt.size(lista_n_paises_cumplen)
if n > 10:
    respuesta = lt.newList("ARRAY_LIST")
    prim_5 = lt.subList(lista_n_paises_cumplen,1,5)
    ult_5 = lt.subList(lista_n_paises_cumplen,n-4,5)
    for job in lt.iterator(prim_5):
        lt.addLast(respuesta,job)
    for job in lt.iterator(ult_5):
        lt.addLast(respuesta,job)

    return respuesta, total_ofertas,mayor_pais, junior, mid, senior
else:
    return lista_n_paises_cumplen,total_ofertas,mayor_pais, junior, mid, senior
```

Descripción

Primero se saca una lista en orden ascendente de la cantidad de ofertas de trabajo por país, después se itera sobre los primeros N que el usuario diga y se recorren los trabajos que coincidan con el año y el mes escogidos. Finalmente, se sacan las estadísticas solicitadas para cada nivel de experiencia.

Entrada	La estructura de datos, el año, el mes y la cantidad de países sobre los que sacar los datos.
Salidas	Las estadísticas por nivel de experticia en los N países con más ofertas en determinado año y mes

Implementado (Sí/No)	Si. Implementado por el grupo

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Se recorren los países (81)	O(1)
Se recorren las trabajos de cada país	O(t)
Agregamos al un mapa los trabajos que cumplen	O(1)
Recorremos la cantidad de países que el usuario solicitó	O(c)
Recorremos los trabajos en esos países	O(j)
TOTAL	***O(t+j)***

t = Total de trabajos en un país

c = Total de países solicitados por el usuario (<=81)

