ANÁLISIS DEL RETO

1. Samuel Reyes, s.reyese@uniandes.edu.co, 202213995.
2. Javier Conde, j.condeo@uniandes.edu.co, 202314362.
3. Zair Montoya, z.montoya@uniandes.edu.co, 202321067.

Requerimiento <<1>>

Descripción

```python
def req_1(control,fecha_inicial,fecha_final):
    """
    Función que soluciona el requerimiento 1
    """
    time_data = control["model"]["jobs_time"]
    data = om.values(time_data,fecha_inicial,fecha_final)
    lista = lt.newList("ARRAY_LIST")
    titulos = ["published_at","title","company_name","experience_level","country_code","city","company_size","workplace_type","Habilidades_solicitadas","longitude","latitude" ]
    contador = 0
    skills = mp.valueSet(control["model"]["skills"])
    diccionario_habilidades = {}
    for id in lt.iterator(skills):
        for habilidad in id:
            nombre = habilidad["name"]
            ids = habilidad["id"]
            if ids not in diccionario_habilidades:
                diccionario_habilidades[ids] =[nombre]
            else:
                diccionario_habilidades[ids].append(nombre)


    for dato in lt.iterator(data):
        for trabajo in dato:
            id = trabajo["id"]
            skill = diccionario_habilidades[id]
            trabajo["Habilidades_solicitadas"] = skill
            lt.addLast(lista,trabajo)
            contador +=1

    sort_function_time(lista)
    respuesta = generacion_respuesta(titulos,lista)
    return contador , respuesta
```

conocer las ofertas laborales publicadas durante un intervalo de fechas específico.

Entrada	• La fecha inicial del periodo a consultar • La fecha final del periodo a consultar
Salidas	• El número total de ofertas laborales publicadas durante las fechas indicadas. • Todas las ofertas laborales publicadas en el intervalo ordenados cronológicamente desde la más reciente a la más antigua.
Implementado (Sí/No)	Si se implementó, Zair Montoya

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
sort	O(n log n)
Iteración	O(n)
Generación_respuesta	O(m)
TOTAL	***O(n log n +m)***

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	**Apple M1**
Memoria RAM	8 GB
Sistema Operativo	macOS Sonoma 14.4.1

Entrada	Tiempo (ms)
small	37889,22
10 pct	59067,74
30 pct	76592,37
80 pct	76948,3
large	158916,98

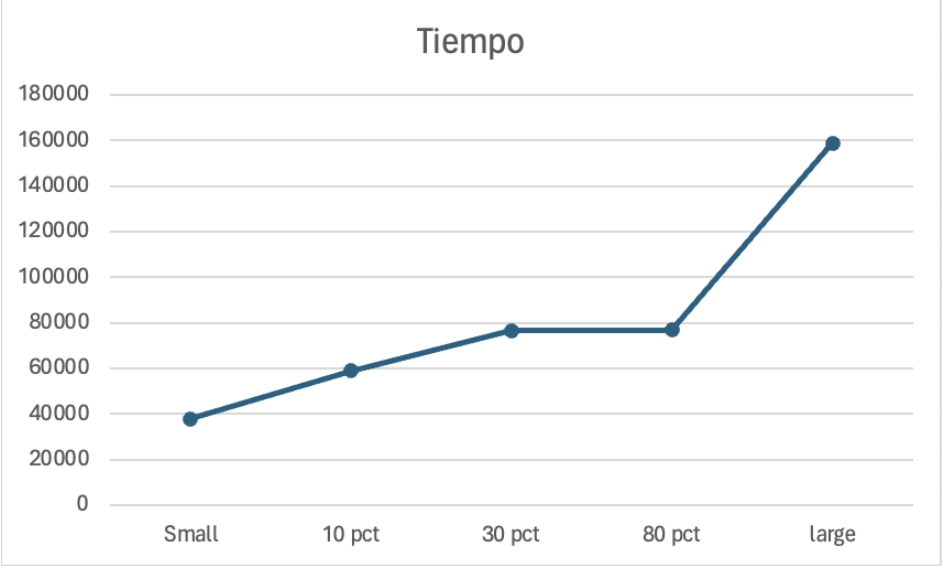
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	37889,22
10 pct	Dato2	59067,74
30 pct	Dato3	76592,37
80 pct	Dato4	76948,3
large	Dato5	158916,98

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad de tiempo de este requerimiento es O(n log n + m), principalmente debido al uso del algoritmo de ordenamiento mergesort al final. A pesar de que las operaciones previas como obtener valores de un orderedmap y agregar elementos a una lista tienen complejidades de O(log n) y O(1) respectivamente, la complejidad está dominada por el paso de ordenamiento con mergesort. Pero se le agrega el m por el uso de la función generación respuesta (esto se ve presente del 1 al 6)

