Sustentación Reto 1

EDA (202401)

Ana Sofia Trillos Cod 202222702

Sara García Cod 202320378

Daniela González Cod 202320856

Requerimiento 1

N ofertas por experiencia en un país especifico

```python
def ultimosNPaisExp(data_structs, codPais, exp, n):
    """

    Args:
    El número (N) de ofertas a listar (ej.: 3, 5, 10 o 20).
    Código del país (ej.: PL, CO, ES, etc).
    Nivel de experticia de las ofertas a consultar (junior, mid, o senior).
    Returns:
    El total de ofertas de trabajo ofrecidas según la condición (junior, mid, o senior).
    Para cada una de las ofertas de la consulta debe presentar la siguiente información:


    """

    ofertas_tot= lt.newList("ARRAY_LIST")
    a= lt.newList("ARRAY_LIST")

    for oferta in lt.iterator(data_structs["jobs"]):
        if oferta["experience_level"]== exp and oferta["country_code"]== codPais:
            lt.addLast(ofertas_tot, oferta)

    size = lt.size(ofertas_tot)

    i = size
    while i > 0 and lt.size(a)< n:
        lt.addLast(a, lt.getElement(ofertas_tot, i))
        i-= 1
    return a, size
```

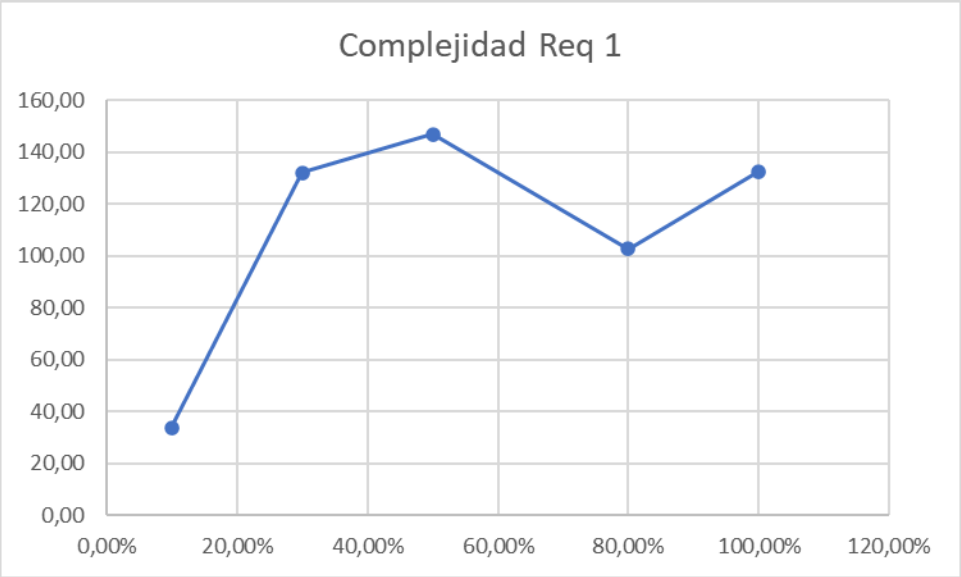
Este requerimiento, a través de los parámetros: estructura de datos, código de país, experiencia y n, busca que el usuario conozca las ofertas existentes según el nivel de experiencia en un país específico. Por lo tanto, lo primero que se debe hacer es, mediante un bucle for, iterar sobre la lista de trabajos. Para asi, confirmar que en cada oferta estén presentes el nivel de experiencia y el código del país, agregando al final aquellos que cumplan con esta condición. Luego, usando el tamaño de las ofertas, devolvemos el número total de ofertas. Terminando, con un bucle while, organizamos las n ofertas que el usuario desee por orden cronológico.

Operación	Complejidad
For	O(N)
While	O(N)

Complejidad Final: O(N)

Prueba experimental:

Porcentaje de la muestra [pct]	Tiempo [ms]
10.00%	33.970
30.00%	132.029
50.00%	146.879
80.00%	102.810
100.00%	132.452



Requerimiento 2:

N ciudades más recientes de una ciudad y empresa específicas

```python
def req_2(data_structs, n, city, nom_emp):

    # TODO: Realizar el requerimiento 2
    jobs = data_structs["jobs"]
    ofertas_totales  = lt.newList("ARRAY_LIST")
    ofertas  = lt.newList("ARRAY_LIST")

    for job in lt.iterator(jobs):
        if job["city"] == city and job["company_name"] == nom_emp:
            lt.addLast(ofertas_totales, job)

    size = lt.size(ofertas_totales)

    i = size
    while i > 0 and lt.size(ofertas) < n:
        lt.addLast(ofertas, lt.getElement(ofertas_totales, i))
        i -= 1


    return ofertas, size
```

Imagen #. Código del requerimiento 2 en el *model*

Este requerimiento busca determinar cuáles son las N ofertas más recientes que cumplen con la condición de estar en una ciudad y empresa específica. Para cumplir esto, la función recibe, desde el *controller*, el data_structs, el número N, la ciudad y el nombre de la empresa. A partir de esto, del data_structs se toma solo "jobs"; se crea una lista llamada "ofertas_totales", la cual, mediante un *for* guarda todas las ofertas de la ciudad y empresa específicas (no solo las N más recientes). Luego, mediante el *while,* se toma de "ofertas_totales" únicamente las N más recientes. Puesto que el archivo se ordena previamente en orden cronológico (de oferta más antigua a más reciente), para hallar las N más recientes, es necesario recorrer la lista "ofertas_totales" de manera inversa y añadir los elementos a una nueva lista "ofertas" hasta que la longitud de dicha lista sea N.

