Universidad de los Andes
Facultad de Ingeniería

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

ABET Engineering Accreditation Commission

ANÁLISIS DEL RETO

Lucas Valbuena León, 202325148, l.valbuena1@uniandes.edu.co,

Juan José Cortés Villamil, 202325148, jj.cortesv1@uniandes.edu.co

Requerimiento <<2>>

Descripción

```python
def req_2(data_structs, offer_number, company_name, city):
    """
    Función que soluciona el requerimiento 2
    """
    # TODO: Realizar el requerimiento 2
    lista = lt.newList("ARRAY_LIST")
    for offer in lt.iterator(data_structs):
        if offer["company_name"]==company_name:
            if offer["city"]==city:
                lt.addLast(lista, offer)
    if lt.size(lista)>= (int(offer_number)):
        return lt.subList(lista, 1, int(offer_number))
    else:
        return None
```

En este requerimiento se filtran las ofertas de trabajo por empresa y ciudad y devuelvo el número de ofertas que el usuario quiere ver. Primero se genera un nuevo tipo de arreglo de nombre “lista”, que servirá para llenarlo con las ofertas de trabajo que pasen estos dos filtros. Itero con lt.iterator sobre las ofertas de trabajo para luego pasarlas por los filtros de compañía y ciudad, finalmente si pasan ambos filtros se añaden con la función lt.addLast a un nuevo arreglo donde están las ofertas. A pesar de que la lista contenga todas las ofertas de trabajo que pasaron estos filtros, retornamos con la función lt.subList,

para solo imprimir el número determinado de ofertas por el usuario, si no se encontraron las ofertas suficientes se devuelve None. Es más conveniente tener todas las ofertas agregadas y después sacar una sublista por mantenimiento del código a largo plazo y poder trabajar de más maneras con el arreglo de las ofertas.

Entrada	El catálogo en el apartado de "Jobs", el número de ofertas a imprimir, el nombre de la compañía, la ciudad de la consulta.
Salidas	La sublista con las ofertas filtradas por compañía y ciudad de la oferta.
Implementado (Sí/No)	Sí se implementó y lo hizo Lucas Valbuena Leon

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo:

Pasos	**Complejidad**
Paso 1 (crear el array list)	O(1)
Paso 2 (iterar el arraylist creado)	O(N)
Paso 3 (addLast())	O(1)
Paso 4 (subList())	O(N)
TOTAL	***O(N)***

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).

Procesadores	**AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz**
Memoria RAM	**16,0 GB**
Sistema Operativo	**Windows 11 Pro**

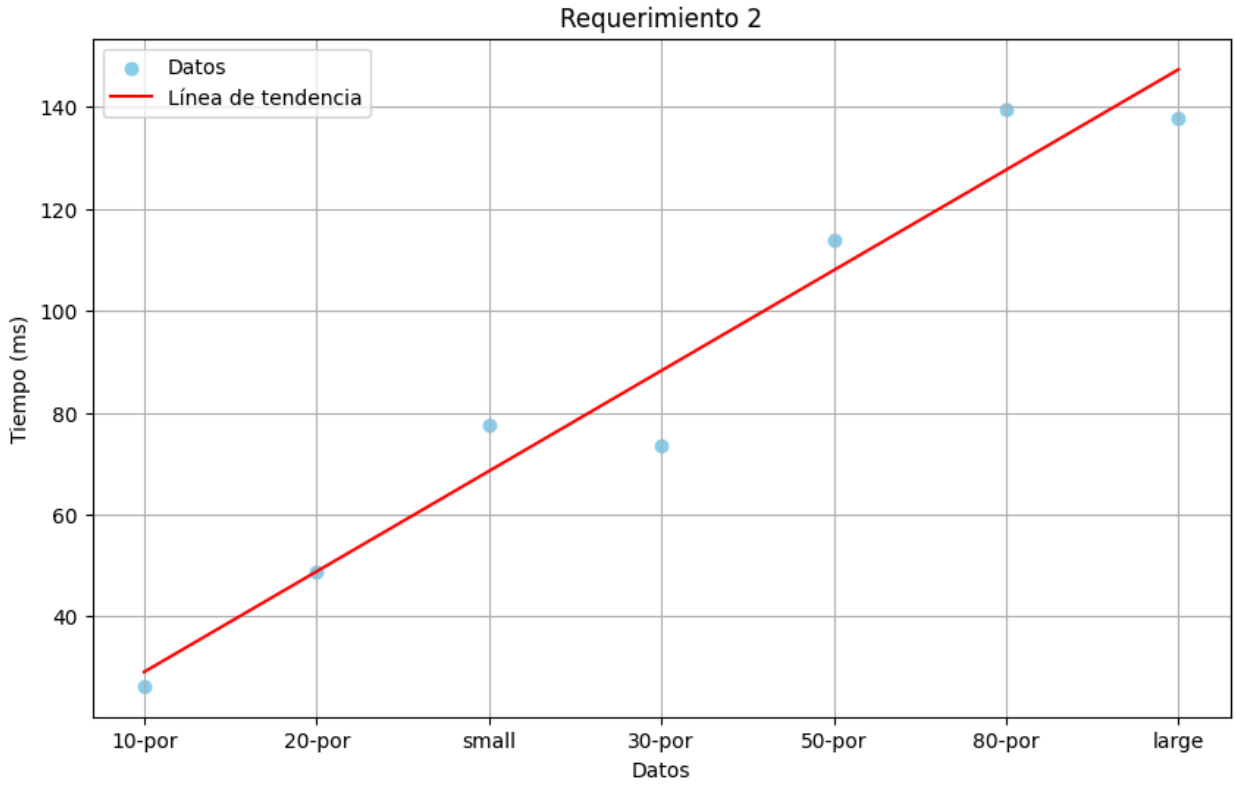
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (ms)**
10-por	26.3366
20-por	48.8534
small	77.633
30-por	73.6195
50-por	113.770
80-por	139.642
large	137.753

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

El análisis de los resultados de implementación denota que a medida que crece el tamaño de los datos crece el tiempo de ejecución linealmente, consecuente con su orden de complejidad O(N). Era predecible el comportamiento lineal de este algoritmo ya que entre más tamaño tenga los datos la iteración va a durar aún más. Tanto en mejor cómo en peor caso va a poseer la misma complejidad temporal ya que va a iterar siempre todo el arreglo y después sacar la sublista.