Requerimiento <<2>>

Descripción

```python
def req_2(control,salario_minimo,salario_maximo):
    """
    Función que soluciona el requerimiento 2
    """
    employment = mp.valueSet(control["model"]["employments"])
    skills = mp.valueSet(control["model"]["skills"])
    trabajo = om.valueSet(control["model"]["jobs"])
    diccionario_salario = {}
    diccionario_habilidades = {}
    lista = lt.newList("ARRAY_LIST")
    titulos = ["published_at","title","company_name","experience_level","country_code","city","company_size","workplace_type","salario_minimo","Habilidades_solicitadas", "longitude","latitude"]
    contador = 0
    salario_minimo_anterior = 0
    for id in lt.iterator(employment):
        for dato in id:
            salario_minimo_actual = dato["minimum_usd_salary"]
            ids = dato["id"]
            # Si aún no se ha almacenado ningún salario mínimo para este id
            if ids not in diccionario_salario:
                diccionario_salario[ids] = salario_minimo_actual
            else:
                if  salario_minimo_actual != None and salario_minimo_anterior != None:
                    salario_minimo_anterior = diccionario_salario[ids]
                    # Comparar el salario mínimo actual con el anterior y almacenar el mayor
                    if salario_minimo_actual < salario_minimo_anterior:
                        diccionario_salario[ids] = salario_minimo_actual
                    else:
                        diccionario_salario[ids] = salario_minimo_anterior

    for id in lt.iterator(skills):
        for habilidad in id:
            nombre = habilidad["name"]
            ids = habilidad["id"]
            if ids not in diccionario_habilidades:
                diccionario_habilidades[ids] =[nombre]
            else:
                diccionario_habilidades[ids].append(nombre)


    for job in lt.iterator(trabajo):
        id = job["id"]
        salary = diccionario_salario[id]
        skill = diccionario_habilidades[id]
        if salary == "Unknown" or salary == None:
            puerta = False
        else:
            puerta = int(salario_minimo) <= int(salary) <= int(salario_maximo)
        if puerta == True:
            contador +=1
            job["minimum_usd_salary"] = salary
            job["salario_minimo"]=job["minimum_usd_salary"]
            job["Habilidades_solicitadas"] = skill

            lt.addLast(lista,job)



    sort_function(lista)
    respuesta = generacion_respuesta(titulos,lista)
    return contador , respuesta
```

conocer las ofertas laborales publicadas existentes dentro de un rango de salario mínimo ofertado

Entrada	• El límite inferior del salario mínimo ofertado. • El límite superior del salario mínimo ofertado.
Salidas	• El número total de ofertas laborales publicadas en los rangos salariales requeridos. • Todas ofertas laborales publicadas en el intervalo, ordenados por el salario mínimo ofertado desde el mayor al menor
Implementado (Sí/No)	Si se implementó, Zair Montoya

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

sort	O(n log n)
Iteración	O(n)
Generación_respuesta	O(m)

TOTAL	***O(n log n +m)***

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	**Apple M1**
Memoria RAM	8 GB
Sistema Operativo	macOS Sonoma 14.4.1

Entrada	**Tiempo (ms)**
small	26243,61
10 pct	49261,85
30 pct	54887,57
80 pct	57650,28
large	67885,17

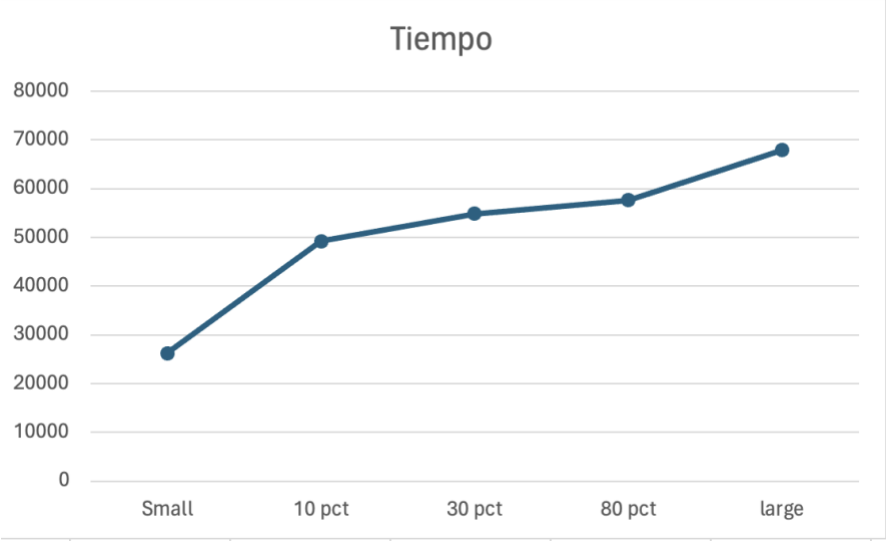
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	26243,61
10 pct	Dato2	49261,85
30 pct	Dato3	54887,57
80 pct	Dato4	57650,28
large	Dato5	67885,17

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento también tiene una complejidad de tiempo de O(n log n). Aunque las iteraciones sobre las estructuras de datos como mapas y orderedmap tienen complejidad lineal O(n), y agregar elementos a una lista es O(1), la complejidad está regida por el paso final de ordenamiento con mergesort que tiene complejidad O(n log n + m).