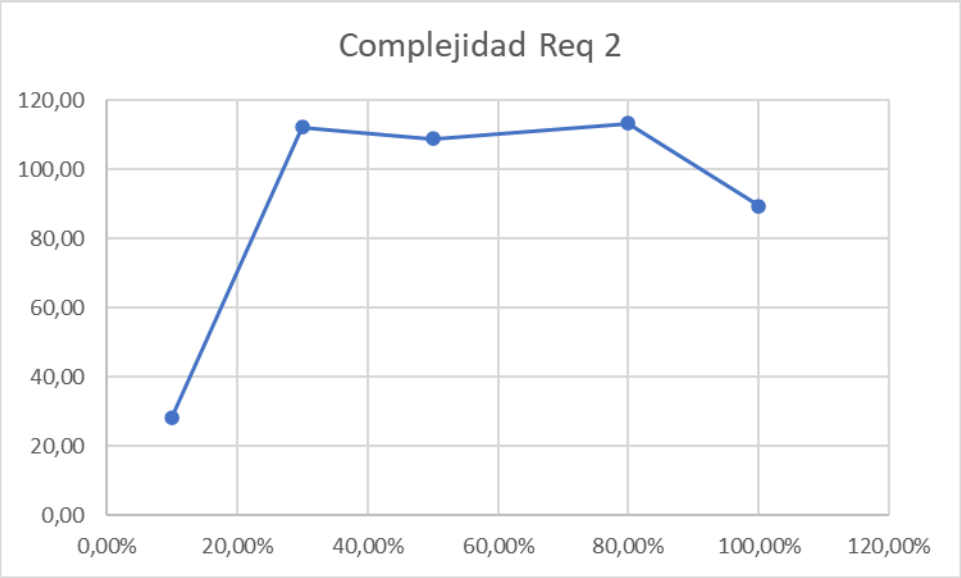
Complejidad:

Paso	**Complejidad**
La primera iteración (el *for*) recorre "jobs", lo cual tomaría un tiempo N, siendo N el tamaño de "jobs".	O(N)
La iteración con el *while* , recorre los elementos de la sublista ofertas_totales	O(N)

Complejidad Final: O(N)

Prueba experimental:

Porcentaje de la muestra [pct]	Tiempo [ms]
10.00%	28.102
30.00%	112.153
50.00%	108.864
80.00%	113.324
100.00%	89.431



Requerimiento 3 (Ana Sofia Trillos)

```python
def req_3(data_structs, empresa, fecha_i, fecha_f):
    td_fechas= lt.newList(datastructure="ARRAY_LIST")

    for nombre in lt.iterator(data_structs["jobs"]):
        if empresa == nombre["company_name"]:
            fecha_convertida = datetime.strptime(nombre["Published_at"], "%Y-%m-%dT%H:%M:%S.%fZ").strftime("%y-%m-%d")
            fecha_I = datetime.strptime(fecha_i, "%Y-%m-%d")
            fecha_F = datetime.strptime(fecha_f, "%Y-%m-%d")
            if fecha_I <= fecha_convertida and fecha_F >= fecha_convertida:
                lt.addFirst(td_fechas, nombre)

    shs.sort(td_fechas,organiza_fecha_pais)

    j_ofertas= lt.newList(datastructure="ARRAY_LIST")
    m_ofertas= lt.newList(datastructure="ARRAY_LIST")
    s_ofertas= lt.newList(datastructure="ARRAY_LIST")
    #listas pfertas j,m,s
    for oferta in lt.iterator(td_fechas):
        if oferta["Experience_level"] == "junior":
            lt.addLast(j_ofertas, oferta )
        elif oferta["Experience_level"] == "mid":
            lt.addLast(m_ofertas, oferta )
        elif oferta["Experience_level"] == "senior":
            lt.addLast(s_ofertas, oferta )
    ofer= lt.size(td_fechas)
    ju= lt.size(j_ofertas)
    mi= lt.size(m_ofertas)
    se= lt.size(s_ofertas)
    return ofer, ju, mi, se, td_fechas
```

Operación	**complejidad**
En el primer for se itera sobre “jobs” en la cual depende del número de elementos n. por medio de los if se convierte el formato del tiempo y confirmar que las fechas si estén entre el rango, agregando al principio de una nueva lista aquellas que si entren.	**O(n)**
Shell sort para ordenar las fechas y el país con ayuda de la función auxiliar *organiza_fecha_pais*	**O (n Log n)**
Creaciones listas vacías para almacenar datos del nivel de experiencia	**O (1)**
For que itera sobre todas las fechas para clasificar depende del nivel de experiencia	**O(n)**

Complejidad Final del código es O (n Log n) debido a la forma en la que se deben organizar las fechas por país.

Requerimiento 4 (Sara García)

Ofertas publicadas en un país específico durante un periodo de tiempo

```python
def paisRangoT(data_structs, codPais, datei, datef):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    Pos_datei = searchFecha(data_structs, datei, True)
    Pos_datef = searchFecha(data_structs, datef, False)

    size_lst_fechas = Pos_datef - Pos_datei

    lst_fechas=lt.subList(data_structs["jobs"],Pos_datei, size_lst_fechas)
    rango = lt.newList("ARRAY_LIST")

    for job in lt.iterator(lst_fechas):
        if job["country_code"] == codPais:
            lt.addLast(rango, job)


    empresas = lt.newList("ARRAY_LIST")
    for job in lt.iterator(rango):
        if not(lt.isPresent(empresas, job["company_name"])):
            lt.addLast(empresas, job["company_name"])
```

Complejidad:

Paso	**Complejidad**
La primera parte del requisito busca las fechas inicial y final por medio de una función externa que realiza una búsqueda binaria.	O(2logN)
Luego, utilizando los límites establecidos, crea una lista de solo las fechas en este rango.	O(1), al ser un arreglo, llegar a las posiciones es constante.