Requerimiento <<4>>

Descripción

```
def req_4(data_structs, codigo_pais,fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4

    YearIn,MonthIn,DayIn = map(int, fecha_inicial.split("-"))
    YearFin,MonthFin,DayFin = map(int, fecha_final.split("-"))

    lstJobsFit = lt.newList("ARRAY_LIST")

    empresas = {}
    ciudades = {}

    empresa_base = None
    ciudad_base = None

    for job_offer in lt.iterator(data_structs["jobs"]):
        Y,M,D = job_offer["published_at"].split("-")
        Y = int(Y)
        M = int(M)
        D= int(D[:2])


        if (Y >= YearIn and M >= MonthIn and D >= DayIn) and (Y<=YearFin and M <=
MonthFin and D <= DayFin) and (job_offer["country_code"] == codigo_pais):
            lt.addLast(lstJobsFit, job_offer)


            if job_offer["company_name"] not in empresas:

                empresas[job_offer["company_name"]] = 0

            if job_offer["city"] not in ciudades:
                ciudades[job_offer["city"]] = 0


            empresa_base = job_offer["company_name"]
            ciudad_base = job_offer["city"]
            empresas[job_offer["company_name"]] += 1
            ciudades[job_offer["city"]] += 1


    max_empresa = max_min_conteo_ofertas(empresas, empresa_base,True)
```

```
    min_ciudad = max_min_conteo_ofertas(ciudades, ciudad_base,False)
    sorted_lstJobsFit = quk.sort(lstJobsFit,compare_jobs_fit)

    return sorted_lstJobsFit , empresas , ciudades, max_empresa, min_ciudad

#               !FUNCIONES ASOCIADAS AL REQUERIMIENTO 4!
#------------------------------------------------------------------------>
def max_min_conteo_ofertas(data_conteo, base, max):
    if max == True:
        empresa = None
        maximo=0
        for key in data_conteo.keys():
            if data_conteo[key] > maximo:
                maximo= data_conteo[key]
                empresa = key
        return (empresa,maximo)
    else:
        ciudad = None
        min = data_conteo[base]
        for key in data_conteo.keys():
            if data_conteo[key] <= min:
                min= data_conteo[key]
                ciudad = key
        return (ciudad,min)

def compare_jobs_fit(job1, job2):
    Y1,M1,D1 = job1["published_at"].split("-")
    Y1 = int(Y1)
    M1 = int(M1)
    D1= int(D1[:2])
    Y2,M2,D2 = job2["published_at"].split("-")
    Y2 = int(Y2)
    M2= int(M2)
    D2= int(D2[:2])

    if Y1 < Y2:
        return True
    elif Y1 > Y2:
        return False
    else:
        if M1 < M2:
            return True
        elif M1 > M2:
            return False
        else:
            if D1 < D2:
                return True
```

```
            elif D1 > D2:
                return False
            else:

                if job1["company_name"] < job2["company_name"]:
                    return True
                elif job1["company_name"] > job2["company_name"]:
                    return False
                else:
                    return True
#---------------------------------------------------------------------^
```

En este requerimiento se consultan las ofertas hechas en un país en un rango de fechas que entrega el usuario. Primero convierto la fecha que me da el usuario en variables por separado para poder manipularlas mejor. No use DateTime() porque curiosamente cuando lo probé el tiempo de ejecución del algoritmo incrementó considerablemente. Luego itera por cada oferta de trabajo en "Jobs" en "model" y filtra por código de país y por fecha de publicación de la oferta de trabajo. Creé dos diccionarios para poder depositar tanto las ciudades como las empresas y poder llevar registro de las empresas y las ciudades presentes. Como hago un condicional "if not in" dentro del ciclo la complejidad asciende en el peor caso a $O(N^2)$. Luego en una función busco el conteo máximo de ofertas de empresas y el mínimo de ofertas publicadas en una ciudad pues el requerimiento pide mostrar esta información al usuario. Finalmente sorteo la lista con Quicksort cuya función sort_crit() creo también asociada a ese requerimiento 4.

Entrada	El catálogo en el apartado de "Jobs", la fecha inicial, la fecha final y el código del país.
Salidas	Lista de las ofertas sorteadas por la fecha de publicación y si presentan una igual por nombre de compañía; el diccionario de empresas; el diccionario de ciudades, la empresa con el máximo conteo de ofertas y la ciudad con el mínimo conteo de ofertas.
Implementado (Sí/No)	Sí se implementó y lo hizo Juan José Cortés Villamil

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (iterar por las ofertas de trabajo)	O(N)
Paso 2 (condicional para recorrer el diccionario)	O(N)
Paso 3 (addLast())	O(1)
TOTAL	***O(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).

Procesadores	**Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz**
Memoria RAM	8.00 GB (7.88 GB usable)
Sistema Operativo	Windows 10

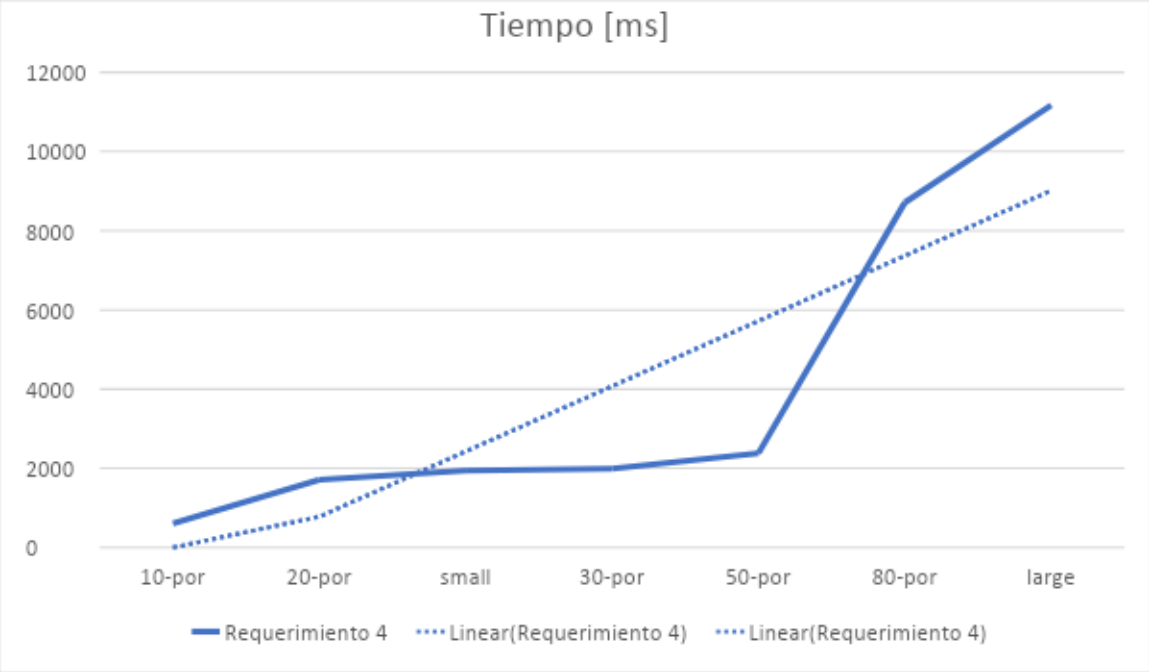
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10-por	607.7445999979973
20-por	1714.6324000060558
small	1945.4221999943256
30-por	1988.9062000215054
50-por	2378.3586000204086
80-por	8710.181499958038
large	11170.85640001297

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

El análisis de los resultados de implementación denota que a medida que crece el tamaño de los datos crece el tiempo de ejecución, pero no dramáticamente. De todos modos, el tiempo de ejecución varía dependiendo de la máquina, pero el orden de crecimiento es $O(N^2)$. Tiene este ordenamiento porque itera por todas las ofertas de trabajo y las filtra por fecha, luego hace un condicional para preguntar si el nombre de la empresa no está en el diccionario. Como tal hacer este condicional no es N pero sería N en el peor caso si se tuviese que hay N empresas cosa que no es el caso en este reto.