Requerimiento <<3>>

Descripción

```python
def req_3(control,n,codigo_pais,nivel_experticia):
    """
    Función que soluciona el requerimiento 3
    """
    employment = mp.valueSet(control["model"]["employments"])
    lista = lt.newList("ARRAY_LIST")
    titulos = ["published_at","title","company_name","experience_level","country_code","city","company_size","workplace_type","salario_minimo","Habilidades_solicitadas","longitude","latitude" ]
    contador = 0
    skills = mp.valueSet(control["model"]["skills"])
    trabajo = om.valueSet(control["model"]["jobs"])
    diccionario_salario = {}
    diccionario_habilidades = {}
    salario_minimo_anterior = 0

    for id in lt.iterator(employment):
        for dato in id:
            salario_minimo_actual = dato["minimum_usd_salary"]
            ids = dato["id"]
            # Si aún no se ha almacenado ningún salario mínimo para este id
            if ids not in diccionario_salario:
                diccionario_salario[ids] = salario_minimo_actual
            else:
                if  salario_minimo_actual != None and salario_minimo_anterior != None:
                    salario_minimo_anterior = diccionario_salario[ids]
                    # Comparar el salario mínimo actual con el anterior y almacenar el mayor
                    if salario_minimo_actual < salario_minimo_anterior:
                        diccionario_salario[ids] = salario_minimo_actual
                    else:
                        diccionario_salario[ids] = salario_minimo_anterior

    for id in lt.iterator(skills):
        for habilidad in id:
            nombre = habilidad["name"]
            ids = habilidad["id"]
            if ids not in diccionario_habilidades:
                diccionario_habilidades[ids] =[nombre]
            else:
                diccionario_habilidades[ids].append(nombre)

    for job in lt.iterator(trabajo):
        id = job["id"]
        pais = job["country_code"]
        experticia = job["experience_level"]
        salary = diccionario_salario[id]
        skill = diccionario_habilidades[id]

        if codigo_pais  == pais and nivel_experticia == experticia :

            job["minimum_usd_salary"] = salary
            job["salario_minimo"]=job["minimum_usd_salary"]
            job["Habilidades_solicitadas"] = skill

            lt.addLast(lista,job)

    sort_function_time(lista)

    lista = lt.subList(lista,0, n)
    respuesta = generacion_respuesta(titulos,lista)
    return contador , respuesta
```

consultar las N ofertas laborales más recientes para un país y que requieran un nivel de experiencia específico.

Entrada	• El número (N) de ofertas laborales a consultar • Código del país para la consulta • Nivel de experticia de las ofertas de interés
Salidas	• El número total de ofertas laborales publicadas para un país y que requieran un nivel de experiencia especifico. • Las N ofertas laborales publicadas más recientes que cumplan con las condiciones especificadas.
Implementado (Sí/No)	Si se implementó, Zair Montoya

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

sort	O(n log n)

Iteración	O(n)
Generación_respuesta	O(m)
TOTAL	***O(n log n +m)***

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	**Apple M1**
Memoria RAM	8 GB
Sistema Operativo	macOS Sonoma 14.4.1

Entrada	**Tiempo (ms)**
small	4554,21
10 pct	5449,23
30 pct	8201,01
80 pct	12353,11
large	16308,70

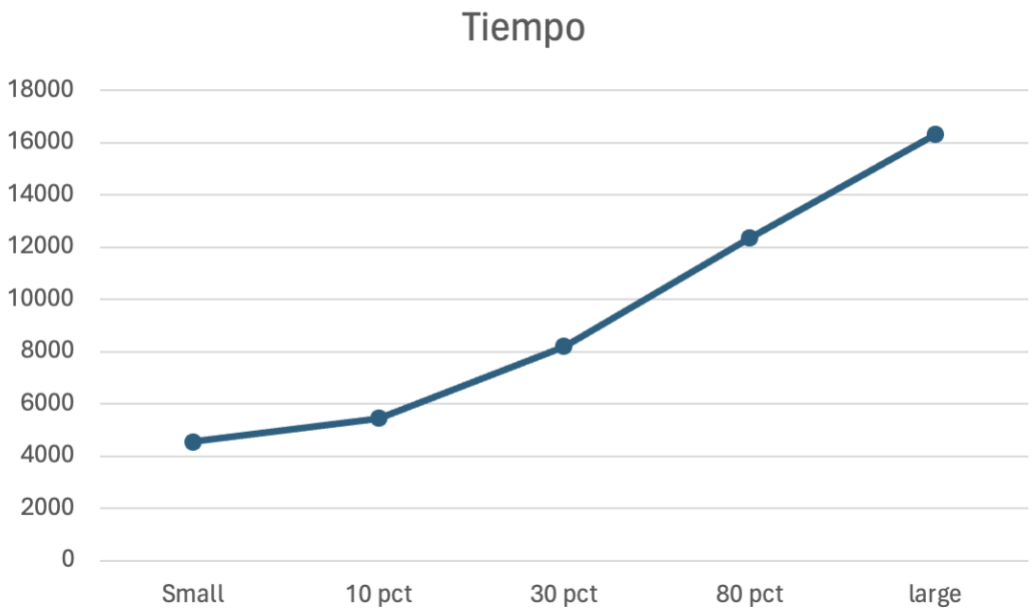
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	4554,21
10 pct	Dato2	5449,23
30 pct	Dato3	8201,01
80 pct	Dato4	12353,11
large	Dato5	16308,70

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Al igual que los anteriores, la complejidad de tiempo de este requerimiento es O(n log n + m). Las iteraciones sobre mapas y orderedmap son O(n), y agregar elementos a una lista es O(1), pero el ordenamiento final con mergesort determina la complejidad total.