Después, se crea una lista llamada rango. Por medio del primer for, se itera la lista de fechas, añadiendo a rango únicamente aquellas que están en la ciudad que se desea buscar.	O(N)
El siguiente paso busca crear una lista que contenga las empresas (sin repetirse) que hay en la lista general. De esta forma, al conocer el tamaño de la lista de empresas, se podrá saber cuántas empresas hay en ese país durante ese periodo de tiempo. Esto se hace con un for que recorre la lista "rango". No obstante, para evitar que se repita una empresa, se utiliza lt.isPresent; esta función recorre la lista de empresas verificando si un elemento está. Por lo tanto, en esta parte de la función hay dos iteraciones anidadas.	O(N^2)

```
372
373     ciudades = {}
374     for job in lt.iterator(rango):
375         if job["city"] not in ciudades:
376             ciudades[job["city"]] = 1
377         elif job["city"] in ciudades:
378             ciudades[job["city"]] += 1
379
380     if len(ciudades) != 0:
381         mayor = 0
382         for city in ciudades:
383             if ciudades[city] > mayor:
384                 mayor = ciudades[city]
385                 city_may = city
386
387         menor = mayor
388         for city in ciudades:
389             if ciudades[city] < menor:
390                 menor = ciudades[city]
391                 city_men = city
392         if menor == mayor:
393             menor = mayor
394             city_men = city_may
395
396     else:
397         city_may = 0
398         mayor = 0
399         city_men = 0
400         menor = 0
401
402
403     cities = [city_may, mayor, city_men, menor]
404
405     rango_ord = shs.sort(rango, sortCrit_companyName_MenorMayor)
406     rango_ordenado = mes.sort(rango_ord, sortCrit_JobDates)
407
408     return rango_ordenado, empresas, cities
```

Paso	**Complejidad**
Luego, para conocer las ciudades con mayor y menor número de ofertas, se crea un diccionario que contiene las ciudades como llave y el número de veces que aparecen como valor. Este diccionario se crea mediante una iteración de la lista “rango” en la línea 374.	O(N)
Para hallar cuál de las ciudades tiene mayor y menor número de ofertas se hacen dos iteraciones separadas. Una que busca el mayor y otra para el menor.	O(N), ambas búsquedas son unas iteraciones con *for* simple (no anidado), por lo que ambas serían 2N, pero en notación BigOh queda O(N).
Se realiza un ordenamiento con Shell Sort por nombre de las empresas, dado que tiene buena eficiencia de tiempo y no tiene tan alta complejidad. Esto busca que, si dos ofertas fueron publicadas en la misma fecha, queden ordenadas alfabéticamente por nombre de empresa.	O(N^3/2)

Por último, realizo un ordenamiento con Merge Sort por fecha de publicación. Esto se hace para que las ofertas queden ordenadas cronológicamente. Se escoge Merge dado que es **estable**. Por lo tanto, cuando encuentre dos ofertas publicadas en la misma fecha, las ordenará según la organización previo, que por el paso anterior sería alfabética.	O(NLogN)

Complejidad Final: O(N^2)

Prueba experimental:

Porcentaje de la muestra [pct]	Tiempo [ms]
10.00%	42.494
30.00%	324.472
50.00%	393.188
80.00%	318.839
100.00%	420.350

Requerimiento 5 (Daniela González)

```
335
336     Posjob_fechaInicial=searchFecha(data_struct, fecha_inicial, True)
337     Posjob_fechaFinal=searchFecha(data_struct, fecha_final, False)
338
339     size_sublistFechas=Posjob_fechaFinal-Posjob_fechaInicial
340
341     sublistFechas=lt.subList(data_struct["jobs"],Posjob_fechaInicial, size_sublistFechas)
342     sublistCities=lt.newList("ARRAY_LIST")
343     list_para_empresas=lt.newList("ARRAY_LIST")
344
345     for job in lt.iterator(sublistFechas):
346         if job["city"]==ciudad:
347             lt.addLast(sublistCities, job)
348             lt.addLast(list_para_empresas, job)
349
350     #Ordeno alfabéticamente
351     sublistCities_alfabetica=shs.sort(sublistCities, sortCrit_companyName_MenorMayor)
352     #Ordeno otra vez por fechas pero esta vez usando merge que es estable para mantener el orden alfabético
353     sublistCities_final_sort=mes.sort(sublistCities_alfabetica, sortCrit_JobDates)
354
355     jobsxempresas=newList_jobsxempresas(list_para_empresas)
356
```

Paso	**Complejidad**
En la primera parte del requisito 5, se hace un binary search usando searchFecha() para buscar la primera fecha del rango pedido por el usuario y la última fecha.	O(2logN)
Se saca un sublist usando las posiciones de esas fechas.	Ya que jobs es cargado como un array list, llegar a esas posiciones es O(1)
Itero por el sublist, añadiendo al final de sublistCities las ofertas de la ciudad que busco	O(N) Aunque ya que primero se hizo el extracto de las ofertas del rango, si el rango no son todas las fechas del archivo, será un poco menor el tiempo que ON
Ordeno alfabéticamente por nombre de empresas a sublistCities usando shellsort	O(N^3/2) Como hay tantas ciudades repetidas, uso shellsort que terminó de segundo cuando habían few unique en https://www.toptal.com/developers/sorting-algorithms
Ordeno con merge sort por fechas porque es estable, por ende, mantendrá el orden alfabético obtenido del shellsort para los trabajos que tienen la misma fecha	O(NLogN)
Llamo a newList_jobsxempresas para sacar una lista de las empresas (es una función auxiliar que facilita contar las empresas, ver sección funciones auxiliares para detalles de complejidad y funcionamiento)	O(N^3/2)

```
    jobsxempresas=newList_jobsxempresas(list_para_empresas)

    menorNum=300000
    mayorNum=0
    #Recorro la lista de jobsxempresas para sacar el menor y mayor
    for empresa in lt.iterator(jobsxempresas):
        if empresa[1]>mayorNum:
            mayorNum=empresa[1]
            mayorNombre=empresa[0]
        if empresa[1]<menorNum:
            menorNum=empresa[1]
            menorNombre=empresa[0]

    total_empresas=lt.size(jobsxempresas)

    ans=[sublistCities_final_sort, mayorNombre, mayorNum, menorNombre, menorNum, total_empresas]

    return ans
```