Requerimiento <<5>>

Descripción

```
def req_5(data_structs, city, first_date, last_date):
    """
    Función que soluciona el requerimiento 5
    """
    # TODO: Realizar el requerimiento 5


    """
    Pasamos y comparamos los datos en formato datetime.datetime para luego agregarlos en nuestro ARRAY-LIST
    Utilizamos el algoritmo de ordenamiento quick sort para organizar las ofertas cronologicamente y les pasamos la funcion de comparacion como parametro, esta
    organiza cronologicamente y si tienen la misma fecha, organiza alfabeticamente.
    """




    lista = lt.newList("ARRAY_LIST")
    primer_limite = first_date
```

```
    primer_limite0= datetime.strptime(primer_limite, "%Y-%m-%d")


    ultimo_limite = last_date
    ultimo_limite0 = datetime.strptime(ultimo_limite, "%Y-%m-%d")



    for pub in lt.iterator(data_structs):
        if pub["city"]==city:


            fecha = pub["published_at"]
            fecha0 = datetime.strptime(fecha, "%Y-%m-%dT%H:%M:%S.%fZ")




            if primer_limite0 <= fecha0 and fecha0 <= ultimo_limite0:
                lt.addLast(lista, pub)


    lista_or = quk.sort(lista,compare_jobs_fit)


    return lista_or
```

En este requerimiento se consultan las ofertas de una empresa específica en un periodo específico de tiempo en una ciudad específica. Lo primero que se hace es inicializar un nuevo arreglo con el nombre de “lista” donde se almacenarán las ofertas que pasen los filtros, después se pasan todos las fechas a formato datetime.datetime (tanto los que dio el usuario como los que dio el arreglo) porque se encuentran en formato str. Iteramos con lt. iterator sobre el arreglo que pasa por parámetro y aplicamos los filtros de ciudad y de rango de fechas, si pasan ambos filtros se agrega con la función lt.addlast a nuestro arreglo de ofertas. Para organizar las ofertas de trabajo por orden cronológico (y si tiene la misma fecha por nombre de la compañía) se utiliza el algoritmo quicksort, al cual se le diseñó una

función de comparación utilizada en más de un requerimiento, a función consiste en comparar las fechas y como último recurso comparar los nombres de las compañías, esta función devuelve valores booleanos. Una vez aplicado el quicksort, con nuestro arreglo y la función de comparación, devolvemos el arreglo organizado (en otra variable, no en la misma).

Entrada	El catálogo en el apartado de "Jobs", la ciudad de la consulta, la fecha inicial a consultar y la fecha final a consultar.
Salidas	La lista organizada cronológicamente con las ofertas de trabajo por empresa entre dos fechas determinadas .
Implementado (Sí/No)	Sí se implementó y lo hizo Lucas Valbuena Leon.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (crear el array list)	O(1)
Paso 2 (paso formato a date.datetime)	O(1)
Paso 3 (iterar el arraylist creado)	O(N)
Paso 4 (addLast())	O(1)
Paso 5 (ejecutar quicksort())	$O(N^2)$
TOTAL	***$O(N^2)$***

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).

Procesadores	**AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz**
Memoria RAM	**16,0 GB**
Sistema Operativo	**Windows 11 Pro**
Librerías	**Datetime**

Tablas de datos

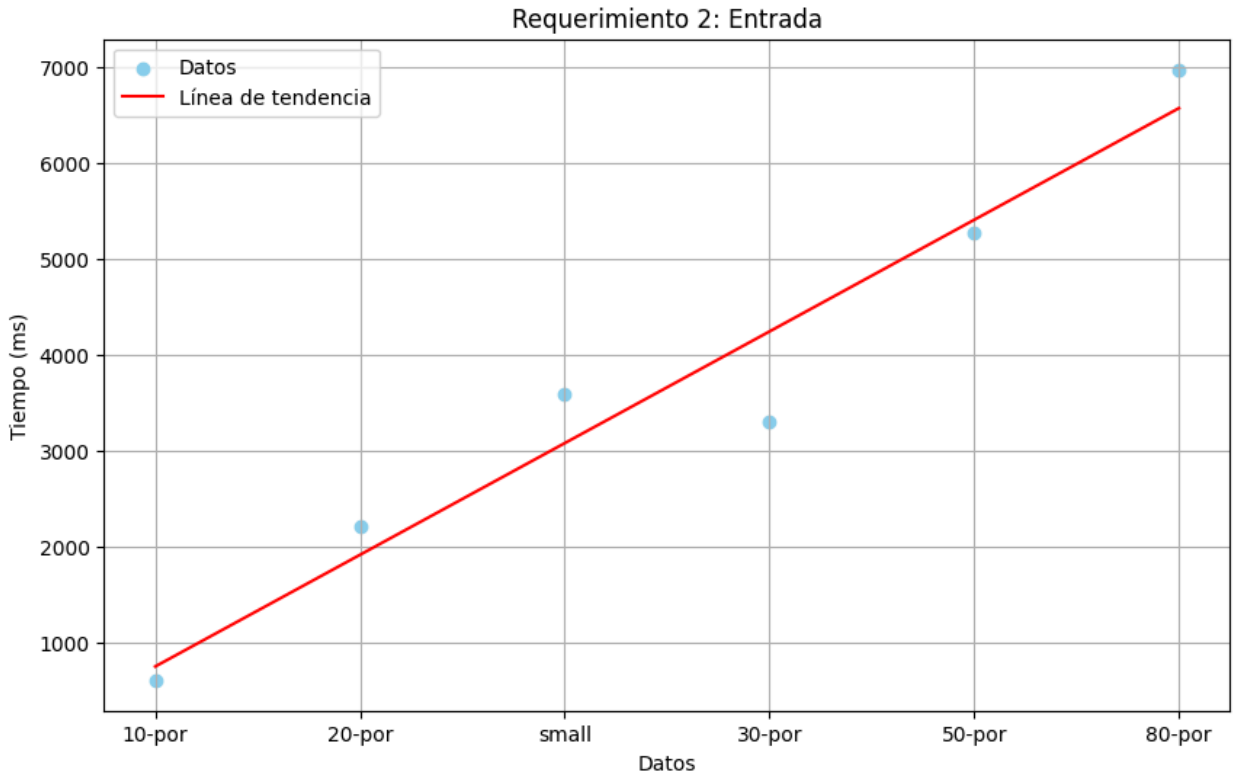
Las tablas con la recopilación de datos de las pruebas.