Requerimiento <<4>>

Descripción

```python
def req_4(control, n_offers, city_name, job_location):
    """
    Función que soluciona el requerimiento 4
    """
    jobs_values = om.valueSet(control["model"]["jobs"])
    employments_tuples = control["model"]["employments"]
    skills_tuples = control["model"]["skills"]
    ans = lt.newList("ARRAY_LIST")
    titles = ["published_at", "title", "company_name", "experience_level", "country_code", "city", "company_size", "workplace_type", "salary_from", "skills","longitude","latitude"]

    for each_job in lt.iterator(jobs_values):
        if each_job["city"] == city_name and each_job["workplace_type"] == job_location:
            job_id = each_job["id"]

            #Encuentra el salario mínimo ofrecido por trabajo
            employment = mp.get(employments_tuples, job_id)
            min_salary = int(lt.size(jobs_values))

            for i in employment["value"]:
                if i["salary_from"] != "Unknown" and int(i["salary_from"]) < min_salary:
                    min_salary = int(i["salary_from"])

            if min_salary == lt.size(jobs_values):
                min_salary = "Unknown"

            #Encuntra las habilidades solicitadas por trabajo
            skills = mp.get(skills_tuples, job_id)
            skills_ans = []

            for i in skills["value"]:
                skills_ans.append(i["name"])

            #Agrega el trabajo a la lista completa de respuesta
            each_job["salary_from"] = min_salary
            each_job["skills"] = skills_ans
            lt.addLast(ans, each_job)

    size = lt.size(ans)

    resultado_final = lt.newList('ARRAY_LIST')
    a = 0
    for sorted in lt.iterator(ans):
            if a < n_offers:
                lt.addLast(resultado_final,sorted)
                a += 1

    resultado_final = generacion_respuesta(titles,resultado_final)

    return resultado_final, size
```

Consultar las N ofertas laborales más recientes para una ciudad y que requieran un tipo de ubicación de trabajo específico.

Entrada	• El número (N) de ofertas laborales para consulta • Nombre de la ciudad para la consulta • Tipo de ubicación de trabajo.
Salidas	• El número total de ofertas laborales publicadas para la ciudad y con un tipo de ubicación específica. • Las N ofertas laborales publicadas más recientes que cumplan con las condiciones especificadas.
Implementado (Sí/No)	Si se implementó, Samuel Reyes

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
sort	O(n log n)
Iteración	O(n)
Generación_respuesta	O(m)

TOTAL	***O(n log n +m)***

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	**Apple M1**
Memoria RAM	8 GB
Sistema Operativo	macOS Sonoma 14.4.1

Entrada	**Tiempo (ms)**
small	234,39
10 pct	236,26
30 pct	293,45
80 pct	502,12
large	698,24

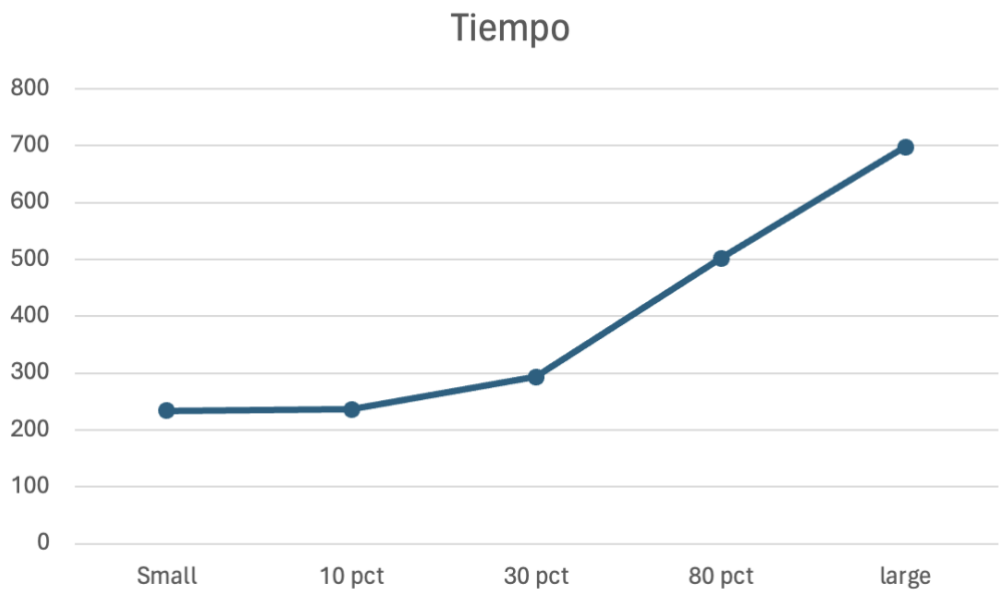
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	234,39
10 pct	Dato2	236,26
30 pct	Dato3	293,45
80 pct	Dato4	502,12
large	Dato5	698,24

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad de tiempo sigue siendo O(n log n + m) en este requerimiento. A pesar de que solo se itera sobre un orderedmap (O(n)) y se agregan elementos a una lista (O(1)), el paso de ordenamiento con mergesort al final es el que domina la complejidad.