j = Total de trabajos en c

Aunque la complejidad de recorrer todos los trabajos en cada país debería ser O(cantidad países * cantidad de trabajos) no asumimos esta complejidad ya que en el peor de los casos sería O(81*cantidad trabajos), por lo que al ser una constante la cantidad de países, no se toma en cuenta para la complejidad.

Pruebas Realizadas

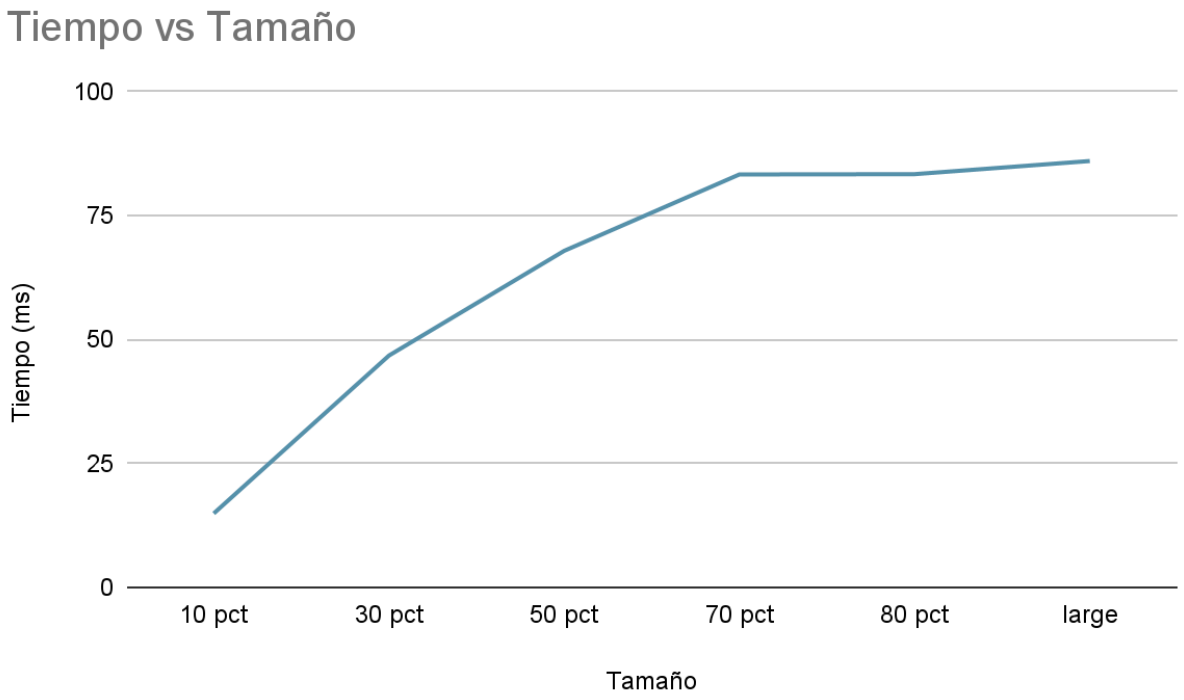
Escriba el número de países a consultar: 100

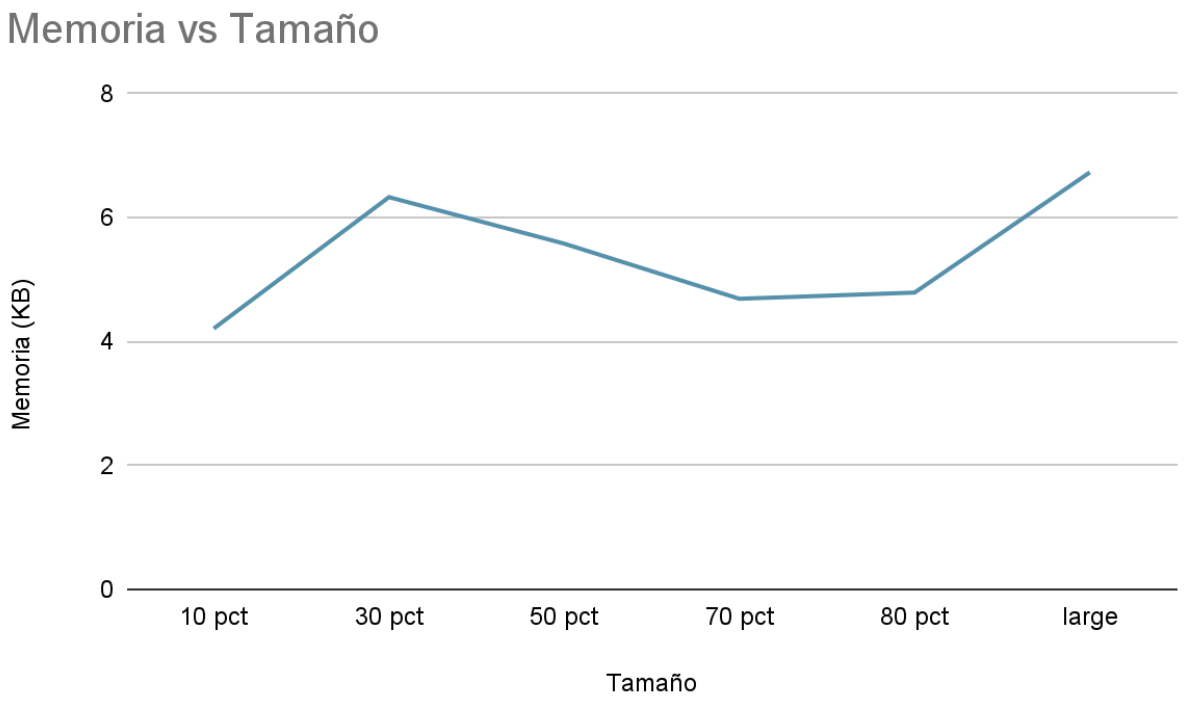
Escriba el año (YYYY): 2022

Escriba el mes (MM): 04

Procesadores	**AMD Ryzen 5 5600 X**
Memoria RAM	32 GB
Sistema Operativo	Windows 11 Pro

Entrada	Tiempo (ms)	Memoria (KB)
10 pct	14.92	4.21
30 pct	46.77	6.33
50 pct	67.89	5.58
70 pct	83.30	4.69
80 pct	83.36	4.79
large	86.01	6.73





Análisis

Las gráficas si parecen indicar un comportamiento lineal como esperado, sin embargo como los datos tienen un límite, al final se empieza a “aplanar” la gráfica. Si la base de datos tuviera más información podríamos apreciar, seguramente, el comportamiento de este algoritmo mejor. Además, los trabajos que cumplen las condiciones y que le dan la complejidad a este algoritmo, no son tantos comparados con el total de trabajos puesto a que estos tienen que ser de un año y mes, por lo tanto, es esperable un buen rendimiento para esta función. En cuanto a la memoria, los picos que hay deben tratarse de procesos que sucedieron en simultáneo mientras el programa.