Entrada	**Tiempo (ms)**
10-por	613.704
20-por	2212.48
small	3599.822

30-por	3307.219
50-por	5284.839
80-por	6979.594
large	El tiempo de espera fue demasiado

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Según los resultados del análisis el comportamiento del algoritmo es lineal pero teóricamente debería ser $O(N^2)$. Utilizando un rango de valores más amplio en la tabla podríamos ser capaces de observar una tendencia cuadrática en la línea de tendencia. Puede que no esté ejecutando el peor caso de la complejidad sino un caso promedio que podría ser O(N).

Requerimiento <<6>>

Descripción

```
def req_6(data_structs, N, codigo_pais, lvl_experticia ,fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 6
    """
    # TODO: Realizar el requerimiento 6
    job_offers = lt.newList("ARRAY_LIST")

    YearIn,MonthIn,DayIn = map(int, fecha_inicial.split("-"))
    YearFin,MonthFin,DayFin = map(int, fecha_final.split("-"))

    newCatalog = {"model":{}}

            #           ¡SALARIO POR TIPO DE EMPLEO!

    #A partir de aquí hago un diccionario que tenga como llaves el id de un trabajo y como valores debe tener de cuanto a cuanto va el salario

    salario_x_empleo = {}
    for employment_type in lt.iterator(data_structs["employments_types"]):
        if(employment_type["salary_from"]!="") and (employment_type["salary_to"]!=""):
            salario_medio = (float(employment_type["salary_from"]) + float(employment_type["salary_to"]))/2
            salario_x_empleo[employment_type["id"]]=salario_medio
```

```
        else:
            salario_x_empleo[employment_type["id"]]=0


    #NO DAN CÓDIGO DE PAÍS
    #A partir de aquí se escribe el código en relación a meter todas las ofertas de una ciudad en una lista separada y luego meter todas esas listas en otro ARRAY_LIST
    if codigo_pais == None:

        #           ¡SEPARAR OFERTAS POR CIUDAD!


        #A partir de aquí hago todo relacionado con jobs para separar ofertas por ciudad
        for job_offer in lt.iterator(data_structs["jobs"]):

            Y,M,D = job_offer["published_at"].split("-")
            Y = int(Y)
            M = int(M)
            D= int(D[:2])

            if  (job_offer["experience_level"] == lvl_experticia):
                if (Y >= YearIn and M >= MonthIn and D >= DayIn) and (Y<=YearFin and M <= MonthFin and D <= DayFin):
                    job_offer["offer_salary"] = salario_x_empleo[job_offer["id"]]
```

```
                    if job_offer["city"] not in newCatalog["model"].keys():

newCatalog["model"][job_offer["city"]]=lt.newList("ARRAY_LIST")
                        newCatalog["model"][job_offer["city"]]["belongs_to"] =
job_offer["city"]


                        newCatalog["model"][job_offer["city"]]["sum_salary"] = 0
                        newCatalog["model"][job_offer["city"]]["avg_salary"] = 0

newCatalog["model"][job_offer["city"]]["cant_ofertas_con_salario"] = 0



newCatalog["model"][job_offer["city"]]["empresas_presentes"] = []


                    if job_offer["company_name"] not in
newCatalog["model"][job_offer["city"]]["empresas_presentes"]:

newCatalog["model"][job_offer["city"]]["empresas_presentes"].append(job_offer["co
mpany_name"])
                    if    salario_x_empleo[job_offer["id"]] != 0:

newCatalog["model"][job_offer["city"]]["cant_ofertas_con_salario"] +=1
                        newCatalog["model"][job_offer["city"]]["sum_salary"] +=
salario_x_empleo[job_offer["id"]]


                    lt.addLast(newCatalog["model"][job_offer["city"]], job_offer)


    else:
        #SÍ DAN CÓDIGO DE PAÍSES
```

```
        #           ¡SEPARAR OFERTAS POR CIUDAD!


        #A partir de aquí hago todo relacionado con jobs para separar ofertas por
ciudad
        for job_offer in lt.iterator(data_structs["jobs"]):


            Y,M,D = job_offer["published_at"].split("-")
            Y = int(Y)
            M = int(M)
            D= int(D[:2])


            if  (job_offer["experience_level"] == lvl_experticia) and
(job_offer["country_code"] == codigo_pais):
                if (Y >= YearIn and M >= MonthIn and D >= DayIn) and (Y<=YearFin
and M <= MonthFin and D <= DayFin):
                    job_offer["offer_salary"] = salario_x_empleo[job_offer["id"]]




                    if job_offer["city"] not in newCatalog["model"].keys():

newCatalog["model"][job_offer["city"]]=lt.newList("ARRAY_LIST")
                        newCatalog["model"][job_offer["city"]]["belongs_to"] =
job_offer["city"]
```

```
                        newCatalog["model"][job_offer["city"]]["sum_salary"] = 0
                        newCatalog["model"][job_offer["city"]]["avg_salary"] = 0
newCatalog["model"][job_offer["city"]]["cant_ofertas_con_salario"] = 0


newCatalog["model"][job_offer["city"]]["empresas_presentes"] = []




                    if job_offer["company_name"] not in
newCatalog["model"][job_offer["city"]]["empresas_presentes"]:
newCatalog["model"][job_offer["city"]]["empresas_presentes"].append(job_offer["co
mpany_name"])
                    if    salario_x_empleo[job_offer["id"]] != 0:
newCatalog["model"][job_offer["city"]]["cant_ofertas_con_salario"] +=1
                        newCatalog["model"][job_offer["city"]]["sum_salary"] +=
salario_x_empleo[job_offer["id"]]


                    lt.addLast(newCatalog["model"][job_offer["city"]], job_offer)



        #           ¡PROCESAR LAS CIUDADES Y TRANSFORMALAS EN ARRAY_LIST PARA
PODER USAR EL SORT!


    data = convertir_dict_TAD(newCatalog)
```

```
    encontrar_promedio_ofertas_ciudad(data)


        #sorted_list = quk.sort(data,compare_conteo_ciudades)
    sorted_list = sa.sort(data,compare_ciudades)
    for city in lt.iterator(sorted_list):
        sa.sort(city,compare_ofertas_en_ciudades)


        """
        for city in lt.iterator(sorted_list):
           print(city["size"])
           print(city["belongs_to"])
           print(city["avg_salary"])
        """


    if lt.size(sorted_list) < N:
        return lt.subList(sorted_list,1,lt.size(sorted_list))


    else:
        return lt.subList(sorted_list,1,N)


    #¡FUNCIONES ASOCIADAS AL REQUERIMIENTO 6!
    #.....................................................................>


def encontrar_promedio_ofertas_ciudad(data):
```

```
    for city in lt.iterator(data):
        #city["avg_salary"] = city["sum_salary"]/lt.size(city)
        if city["cant_ofertas_con_salario"] != 0:
            city["avg_salary"] =
city["sum_salary"]/city["cant_ofertas_con_salario"]
        else:
            city["avg_salary"] = 0



def convertir_dict_TAD(newCatalog):
    data = lt.newList("ARRAY_LIST")
    for key in newCatalog["model"].keys():
        lt.addLast(data, newCatalog["model"][key])

    return data



def compare_ciudades(ciudad1,ciudad2):
    if int(lt.size(ciudad1)) < int(lt.size(ciudad2)):
        return False
    elif int(lt.size(ciudad1)) > int(lt.size(ciudad2)):
        return True
    else:
        if int(ciudad1["avg_salary"]) < int(ciudad2["avg_salary"]):
            return False
        elif int(ciudad1["avg_salary"]) > int(ciudad2["avg_salary"]):
```

```
            return True
        else:
            return False



def compare_ofertas_en_ciudades(oferta1, oferta2):
    if int(oferta1["offer_salary"]) < int(oferta2["offer_salary"]):
        return False
    elif int(oferta1["offer_salary"]) > int(oferta2["offer_salary"]):
        return True



def manipular_NJobOffers_req6(NJobOffers):
    total_ciudades= lt.size(NJobOffers)
    total_empresas= []
    total_ofertas=0
    avg_salary_total_offers = 0

    for city in lt.iterator(NJobOffers):
        total_ofertas += city["cant_ofertas_con_salario"]

        total_empresas += city["empresas_presentes"]




    for city in lt.iterator(NJobOffers):
```

```
        avg_salary_total_offers += city["avg_salary"]/total_ciudades


    nombre_ciudad_mayor = NJobOffers["elements"][0]["belongs_to"]
    conteo_ciudad_mayor = lt.size(NJobOffers["elements"][0])


    nombre_ciudad_menor = NJobOffers["elements"][total_ciudades-1]["belongs_to"]
    conteo_ciudad_menor = lt.size(NJobOffers["elements"][total_ciudades-1])


    for city in lt.iterator(NJobOffers):
        city["oferta_mayor"] =
(city["elements"][0]["company_name"],city["elements"][0]["offer_salary"])
        city["oferta_menor"] =
(city["elements"][lt.size(city)-1]["company_name"],city["elements"][lt.size(city)
-1]["offer_salary"])


    return total_ciudades,len(set(total_empresas)),total_ofertas,
avg_salary_total_offers,nombre_ciudad_mayor, conteo_ciudad_mayor,
nombre_ciudad_menor, conteo_ciudad_menor



#^.....................................................................^
```