Requerimiento <<5>>

Descripción

```
def req_5(n,ls_companysize,li_companysize,skill,ls_companyskill,li_companyskill,control):
    """
    Función que soluciona el requerimiento 5
    """
    jobs = om.values(control['model']['jobs_compsize'],li_companysize,ls_companysize)
    employments = mp.valueSet(control['model']['employments'])
    titles = ["published_at", "title", "company_name", "experience_level", "country_code", "city", "company_size", "workplace_type", "salario_minimo", "habilidades","longitude","latitude"]
    result_list = lt.newList('ARRAY_LIST')
    contador = 0
    for lista in lt.iterator(jobs):
        for dato in lista:
            j = dato["habilidades"]
            for  i in j:
                skill_d = i["skill"]
                level = i["level"]

                if is_number(level):
                    if str(skill_d) == str(skill) and (li_companyskill <= int(level) <= ls_companyskill):
                        lt.addLast(result_list, dato)
                        contador += 1

    size = lt.size(result_list)



    resultado_final = lt.newList('ARRAY_LIST')
    a = 0
    for sorted in lt.iterator(result_list):
            if a < n:
                lt.addLast(resultado_final,sorted)
                a += 1

    resultado_final = generacion_respuesta(titles,resultado_final)
    return contador, resultado_final
```

Consultar las N ofertas más antiguas con un nivel mínimo y máximo para una habilidad solicitada para empresas en un rango de tamaño.

Entrada	• El número (N) de ofertas laborales para consulta. • El límite inferior del tamaño de la compañía. • El límite superior del tamaño de la compañía. • Nombre de la habilidad solicitada. • El límite inferior del nivel de la habilidad. • El límite superior del nivel de la habilidad.
Salidas	• El número total de ofertas laborales publicadas para las compañías que tengan un tamaño en un rango y que requieran una habilidad específica. • Las N ofertas laborales publicadas más antiguas que cumplan con las condiciones especificadas.
Implementado (Sí/No)	Si se implementó, Javier Conde

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
sort	O(n log n)
Iteración	O(n)
Generación_respuesta	O(m)
TOTAL	***O(n log n +m)***

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	**Apple M1**
Memoria RAM	8 GB
Sistema Operativo	macOS Sonoma 14.4.1

Entrada	**Tiempo (ms)**
small	258.25
10 pct	282.14
30 pct	421.54
80 pct	687.79
large	6309.52

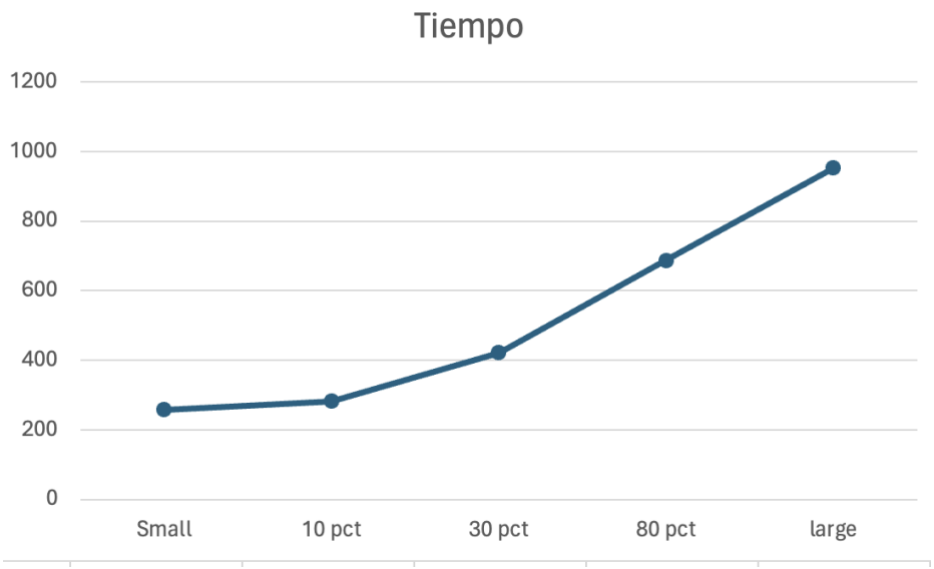
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	258.25
10 pct	Dato2	282.14
30 pct	Dato3	421.54
80 pct	Dato4	687.79
large	Dato5	6309.52

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Nuevamente, la complejidad de tiempo es O(n log n+m). Obtener valores de un orderedmap es O(log n), iterar sobre ellos es O(n), agregar a una lista es O(1), pero el ordenamiento final con mergesort determina la complejidad total.