Paso	Complejidad
Recorro la lista de empresas, buscando cual es la que tiene menos ofertas y cual tiene más ofertas	O(M), siendo M la cantidad de empresas que hay en el archivo. Ya que las empresas se repiten más de una vez, M<<N y se puede ignorar para la complejidad final,
Saco el size de la lista de empresas para saber cuantas empresas hay en total	O(1)

Complejidad final del requisito 5: ON^(3/2)

Prueba experimental:

Porcentaje de la muestra [pct]	Tiempo [ms]
10.00%	1402.897
30.00%	3271.230
50.00%	6529.772
80.00%	6218.178
100.00%	5178.409

Requerimiento 6

Dada a la gran cantidad de pasos y partes del requisito 6, solo se comentará las complejidades que son diferentes a O constantes. Los addLast, deleteLast, y get elements son hechos en array lists entonces tienen complejidad O(1). Se decidió priorizar tiempo y rapidez sobre espacio.

```
398
399     Posjob_fechaInicial=searchFecha(data_structs, fecha_inicial, True)
400     Posjob_fechaFinal=searchFecha(data_structs, fecha_final, False)
401
402     size_sublistFechas=Posjob_fechaFinal-Posjob_fechaInicial
403
404     sublistFechas=lt.subList(data_structs["jobs"],Posjob_fechaInicial, size_sublistFechas)
405     list_exp_pais = lt.newList("ARRAY_LIST")
406
407     if codPais != None:
408         for job in sublistFechas["elements"]:
409             if job["experience_level"] == exp_level and job["country_code"] == codPais:
410                 lt.addLast(list_exp_pais, job)
411
412     else:
413         for job in sublistFechas["elements"]:
414             if job["experience_level"] == exp_level:
415                 lt.addLast(list_exp_pais, job)
416
```

Paso	Complejidad
Uso de binary search para sacar las ofertas del rango de fechas (líneas 399-405)	O(2LogN)
Recorro las ofertas del rango de fechas para filtrar y obtener solamente los trabajos del nivel de experiencia y país pedidos (407-415)	O(N)

```
417     #Sorting de ciudades de mayor a menor para que después List Cities quede en orden alfabético
418     sorted_Filtrada=shs.sort(list_exp_pais, sortCrit_Cities_MenorMayor)
419
420     #Creo una lista de listas de ciudades
421     ListCities=creador_lista_de_listas(sorted_Filtrada,"city")
422
423     #Sorting por numero de ofertas. Uso de merge porque es estable para mantener orden alfabético
424     sorted_ListCities=mes.sort(ListCities, sort_crit_sizeListas_MayorMenor)
425
426     #Si las ciudades quqe aplican a los criterios son menor que N, añado todas las quqe aplican
427     if lt.size(sorted_ListCities)<numCiudades:
428         size_sublistN_cities=lt.size(sorted_ListCities)
429         sublistN_cities=lt.subList(sorted_ListCities,1,size_sublistN_cities)
430     #Saco las primeras N ciudades con mayorees ofertas
431     else:
432         sublistN_cities=lt.subList(sorted_ListCities,1,numCiudades)
433         size_sublistN_cities=numCiudades
434
```

Paso	Complejidad
Sort de las ofertas alfabetico según las ciudades (408).	O(N^3/2)
Llamo creador_lista_de_listas(). Convierto esa lista de ofertas ordenadas en una lista de listas, donde los elementos son listas para cada ciudad y cada una de ellas contiene los diccionarios de ofertas (mirar funciones auxiliares para detalles de complejidad y funcionamiento) (421).	O(N)
Merge sort de la lista de ciudades usando el tamaño de las listas (osea la cantidad de ofertas que tiene cada una) como criterio de ordenamiento. Se usó merge sort porque es	O(MLogM) M es igual al número de ciudades que hay. Ya que las ciudades se repiten mucho más