En este requerimiento clasificó N ciudades con mayor número de ofertas de trabajo por experiencia entre un rango de experiencias. A primera vista pareciera que tengo muchos ciclos anidados pero no es el caso, la mayor complejidad es O(N^2). En un principio cree un diccionario llamado “salario_x_empleo” para guardar el salario que ganaba cada tipo de trabajo en la llave del trabajo. Luego itero por las ofertas de trabajo y filtro con base en el nivel de experiencia y las fechas entregadas por el usuario. Por eso este requerimiento se demora menos que el 4, porque revisa menos ofertas al tener un filtro más restringido. Previo a ese ciclo cree un nuevo catálogo con la siguiente estructura:

```
newCatalog = {"model":{}}
```

De modo que en este catálogo voy a ir guardando las ofertas de cada ciudad en una llave con el nombre de la ciudad. De todos modos hay que devolver la lista ordenada y no puedo usar los algoritmos de ordenamiento con estructuras de datos que no sean de DISClib entonces para eso cree una función que vuelve ese nuevo catálogo un TAD de listas, y cada lista tiene un nuevo atributo que dice el nombre de la ciudad a la que pertenece esa lista. Finalmente sorteo las ofertas y las ciudades. Cabe aclarar que esta función ya deja todo hecho del requerimiento seis pero como hay que manipular información para imprimirla cree otro requerimiento para que se encargue de entregarle al controller todo lo que hay que imprimir. Como en este requerimiento el usuario puede no dar código de país repito el código pero sin el filtro del código de país.

Entrada	data_structs, N, codigo_pais, lvl_experticia ,fecha_inicial, fecha_final
Salidas	Retorna una sublista con N elementos, si la consulta no tiene suficientes N elementos devuelve los que tenga.
Implementado (Sí/No)	Sí se implementó y lo implementó Juan José Cortés

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (Iteración por “employments types” para la creación del diccionario)	O(N)
Paso 2 (Iteración por ofertas de trabajo)	O(N)
Paso 3 (Condicional para verificar si la ciudad está en el diccionario)	O(N)
Paso 4 (lt.addLast())	O(1)
TOTAL	***O(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).

Procesadores	**Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz**
Memoria RAM	8.00 GB (7.88 GB usable)
Sistema Operativo	Windows 10

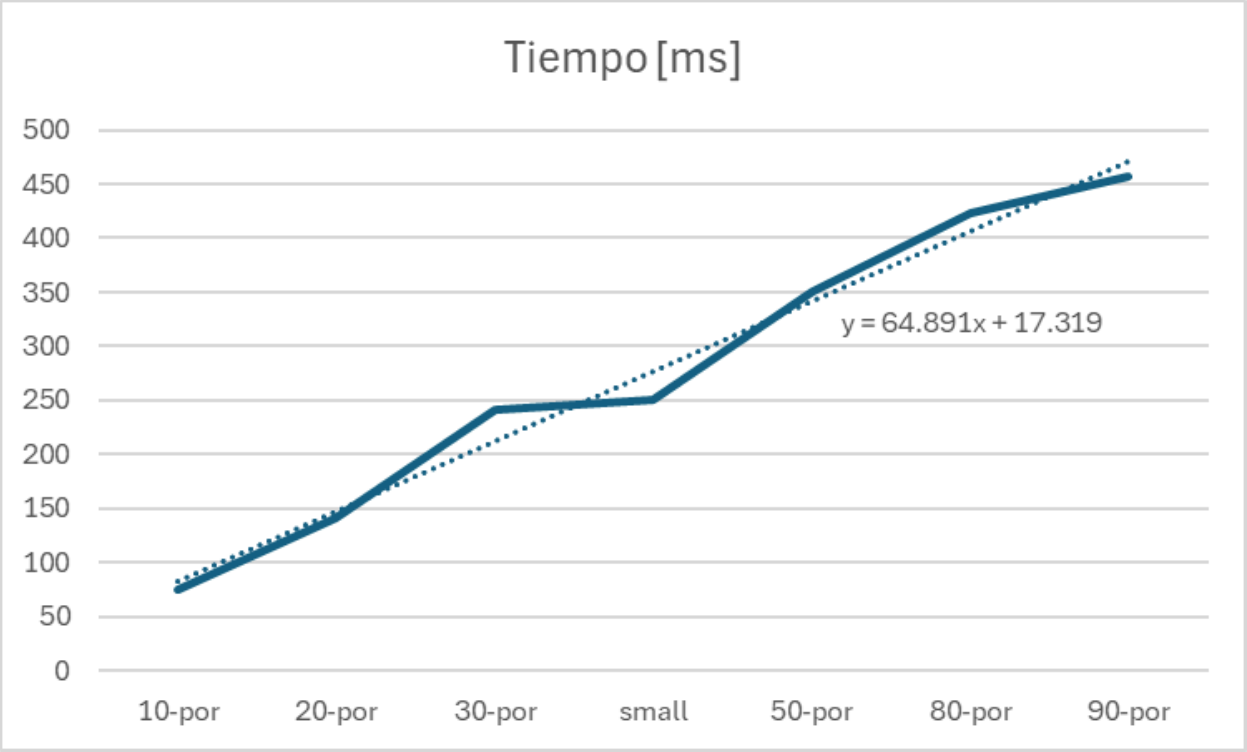
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10 pct	74.8246000111103
20 pct	141.36340001225471
small	250.55530002713203
30 pct	241.6928000152111
50 pct	349.94020000100136
80 pct	423.53079998493195
90 pct	456.2828000187874
large	Hay un error con el csv de large