Requerimiento <<6>>

Descripción

```python
def req_6(control,n,fecha_inicial,fecha_final,salario_minimo,salario_maximo):
    """
    Función que soluciona el requerimiento 6
    """
    employment = mp.valueSet(control["model"]["employments"])
    lista = lt.newList("ARRAY_LIST")
    mapa= mp.newMap(numelements=17,
           prime=109345121,
           maptype='CHAINING',
           loadfactor=4)
    multilocation_list_by_id = []
    titulos = ["published_at", "title", "company_name","experience_level","country_code","city","company_size","workplace_type", "salario_minimo", "Habilidades_solicitadas","longitude","latitude"]
    contador = 0
    skills = mp.valueSet(control["model"]["skills"])
    time_data = control["model"]["jobs_time"]
    data = om.values(time_data,fecha_inicial,fecha_final)
    diccionario_salario = {}
    diccionario_habilidades = {}
    diccionario_ciudades = {}
    salario_minimo_anterior = 0

    for id in lt.iterator(employment):
        for dato in id:
            salario_minimo_actual = dato["minimum_usd_salary"]
            ids = dato["id"]
            # Si aún no se ha almacenado ningún salario mínimo para este id
            if ids not in diccionario_salario:
                diccionario_salario[ids] = salario_minimo_actual
            else:
                if  salario_minimo_actual != None and salario_minimo_anterior != None:
                    salario_minimo_anterior = diccionario_salario[ids]
                    # Comparar el salario mínimo actual con el anterior y almacenar el mayor
                    if salario_minimo_actual < salario_minimo_anterior:
                        diccionario_salario[ids] = salario_minimo_actual
                    else:
                        diccionario_salario[ids] = salario_minimo_anterior

    for id in lt.iterator(skills):
        for habilidad in id:
            nombre = habilidad["name"]
            ids = habilidad["id"]
            if ids not in diccionario_habilidades:
                diccionario_habilidades[ids] =[nombre]
            else:
                diccionario_habilidades[ids].append(nombre)
```

```python
    for dato in lt.iterator(data):
        for job in dato:
            id = job["id"]
            ciudad = job["city"]
            salary = diccionario_salario[id]
            skill = diccionario_habilidades[id]
            if salary == "Unknown" or salary == None:
                puerta = False
            else:
                puerta = int(salario_minimo) <= int(salary) <= int(salario_maximo)
            if puerta == True:
                contador +=1
                job["minimum_usd_salary"] = salary
                job["salario_minimo"]=job["minimum_usd_salary"]
                job["Habilidades_solicitadas"] = skill


                if ciudad  in diccionario_ciudades:
                    diccionario_ciudades[ciudad]+=1
                else:
                    diccionario_ciudades[ciudad]=1


                if mp.contains(mapa, ciudad) and ciudad != None:
                    multilocation_list_by_id = mp.get(mapa, ciudad)['value']
                    multilocation_list_by_id.append(job)
                    mp.put(mapa, ciudad, multilocation_list_by_id)
                elif ciudad != None:
                    multilocation_list_by_id.append(job)
                    mp.put(mapa, ciudad, multilocation_list_by_id)

    total_ciudades = len(diccionario_ciudades)
    diccionario_recortado = dict(sorted(diccionario_ciudades.items(), key=lambda item: item[1], reverse=True)[:n])
    n_ciudades_cumplen = []
    for ciudad_d in diccionario_recortado.keys():
        respuesta_d = {ciudad_d:None}
        n_ciudades_cumplen.append(ciudad_d)
        lista_tabajo = mp.get(mapa,ciudad_d)["value"]

        final = generacion_respuesta_dict(titulos, lista_tabajo)

        respuesta_d[ciudad_d] = final
        lt.addLast(lista,respuesta_d)

    n_ciudades_cumplen = sorted(n_ciudades_cumplen)


    return contador,total_ciudades,n_ciudades_cumplen,lista
```

Consultar las N ciudades que presentan la mayor cantidad de ofertas laborales publicadas entre un par de fechas y que estén en un rango de salario ofertado.

Entrada	• El número (N) de ciudades a consultar.

	• La fecha inicial del periodo a consultar (con formato "%Y-%m-%d"). • La fecha final del periodo a consultar (con formato "%Y-%m-%d"). • El límite inferior del salario mínimo ofertado. • El límite superior del salario mínimo ofertado.
Salidas	• El número total de ofertas laborales publicadas entre un par de fechas y que estén en un rango de salario ofertado. • El número total de ciudades que cumplan con las especificaciones. • Las N ciudades que cumplan las condiciones especificadas ordenadas alfabéticamente. • Para la ciudad con la mayor cantidad de ofertas laborales publicadas, que cumplan con los requisitos
Implementado (Sí/No)	Si se implementó, Zair Montoya

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
sort	O(n log n)
Iteración	O(n)
Generación_respuesta	O(m)
TOTAL	***O(n log n +m)***

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	**Apple M1**
Memoria RAM	8 GB
Sistema Operativo	macOS Sonoma 14.4.1

Entrada	**Tiempo (ms)**
small	151203,74
10 pct	169894,01
30 pct	206362,6
80 pct	318378,23
large	422354,75