estable para mantener el orden alfabético que traía (424).	que las empresas, M<<N y se puede ignorar para la complejidad final,
El ordenamiento por list size del paso anteiror permite que sacar las N primeras ciudades de sorted_ListCities sean las N ciudades con más ofertas (426-433).	O(1)

```
448         for CityList in lt.iterator(sublistN_cities):
449             #Para sacar el promedio despues
450             sum_salarios=0
451 
452             element["Ciudad"]=CityList["elements"][0]["city"]
453             element["Numero_total_ofertas"]=lt.size(CityList)
454             sumatoria_jobs+=lt.size(CityList)
455 
456             jobsxempresa=newList_jobsxempresas(CityList)
457             element["Numero_total_empresas"]=lt.size(jobsxempresa)
458 
459             #Recorro la lista de jobsxempresas para sacar la empresa con mas ofertas
460             mayorNum=0
461             for empresa in lt.iterator(jobsxempresa):
462                 if empresa[1]>mayorNum:
463                     mayorNum=empresa[1]
464                     mayorNombre=empresa[0]
465 
466             element['Empresa_con_mas_ofertas']=mayorNombre,"con",mayorNum,"ofertas"
467 
468             #Recorro cada trabajo para después sacar empresas totales y sacar info de salarios
469             peor_oferta=[None, 90000000]
470             mejor_oferta=[None, 0]
471             sizeCityList=lt.size(CityList)
472 
473             for job in lt.iterator(CityList):
474                 lt.addLast(jobs_list, job)
475 
476                 #El try para que no saque error si esa oferta no tenía salario
477                 try:
478                     salarioJob_promedio=(int(job["salary_from"])+int(job["salary_to"]))/2
479                     sum_salarios+=salarioJob_promedio
480 
```

```
481                     if peor_oferta[1]>salarioJob_promedio:
482                         peor_oferta[1]=salarioJob_promedio
483                         peor_oferta[0]=job["title"]
484                     if mejor_oferta[1]<salarioJob_promedio:
485                         mejor_oferta[1]=salarioJob_promedio
486                         mejor_oferta[0]=job["title"]
487                 except:
488                     #En ese caso no se tendra en cuenta para el promedio esa oferta
489                     sizeCityList-=1
490 
491 
492             salarioCity_promedio=sum_salarios/sizeCityList
493             sumatoria_salario_totales+=salarioCity_promedio
494 
495             element["Salario_promedio"]=salarioCity_promedio
496             element['Peor_oferta']=peor_oferta
497             element["Mejor_oferta"]=mejor_oferta
498 
499             lt.addLast(ultima_lista, element.copy())
500 
501     empresasList=newList_jobsxempresas(jobs_list)
502     empresas_totales=lt.size(empresasList)
```

Pasos del ciclo 448-502	**Complejidad**
Se itera cada lista de ciudades para sacar la información de cada ciudad en un diccionario. Ese diccionario es el que va a ser utilizado para imprimir la información de la ciudad en el tabulate.	O(M) M es igual al número de ciudades que hay. Ya que las ciudades se repiten mucho más que las empresas, M<<N y se puede ignorar para la complejidad final,
Se llama la función jobsxempresa() (Ver funciones auxiliares para detalles de complejidad y funcionamiento) (456).	O(M* N^3/2)

Se recorre la lista de empresas para sacar la empresa con más ofertas (460-466).	O(M*E) E es igual al número de empresas que hay.
Recorro cada trabajo de la ciudad para sacar la información de los salarios. Ya que en la carga de datos se añadió otros dos keys a cada trabajo poniendo el salario mínimo y máximo, no es necesario acceder al documento de employment types. También se añade cada uno de estos trabajos a una lista adicional, para después usar esta lista como parametro de la función newList_jobsxempresas() y contar el número total de empresas, ya que no se puede hacer directamente con nuestra lista de listas. (473-489)	O(M*N)

```
501     empresasList=newList_jobsxempresas(jobs_list)
502     empresas_totales=lt.size(empresasList)
503
504     #Tuple del nombre de la ciudad y su cantidad de ofertas
505     ciudad_más_ofertas=lt.firstElement(ultima_lista)["Ciudad"], lt.firstElement(ultima_lista)["Numero_total_ofertas"]
506     ciudad_menos_ofertas=lt.lastElement(ultima_lista)["Ciudad"], lt.lastElement(ultima_lista)["Numero_total_ofertas"]
507
508     if codPais != None:
509         salario_promedio_total=round((sumatoria_salario_totales/size_sublistN_cities))
510     else:
511         salario_promedio_total=None
512
513     ans=[ultima_lista, size_sublistN_cities, empresas_totales, sumatoria_jobs, salario_promedio_total,
514          ciudad_más_ofertas, ciudad_menos_ofertas]
515
```

Paso	**Complejidad**
Se llama newList_jobsxempresas para sacar la información de todas las empresas de todos los trabajos (501).	O(N^3/2)

La complejidad más grande pareciera ser cuadrática en O(M*N*LogN), sin embargo, ya que el número de ciudades es muchísimo menor que el número de trabajos total, se puede despreciar esta M.

Consecuentemente, quedaría una complejidad más similar a O(N^3/2) (la segunda complejidad más grande que viene del shell sort) que a una cuadrática.

Prueba experimental:

Porcentaje de la muestra [pct]	**Tiempo [ms]**
10.00%	868.831
30.00%	3201.534
50.00%	8320.483
80.00%	10494.418
100.00%	12034.15

Requerimiento 7

Dada a la gran cantidad de pasos y partes del requisito 7, solo se comentara las complejidades que son diferentes a O constantes. Los addLast, deleteLast, y get elements son hechos en array lists entonces tienen complejidad O(1). Se decidió priorizar tiempo y rapidez sobre espacio.

```
619
620     Posjob_fechaInicial=searchFecha(data_structs, fecha_inicial, True)
621     Posjob_fechaFinal=searchFecha(data_structs, fecha_final, False)
622
623     size_sublistFechas=Posjob_fechaFinal-Posjob_fechaInicial
624
625     sublistFechas=lt.subList(data_structs["jobs"],Posjob_fechaInicial, size_sublistFechas)
626     listPaises = lt.newList("ARRAY_LIST")
627
628     sorted_sublistFechas=shs.sort(sublistFechas, sort_crit_Paises_MenorMayor)
629
630     #Creo una lista de listas de países que cada uno contiene sus ofertas de trabajo
631     listPaises=creador_lista_de_listas(sorted_sublistFechas, "country_code")
632
633     #Sort de mayor a menor ofertas
634     sorted_listPaises=shs.sort(listPaises, sort_crit_sizeListas_MayorMenor)
635     #Saco los N paises con mayores ofertas
636     sublist_NPaises=lt.subList(sorted_listPaises, 1, numPaises)
637
```