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Según los resultados del análisis el comportamiento del algoritmo es lineal pero teóricamente debería ser $O(N^2)$. Esto probablemente se debe a que no hay N empresas y el for anidado no llega a ser tan significativo para que la función se vuelva cuadrática.

Requerimiento <<7>>

```
def req_7(data_structs, country_number, first_date, last_date ):
    """
    Función que soluciona el requerimiento 7
    """
    # TODO: Realizar el requerimiento 7
    listax = lt.newList("ARRAY_LIST")
    dicc_country = {}
    conteo_countries= {}
    dicc_final = {}
    dicc_experience = {}
    lista_final = lt.newList("ARRAY_LIST")
    first = datetime.strptime(first_date, "%Y-%m-%d")
    last = datetime.strptime(last_date, "%Y-%m-%d")


    """
    Tomamos el ARRAY-LIST original y guardamos las ofertas de trabajo por pais en un diccionario
    y otro diccionario que contenga el pais y el numero de ofertas de trabajo.

    Al diccionario de pais X numero de ofertas de trabajo lo pasamos a otro diccionario con solo los N paises que
    nos pidio el usuario.

    Unimos todo en un ARRAY-LIST el cual tendra como condicion que el pais se encuentre en el diccionario de N paises, despues pasara
    toda la informacion de ofertas X pais a ese ARRAY-LIST
```

```
    Despues de obtener la ultima version del ARRAY-LIST buscamos en ella los tres
niveles de experticia solicitados y los vamos organizando en otro diccionario,
    donde la llave sera el nivel de experticia y el valor sera otro diccionario
con el nivel de experiencia-numero de veces que se repitio.
    """



    for cn in lt.iterator(data_structs["jobs"]):
        fecha = cn["published_at"]
        fecha0= datetime.strptime(fecha, "%Y-%m-%dT%H:%M:%S.%fZ")



        if first <= fecha0 and fecha0 <= last:
            lt.addLast(listax, cn)







            if cn["country_code"]not in dicc_country:
                lista = []
                lista.append(cn)
                dicc_country[cn["country_code"]] = lista


            else:
                dicc_country[cn["country_code"]].append(cn)
```

```
        if cn["country_code"]not in conteo_countries:
            conteo_countries[cn["country_code"]]=1
        else:

            conteo_countries[cn["country_code"]]+=1









if len(conteo_countries)>0:
    for i in range(int(country_number)):
        if len(conteo_countries)>0:
            maximo = max(conteo_countries.values())
            for i in conteo_countries.items():
                if i[1]==maximo:
                    nombre_maximo = i[0]
            dicc_final[nombre_maximo]=maximo
            conteo_countries.pop(nombre_maximo)


for i in dicc_country.items():
    if i[0] in dicc_final:
        for j in i[1]:
            lt.addLast(lista_final, j)
```

```
for i in lt.iterator(lista_final):

    if "junior" == i["experience_level"]:



        if i["experience_level"] not in dicc_experience:
            dicc_sec = {}
            dicc_experience["junior"] = dicc_sec
        else:
            dicc_sec = dicc_experience["junior"]
            #dicc_experience[i["experience_level"]] = dicc_sec

        for j in lt.iterator(data_structs["skills"]):
            if i["id"] == j["id"]:
                if j["name"] not in dicc_sec:
                    dicc_sec[j["name"]] = 1
                else:
                    dicc_sec[j["name"]] += 1
        dicc_experience["junior"] = dicc_sec
        #print(dicc_experience)
```

```
        if "mid" == i["experience_level"]:



            if i["experience_level"] not in dicc_experience:
                dicc_sec1 = {}
                dicc_experience["mid"] = dicc_sec1
            else:
                dicc_sec1 = dicc_experience["mid"]
                #dicc_experience[i["experience_level"]] = dicc_sec1



            for j in lt.iterator(data_structs["skills"]):
                if i["id"] == j["id"]:


                    if j["name"] not in dicc_sec1:
                        dicc_sec1[j["name"]] = 1
                    else:
                        dicc_sec1[j["name"]] += 1
            dicc_experience["mid"] = dicc_sec1
            #print(dicc_experience)


        if "senior" == i["experience_level"]:



            if i["experience_level"] not in dicc_experience:
                dicc_sec2 = {}
```

```
                dicc_experience["senior"] = dicc_sec2
            else:
                dicc_sec2 = dicc_experience["senior"]
                #dicc_experience[i["experience_level"]] = dicc_sec2



            for j in lt.iterator(data_structs["skills"]):
                if i["id"] == j["id"]:
                    if j["name"] not in dicc_sec2:
                        dicc_sec2[j["name"]] = 1
                    else:
                        dicc_sec2[j["name"]] += 1
            dicc_experience["senior"] = dicc_sec2
            #print(dicc_experience)


    #print(dicc_experience)







    return lista_final, dicc_final, dicc_experience
```

Descripción

En este requerimiento se clasifica los países con mayores ofertas de trabajo entre dos fechas específicas y se imprime el número de ofertas que el usuario quiera. Primero inicializamos todo lo que vamos a utilizar, después, tomamos el arreglo original y lo iteramos con lt.iterator para guardar las ofertas de trabajo por país en un diccionario. Al diccionario de país X número de ofertas de trabajo lo pasamos a otro diccionario con solo los N países que pide imprimir el usuario. Unimos toda la información en un

arreglo con un filtro de entrada, el país se debe encontrar en el diccionario de N países y si es así agrega todas las ofertas de trabajo que tenga al arreglo, como llave se encuentra el nombre del país y como valor todas las ofertas de trabajo. Después de sacar el arreglo final buscamos los tres niveles de experticia solicitados y los vamos organizando en otro diccionario, su llave será el nivel de experticia y el valor será un diccionario compuesto por las habilidades y el número de veces que se repiten.