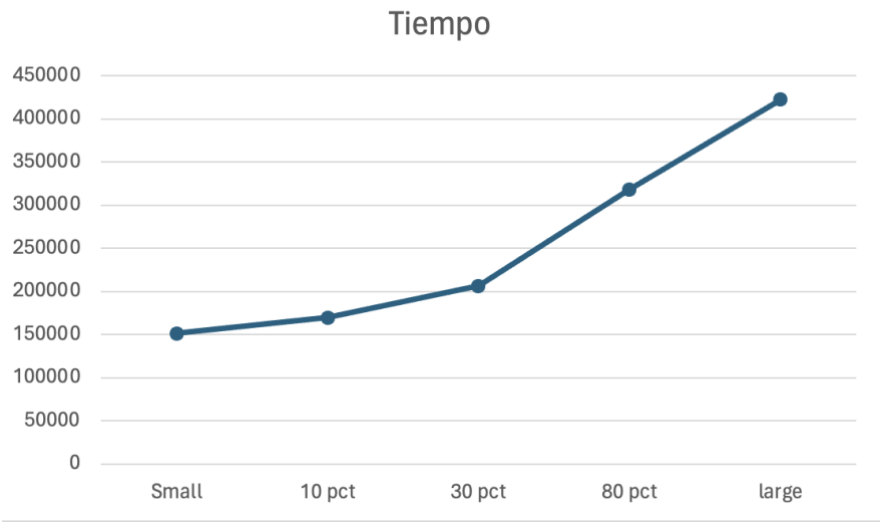
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	151203,74
10 pct	Dato2	169894,01
30 pct	Dato3	206362,6
80 pct	Dato4	318378,23
large	Dato5	422354,75

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento también tiene una complejidad de tiempo de O(n log n + m). Las iteraciones sobre mapas y orderedmap son O(n), obtener valores de un orderedmap es O(log n), agregar a una lista es O(1), pero el paso de ordenamiento con mergesort al final es el que domina la complejidad.

Requerimiento <<7>>

Descripción

```
def req_7(control, year, country_code, property_input):
    """
    Función que soluciona el requerimiento 7
    """
    employments_tuples = control["model"]["employments"]
    time_data = control["model"]["jobs_time"]
    fecha_inicial = year + "-01-01"
    fecha_final = year + "-12-31"
    data = om.values(time_data, fecha_inicial,fecha_final)
    skills_tuples = control["model"]["skills"]
    ans = lt.newList("ARRAY_LIST")
    general_size = 0
    graph_size = 0
    distribution = {}

    titles = ["published_at", "title", "company_name", "country_code", "city", "company_size", "salary_from", "property","longitude","latitude"]

    for i in lt.iterator(data):
        for each_job in i:
            general_size += 1
            if each_job["country_code"] == country_code:
                job_id = each_job["id"]
                skills = mp.get(skills_tuples, job_id)
                skills_ans = []

                #Encuentra el salario mínimo ofrecido por trabajo
                employment = mp.get(employments_tuples, job_id)
                min_salary = lt.size(data)

                for i in employment["value"]:
                    if i["salary_from"] != "Unknown":
                        if int(i["salary_from"]) < min_salary:
                            min_salary = int(i["salary_from"])

                if min_salary == lt.size(data):
                    min_salary = "Unknown"

                each_job["salary_from"] = min_salary
```

```
                if property_input == '1' and each_job["experience_level"] != "Unknown":
                    each_job["property"] = each_job["experience_level"]
                    lt.addLast(ans, each_job)

                    if each_job["experience_level"] in distribution:
                        distribution[each_job["experience_level"]] += 1
                    else:
                        distribution[each_job["experience_level"]] = 1

                if property_input == '2' and each_job["workplace_type"] != "Unknown":
                    each_job["property"] = each_job["workplace_type"]
                    lt.addLast(ans, each_job)

                    if each_job["workplace_type"] in distribution:
                        distribution[each_job["workplace_type"]] += 1
                    else:
                        distribution[each_job["workplace_type"]] = 1

                if property_input == '3':
                    for i in skills["value"]:
                        if i["name"] != "Unknown":
                            skills_ans.append(i["name"])
                            each_job["property"] = skills_ans
                            lt.addLast(ans, each_job)

                            if i["name"] in distribution:
                                distribution[i["name"]] += 1
                            else:
                                distribution[i["name"]] = 1

    graph_size = lt.size(ans)
    ans = generacion_respuesta(titles, ans)

    return ans, general_size, graph_size, distribution
```

Como analista de datos quiero contabilizar las ofertas laborales publicadas para un país y un año específico según alguna propiedad de interés como lo son el nivel de experticia requerido, el tipo de ubicación del trabajo, o habilidad específica.