Paso	**Complejidad**
Uso de binary search para sacar una sublista de las ofertas en el rango de fechas (líneas 620-625)	O(2LogN)
Uso shell sort para ordenar por nombre del país, esto permitira después usar creador_lista_de_listas() (628).	O(N^3/2)
Llamo creador_lista_de_listas(). Convierto esa lista de ofertas en una lista de listas, donde los elementos son listas para cada país y cada una de ellas contiene los diccionarios de ofertas (mirar funciones auxiliares para detalles de complejidad y funcionamiento) (631).	O(N)
Uso de shell sort para ordenar por cantidad de ofertas de cada país y sacar los países con más ofertas (633-636).	O(N^3/2)

```
637
638     #contador para change info
639     pos=1
640     #Quiero que cada trabajo sea sorteado en una lista dependiendo de su experience level
641     for pais in lt.iterator(sublist_NPaises):
642         sorted_jobs_byExp=shs.sort(pais, sortCrit_expLevel_MenorMayor)
643 
644         #Hago que el pais sea una lista de listas de experience level
645         listExpLevels_delPais=creador_lista_de_listas(sorted_jobs_byExp,"experience_level")
646         lt.changeInfo(sublist_NPaises, pos, listExpLevels_delPais)
647 
648         pos+=1
649 
```

Paso	**Complejidad**
Se recorre cada lista de país, voy a crear tres listas que contienen las ofertas divididas por nivel de experiencia.	O(M) M es igual al número de países que el usuario quiere pedir.
Se hace ordenamiento alfabético de las ofertas de cada país con shell sort con base al nivel de experiencia para poder usar creador_lista_de_listas().	O(M*N^3/2) Ya que M es un número mucho más menor que N porque hay menos países que trabajos, se puede despreciar la M y quedaría O(N^3/2).
Se llama a creador_lista_de_listas() para crear las tres listas de experiencias con los trabajos de cada una.	O(M*N), que por las razones previa se simplifica a O(N).

```
655     for pais in lt.iterator(sublist_NPaises):
656         paisImpresion=lt.newList("ARRAY_LIST")
657 
658         for expLevel_List in lt.iterator(pais):
659             #Cada expLevelImpresion va a ser una fila en la tabla que le printea al usuario
660             expLevelImpresion={"País": None, "Nivel de experiencia": None, "Numero_habilidades_solicitadas":None, "Habilidad_m
661                                "Promedio_nivel_mínimo_de_hablidades":None, "Numero_empresas_totales":None, "Empresa_mayor_ofertas":[0,1
662                                "Empresa_menor_ofertas":[0,1], "Numero_empresas_multisedes":None
663                                }
664 
665             list_habilidades=lt.newList("ARRAY_LIST")
666             jobsxempresas=newList_jobsxempresas(expLevel_List)
667 
668             expLevelImpresion["Nivel de experiencia"] = (lt.getElement(expLevel_List, 0))["experience_level"]
669             expLevelImpresion["País"] = (lt.getElement(expLevel_List, 0))["country_code"]
670 
671             #Recorro la lista de jobsxempresas para sacar el menor y mayor
672             menorNum=300000
673             mayorNum=0
674             for empresa in lt.iterator(jobsxempresas):
675                 if empresa[1]>mayorNum:
676                     mayorNum=empresa[1]
677                     mayorNombre=empresa[0]
678                 if empresa[1]<menorNum:
679                     menorNum=empresa[1]
680                     menorNombre=empresa[0]
681 
682             #Añado al diccionario
683             expLevelImpresion["Numero_empresas_totales"]=lt.size(jobsxempresas)
684             expLevelImpresion["Empresa_mayor_ofertas"]=[mayorNombre, mayorNum]
685             expLevelImpresion["Empresa_menor_ofertas"]=[menorNombre, menorNum]
686 
```

```
686 
687             numEmpresasMultisede=0
688 
689             #Para sacar numero de empresas con más de una sede, añadir a la lista de trabajos totales, y crear lista de habilidades
690             for job in lt.iterator(expLevel_List):
691                 #Busco si el job tiene multilocation
692                 multiLocationJob=searchJob_byID(data_structs["multilocation_resumido"],job["id"])
693                 if multiLocationJob!=-1:
694                     numEmpresasMultisede+=1
695 
696                 #Añado a lista toatl de ofertas
697                 lt.addLast(listTotalJobs,job)
698 
699                 #Añado a lista de habilidades
700                 skillJob=searchJob_byID(data_structs["skills"],job["id"])
701                 if skillJob!=-1:
702                     lt.addLast(list_habilidades, skillJob)
703 
704             expLevelImpresion["Numero_empresas_multisedes"]=numEmpresasMultisede
705 
```

```
707             #Para sacar las habilidades de los trabajos
708             jobsxhabiliades=newList_jobsxhabilidades(list_habilidades)
709 
710             #Para sacar menor y mayor habilidad y promedio de nivel
711             topSkill_Num=0
712             topSkill_Name=None
713             lastSkill_Num=90000
714             lastSkill_Name=None
715             sum_skillLevel=0
716             for habilidad in lt.iterator(jobsxhabiliades):
717                 if habilidad[1]>topSkill_Num:
718                     topSkill_Num=habilidad[1]
719                     topSkill_Name=habilidad[0]
720                 if habilidad[1]<lastSkill_Num:
721                     topSkill_Num=habilidad[1]
722                     topSkill_Name=habilidad[0]
723                 sum_skillLevel+=habilidad[2]
724 
725             #Añado al diccionario
726             expLevelImpresion["Promedio_nivel_mínimo_de_hablidades"]=sum_skillLevel/lt.size(jobsxhabiliades)
727             expLevelImpresion["Habilidad_mas_solicitada"]=[topSkill_Name, topSkill_Num]
728             expLevelImpresion["Habilidad_menos_solicitada"]=[lastSkill_Name, lastSkill_Num]
729             expLevelImpresion["Numero_habilidades_solicitadas"]=lt.size(jobsxhabiliades)
730 
731 
732             lt.addLast(paisImpresion, expLevelImpresion)
733 
734         lt.addLast(listImpresion, paisImpresion)
```