Entrada	el catálogo general (sin apartado), el número de países que debe imprimir la consulta, la fecha inicial a consultar y la fecha final a consultar.
Salidas	El arraylist final con las ofertas en N países, el diccionario con los N países y el número de ofertas por país, el diccionario por nivel de experticia, donde se encuentran las habilidades y el número de habilidades por nivel de experticia.
Implementado (Sí/No)	Si se implementa y lo hizo Lucas Valbuena Leon

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1 (crear el primer arraylist)	O(1)
Paso 2 (crear el array list final (también + estructuras))	O(1)
Paso 3 (iterar catalog[“jobs”])	O(N)
Paso 4 (pasar a formato datetime.datetime)	O(1)
Paso 5 (función addLast ())	O(1)
Paso 6 (funcion .append() para dicc_country)	O(1)
Paso 7 (recorrido por N ofertas que pide el usuario)	O(M) (siendo M el número de ofertas que pida el usuario)
Paso 8 (iterar “Lista_final”)	O(N)
Paso 9 (iterar “Skills” dentro de lista final)	O(N) (multiplicado el otro O(N) de arriba)
TOTAL	***O(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).

Procesadores	**Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz**
Memoria RAM	8.00 GB (7.88 GB usable)
Sistema Operativo	Windows 10
Librerías	**Datetime**

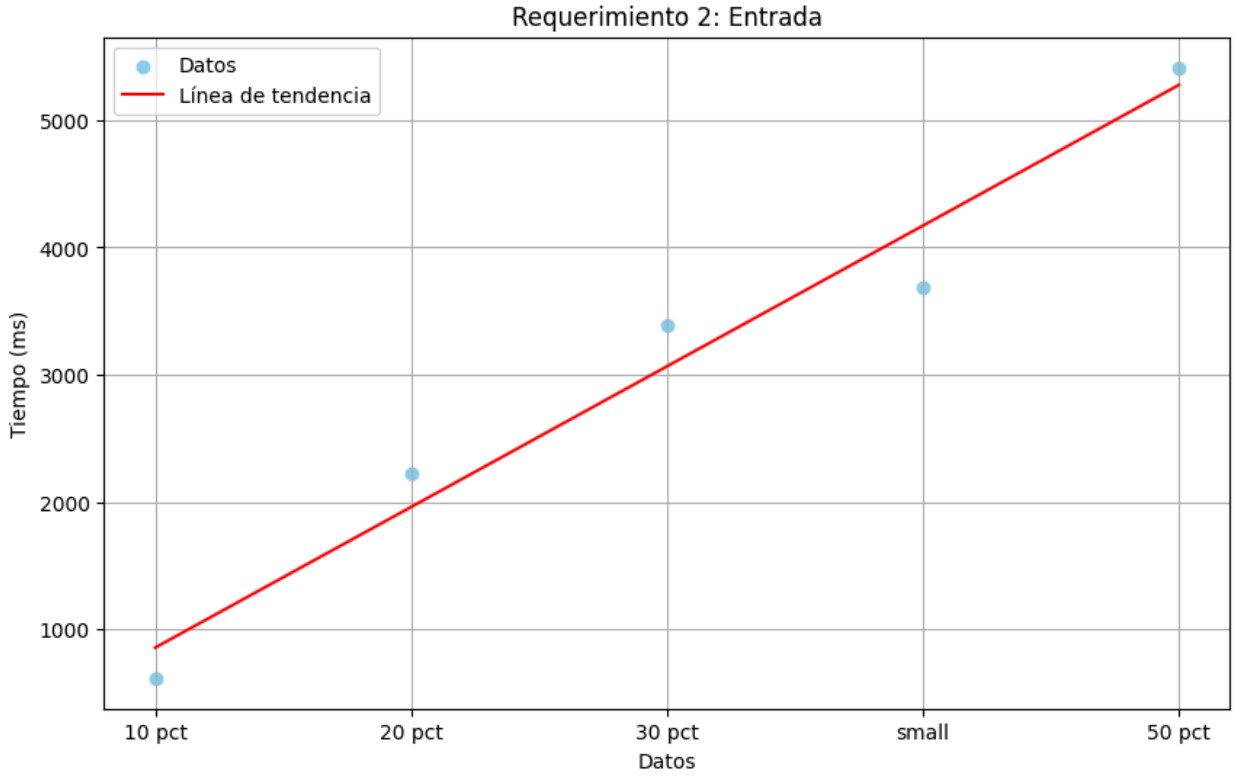
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10 pct	618.9048
20 pct	2226.0519
30 pct	3391.243
small	3684.64
50 pct	5411.193
80 pct	El tiempo de espera fue demasiado
90 pct	El tiempo de espera fue demasiado
large	El tiempo de espera fue demasiado

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

El comportamiento del algoritmo en términos de complejidad es bastante similar a una tendencia lineal O(N), a pesar de que su complejidad sea $O(N^2)$. Esto se podría deber a que el número de iteraciones no es tan grande como para adquirir su complejidad total del peor caso, que en este caso es una cuadrática. Se debe encontrar en un caos promedio entre el mejor caso y el peor caso para adquirir este tipo de comportamiento.

Requerimiento <<8>>

Descripción

Para este requerimiento me base mucho en el código del requerimiento 6, de hecho estuve tentado a llamar la función del requerimiento 6 para ahorrarme código pero necesitaba que se guardaran más cosas en las listas de ciudades por eso son muy parecidos pero son diferentes. Se vuelve a hacer lo mismo que en el requerimiento 6. Se crea un catálogo nuevo en el que se insertarán listas de las ofertas de una ciudad, pero ahora cada ciudad también va a tener un una referencia que dice a qué país pertenece. Cabe aclarar que en un principio cuando se creó el diccionario de "salario_x_empleo" convierto el salario promedio a USD usando la librería de "currency_converter". Al probar el requerimiento 8 puede resultar un error porque es necesario descargar la librería con el siguiente comando pip install "currency_converter". Asimismo se hace un diccionario que es "habilidades_x_empleo" que como llaves tiene ids de empleos y como valores el nivel de habilidad que se pide para el empleo para luego poder calcular el promedio de nivel habilidades de un país. Una vez está hecha la lista de ciudades, se convierte en un TAD y se ordena de la misma forma que en el requerimiento 6. Esta función retorna la lista ordenada y una lista python con las divisas presentes en todas las ciudades que se presenciaron en la consulta.

```
data = convertir_dict_TAD(newCatalog)

encontrar_promedio_ofertas_ciudad(data)



sorted_list = sa.sort(data,compare_ciudades)

for city in lt.iterator(sorted_list):

    sa.sort(city,compare_ofertas_en_ciudades)
```

Ahora, hasta este momento se tiene una lista de ciudades pero se necesita un listado de países, entonces usando otra función en la que se repite el mismo procedimiento que con las ciudades en un principio, es decir se crea un nuevo catálogo que tiene como llaves los países y de este modo se puede ir depositando las ciudades en cada país correspondiente con base a la referencia de país que se creó en la lista de cada ciudad en la primera parte de la función. Una vez el catálogo está lleno, se transforma en un TAD de

DISClib usando la función que cree en el requerimiento 6. Finalmente se ordena la lista de países con base en el promedio de oferta salarial.

Entrada	data_structs,fecha_inicial, fecha_final, lvl_experticia
Salidas	sorted_list (lista de ciudades sorteadas) , divisas (lista de divisas presentes en la consulta) Luego la función `manipular_data_req8(data)` recibe esa lista de ciudades sorteadas y comienza a crear una lista de países para devolver lo que tiene que mostrarse al usuario.
Implementado (Sí/No)	Sí se implemento y lo implementó Juan José Cortés Villamil

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1 (Iteración por los employments types para asignarles llaves al diccionario “salario_x_empleo”)	O(N)
Paso 2 (Iteración por los skills para asignarles llaves al diccionario “habilidades_x_empleo”)	O(N)
Paso 3 (Iteración por las ofertas de trabajo)	O(N)
Paso 4 (condicional que verifica si el nombre de la ciudad está en las llaves del nuevo catálogo)	O(N)
Paso 5 .append() (añadir las divisas presentes en la consulta a una lista python)	O(1)
Paso 6 lt.append() (añadir al final de la lista por ciudad las ofertas)	O(1)
Paso 7 Sorteo ciudades usando shellsort	O(NlogN)
Paso 8 Sortear ofertas en cada ciudad (itera por las ciudades y sortea cada ciudad de modo que se multiplican las complejidades).	O(N^2logN)
Paso 9: Llamar `manipular_data_req8(data)`	O(1)
Paso 10 (iterar por las ciudades)	O(N)
Paso 11 (condicional que verifica si el nombre de la ciudad ya está en el nuevo catálogo que se está creando)	O(N)
Paso 12 `convertir_dict_TAD(paises)`	O(N)
Paso 13 Sacar el promedio de nivel de habilidad	O(N)
Paso 14 Sortear la lista de países por el promedio salarial de cada una.	Depende del algoritmo de ordenamiento que escoja el usuario; por default es quicksort ósea O(N^2)
Paso 15 `resultados_pais(paises, ultimo):`	O(N)

(Segunda parte del requerimiento 8, concentrarse en el país con menor y con mayor oferta salarial)	
TOTAL	***O(N ^{2} logN)***

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).

Procesadores	**Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90 GHz**
Memoria RAM	8.00 GB (7.88 GB usable)
Sistema Operativo	Windows 10
Librerías	currency converter

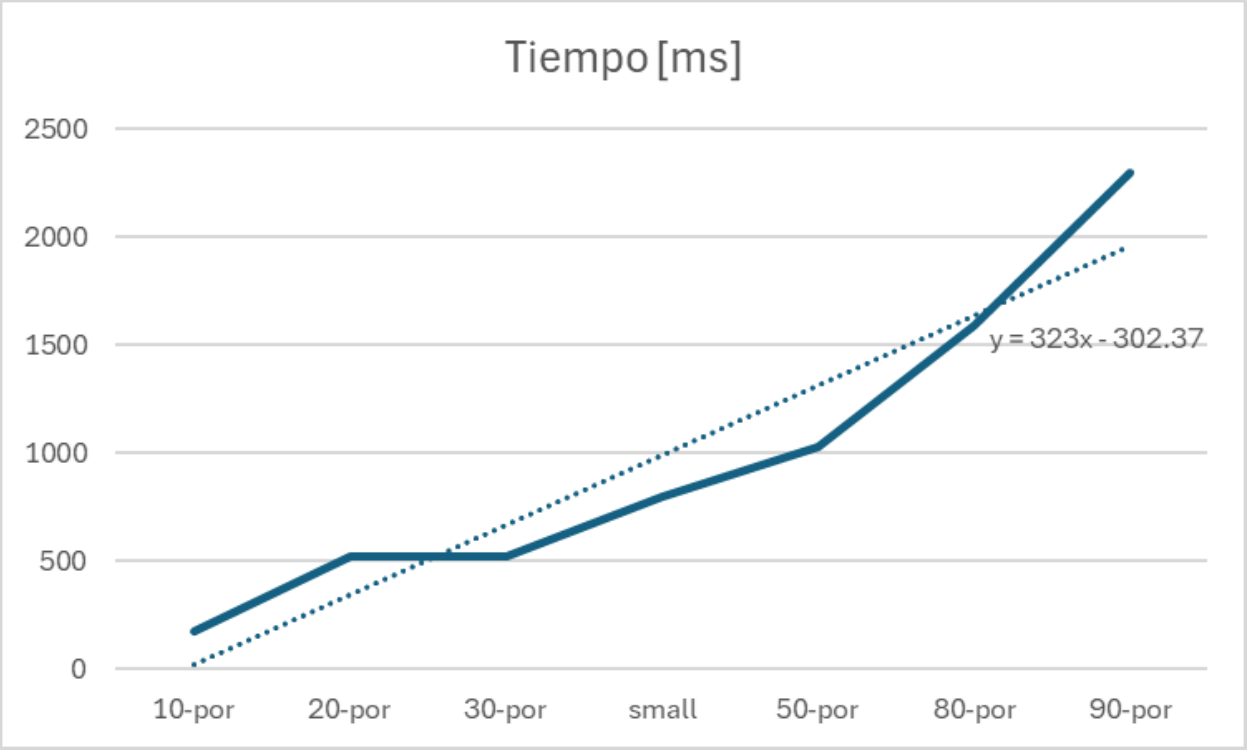
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Entrada	Tiempo (ms)
10 pct	171.47450000047684
20 pct	517.3747000098228
small	799.7129999995232
30 pct	519.4229999780655
50 pct	1027.2080000042915
80 pct	1591.2647999823093
90 pct	2300.96099999547
large	Hay un error con el csv “employments types” de large

Gráficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad de este requerimiento en el peor de los casos es $O(N^2 logN)$ sin embargo el gráfico se ve un comportamiento más bien cuadrático esto se debe a que cuando se sortea dentro de un for las ciudades, la cantidad de ofertas que cada ciudad llega a tener no es igual a N, pero si ese fuese el caso probablemente $O(N^2 logN)$ llegaría a dominar sobre él $O(N^2)$. En este caso la gráfica tiene un comportamiento cuadrático evidente, si fuese $O(N^2 logN)$ crecería mucho más rápido. Se puede evitar esta complejidad para el peor caso pero se necesitan más ciclos y probablemente más memoria.