Pasos del ciclo de líneas 655-735	Complejidad

Dentro de este ciclo, se itera por la lista principal que contine la lista de cada país, que a su vez contiene las tres listas de de los niveles de experiencia de ese pais, las cuales contienen las ofertas de trabajo.	O(M*E), donde M es el número de países, E el número de niveles de experticia, y N el número de trabajos. E siempre es 3 y M es un número mucho menor que N ya que muchos trabajos tienen los mismos países. Por ende, la iteración de "for pais in lt.iterator(sublist_NPaises): for expLevel_List in lt.iterator(pais):" Termina siendo más similar a O(Constante) en vez de cuadrática.
Se llama new_list_jobsxhabilidades tomando toda la información de ese nivel de experticia (666). Ver funciones auxiliares para análisis de esa función.	O(N^3/2)
Recorro la lista de jobsxempresas para sacar la empresa con menor y mayor ofertas de ese nivel (671-680).	En el peor caso escenario de que cada trabajo tuviera una empresa distinta, lo cual no sucede porque las empresas se repiten, pero digamos en hipotésis que no se repitieran, sería complejidad de O(N).
En el ciclo de 690-702 recorro cada oferta de trabajo de la lista de experiencias para sacar la información de las habilidades, sacar el numero de empresas con multiples sedes, y añadir a la lista de trabajos totales. Se hace busqueda binaria en 692 para encontrar si ese trabajo tiene está en el archivo multilocationResumido. Este archivo se creo en la carga de datos y solo guardaba los trabajos que aparecían más de una vez en el archivo multilocation, lo cual significaba que tenían más de una sede. Por ende, si el trabajo es encontrado en multilocationResumido, tiene más de una sede. En 700 se hace busqueda binaria en el archivo de skills para sacar las skills de ese trabajo. Estas se guardan en list_habilidades que después va a ser utilizada para contar las habilidades totales. MultilocationResumido y Skills fueron ordenados en la carga de datos por ID para poder hacer esta busqueda binaria. La lista de trabajos totales es una lista con todas las ofertas para después usar esta lista como parametro de la función	El recorrido por cada trabajo implica O(N), que se convierte en O(N*2LogN) ya que se hacen dos busquedas binarias para cada dato.

newList_jobsxciudades() y contar el número total de ciudades, ya que no se puede hacer directamente con nuestra lista de listas.	
Se llama jobsxhabilidades para crear una lista que contiene el conteo de cada habilidad y su nivel promedio exigido. Mirar funciones auxiliares para mayor detalle.	O(N^3/2)
Se reitera por la lista que tiene el conteo de cada habilidad para sacar las habilidades más y menos pedias y el pormedio del nivel requerido.	Suponiendo el peor caso que cada trabajo tiene una skill única, sería O(N), pero como se repiten algunas habilidades esta reiteración termina siendo menor.

```
737     #Para sacar la extra info de ciudades requeridas
738     jobsxciudades=newList_jobsxciudades(listTotalJobs)
739     ciudadMayorJobs=None
740     ciudadMayorJobs_Num=0
741     for ciudad in lt.iterator(jobsxciudades):
742         if ciudad[1]>ciudadMayorJobs_Num:
743             ciudadMayorJobs_Num=ciudad[1]
744             ciudadMayorJobs=ciudad[0]
745
746     #Lista [0]=nombre del pais con más ofertas, [1] su numero de ofertas
747     paisMayor=lt.firstElement(sublist_NPaises)
748     expPaisMayor=lt.firstElement(paisMayor)
749     trabajoMayor=lt.firstElement(expPaisMayor)
750
751     paisMayorOfertas=[trabajoMayor["country_code"], (lt.size(lt.getElement(paisMayor,1))+lt.size(lt.getElement(paisMayor,2))+lt.size(lt.getElement(paisMayor,3)))]
752
753     ans=[listImpresion, lt.size(listTotalJobs), lt.size(jobsxciudades), [ciudadMayorJobs, ciudadMayorJobs_Num], paisMayorOfertas]
754
755     return ans
```

Pasos	**Complejidad**
Se saca la información de las ciudades de todos los trabajos usando la función auxiliar newList_jobsxciudades (738).	O(N^3/2)
Se reitera por la lista que tiene el conteo de cada ciudad para sacar la ciudad con más ofertas de trabajo.	Suponiendo el peor caso que cada trabajo tiene una ciudad única, sería O(N), pero como se repiten algunas ciudades esta reiteración termina siendo menor.

Complejidad total: O(N^3/2) ya que el ciclo anidado de O(M*E*N) se reduce a O(N).

Prueba experimental:

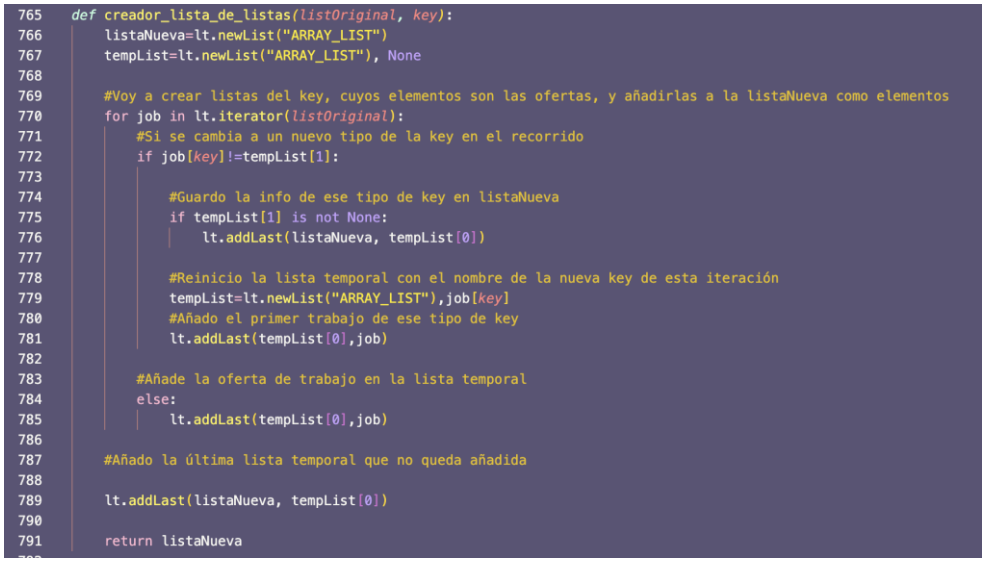
Porcentaje de la muestra [pct]	**Tiempo [ms]**
10.00%	5325.094
30.00%	21137.008
50.00%	40741.863
80.00%	46694.326
100.00%	54385.271

Funciones auxiliares:

Las funciones de searchFecha(), SearchJob_byID(), y SearchSkill() se hacen usando binary search lo cual tiene una complejidad de O(LogN) a diferencia de linear search que O(N).

```
883
884     #Hago sort alfabético por las ciudades
885     list_ciudades_ordenadas=shs.sort(lst, sortCrit_Cities_MenorMayor)
886     list_jobsxciudades=lt.newList("ARRAY_LIST")
887
888     #Recorro la lista de empresas ordenada para sumar las ofertas de cada empresa
889     for job in lt.iterator(list_ciudades_ordenadas):
890
891         if lt.size(list_jobsxciudades)==0:
892             #lt.lastElement saca error si la lista esta vacia, lo cual sucede en la primera iteración, y en este caso añado de una sin compararar
893             lt.addLast(list_jobsxciudades, [job["city"], 1])
894
895         else:
896             last_element_jobsxciudades=lt.lastElement(list_jobsxciudades)
897
898             #Si el trabajo pertence a una empresa distinta que la del elemento pasado, añade uno nuevo
899             if last_element_jobsxciudades[0]!=job["city"]:
900                 lt.addLast(list_jobsxciudades, [job["city"], 1])
901
902             #Si el trabajo tiene la misma empresa que el elemento pasado, le suma 1 a la cantidad de trabajos de la empresa
903             elif lt.size(list_jobsxciudades)>0:
904                 last_element_jobsxciudades[1]+=1
905                 #Es por este changeInfo que escogí array list
906                 lt.changeInfo(list_jobsxciudades, lt.size(list_jobsxciudades), last_element_jobsxciudades)
907
908     return list_jobsxciudades
909
```

Se crearon las funciones de newList_jobsxhabilidades, newList_jobsxciudades, y newList_jobsxempresas para hacer conteos de características de los trabajos de con complejidad O(N^3/2), por ende funcionan de la misma manera. Primero hacen un shell sort con el sort criteria de la característica que quieren contar, lo cual es complejidad O(N^3/2). Ya que ya están ordenados los datos, no se tiene que ir buscando en la lista del conteo la característica por cada trabajo. Se puede hacer la suma directamente al último dato de la lista, y cuando haya un cambio de característica, se crea un nuevo elemento a la lista de conteo. De esta manera, siempre se hace addLast que tiene complejidad O(1).



```
765  def creador_lista_de_listas(listOriginal, key):
766      listaNueva=lt.newList("ARRAY_LIST")
767      tempList=lt.newList("ARRAY_LIST"), None
768
769      #Voy a crear listas del key, cuyos elementos son las ofertas, y añadirlas a la listaNueva como elementos
770      for job in lt.iterator(listOriginal):
771          #Si se cambia a un nuevo tipo de la key en el recorrido
772          if job[key]!=tempList[1]:
773
774              #Guardo la info de ese tipo de key en listaNueva
775              if tempList[1] is not None:
776                  lt.addLast(listaNueva, tempList[0])
777
778              #Reinicio la lista temporal con el nombre de la nueva key de esta iteración
779              tempList=lt.newList("ARRAY_LIST"),job[key]
780              #Añado el primer trabajo de ese tipo de key
781              lt.addLast(tempList[0],job)
782
783          #Añade la oferta de trabajo en la lista temporal
784          else:
785              lt.addLast(tempList[0],job)
786
787      #Añado la última lista temporal que no queda añadida
788
789      lt.addLast(listaNueva, tempList[0])
790
791      return listaNueva
```

Para la función creador lista de listas, está tiene una complejidad de O(N) ya que recorre cada trabajo dado en la lista original y durante este ciclo solo suma al final de listas y hace comparaciones. Sin embargo, esta función tiene el prerequisito de que la lista que se le ingresa tiene que ya estar ordenada. La lógica detrás de esta función es muy similar a las funciones de newList_jobsxcaracteristica, ya que aprovecha el ordenamiento para no tener que recorrer una lista cada vez que se encuentra un objeto que se quiere sumar.