Entrada	• El año relevante (en formato "%Y"). • Código del país para la consulta (ej.: PL, CO, ES, etc). • La propiedad de conteo (experticia, ubicación, o habilidad).
Salidas	• El número de ofertas laborales publicadas dentro del periodo anual relevante. • El número de ofertas laborales publicadas utilizados para crear el gráfico de barras de la propiedad. • Valor mínimo y valor máximo de la propiedad consultada en el gráfico de barras. • El gráfico de barras con la distribución de las ofertas laborales publicadas según la propiedad. • Listado de las ofertas laborales publicadas que cumplen las condiciones de conteo para el gráfico de barras.
Implementado (Sí/No)	Si se implementó, Samuel reyes

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Iteración	O(n)
Generación_respuesta	O(m)
TOTAL	***O(n +m)***

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	**Apple M1**
Memoria RAM	8 GB
Sistema Operativo	macOS Sonoma 14.4.1

Entrada	**Tiempo (ms)**
small	1127,87
10 pct	1270,4
30 pct	1728,34
80 pct	3105,37

large	3413,93

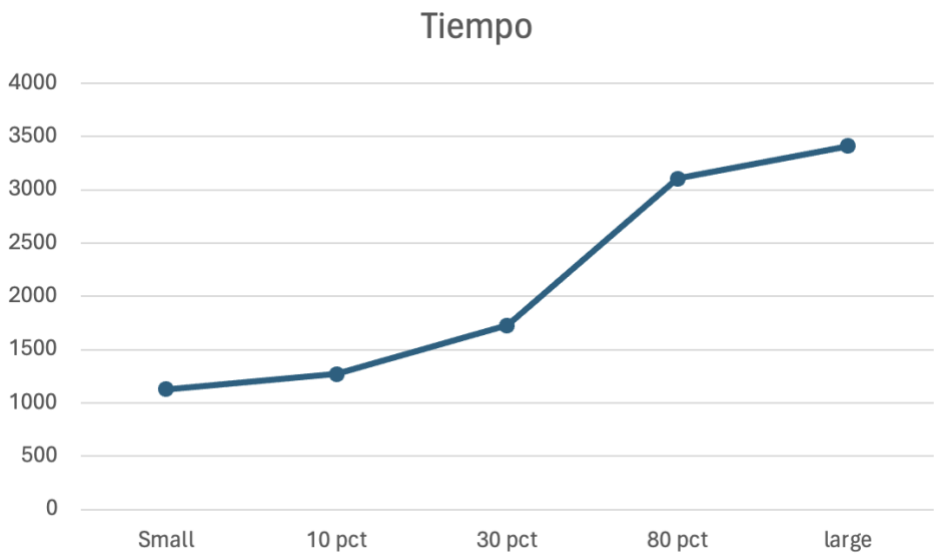
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	1127,87
10 pct	Dato2	1270,4
30 pct	Dato3	1728,34
80 pct	Dato4	3105,37
large	Dato5	3413,93

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

La complejidad de tiempo sigue siendo O(n log n) en este requerimiento. A pesar de que solo se itera sobre un orderedmap (O(n)) y se obtienen valores de él (O(log n + m)), y se agregan elementos a una lista (O(1)), el ordenamiento final con mergesort determina la complejidad total.

Requerimiento <<8>>

Descripción

```python
def req_8(control,respuesta_req,type,req_name):
    """
    Función que soluciona el requerimiento 8
    """
    tqdm.pandas(ascii=True, bar_format='{l_bar}%s{bar}%s{r_bar}' % (Fore.GREEN, Fore.RESET))


    m = folium.Map(location=[0, 0], zoom_start=3)
    marker_cluster = MarkerCluster().add_to(m)

    if type == 1:
        for dato in tqdm(lt.iterator(respuesta_req), colour='green', ascii=True, bar_format='{l_bar}%s{bar}%s{r_bar}' % (Fore.GREEN, Fore.RESET)):
            folium.Marker([dato['latitude'], dato['longitude']], popup=dato,tooltip=dato['title'], icon=folium.Icon(color='lightblue', icon='star', prefix='fa', icon_color='darkred', angle=180, spin=True)).add_to(marker_cluster)
    elif type == 2:
        for city in tqdm(lt.iterator(respuesta_req),colour='red'):
            for ciudad, trabajo in city.items():
                for dato in trabajo:
                    folium.Marker([dato['latitude'], dato['longitude']], popup=dato,tooltip=dato['title'],icon=folium.Icon(color='lightblue', icon='star', prefix='fa', icon_color='darkred', angle=180, spin=True)).add_to(marker_cluster)


    titulo_mapa = f"{req_name}.html"
    m.save(titulo_mapa)

    subprocess.Popen(['start', titulo_mapa], shell=True)
    webbrowser.open_new_tab(titulo_mapa)

def req_8_todos(respuesta_req1,respuesta_req2,respuesta_req3,respuesta_req4,respuesta_req5,respuesta_req6,respuesta_req7):
    tqdm.pandas(ascii=True, bar_format='{l_bar}%s{bar}%s{r_bar}' % (Fore.GREEN, Fore.RESET))

    m = folium.Map(location=[0, 0], zoom_start=3)
    marker_cluster = MarkerCluster().add_to(m)

    for dato in lt.iterator(respuesta_req1):
        folium.Marker([dato['latitude'], dato['longitude']], popup=dato,tooltip=dato['title'], icon=folium.Icon(color='lightblue', icon='star', prefix='fa', icon_color='darkred', angle=180, spin=True)).add_to(marker_cluster)
    for dato in lt.iterator(respuesta_req2):
        folium.Marker([dato['latitude'], dato['longitude']], popup=dato,tooltip=dato['title'], icon=folium.Icon(color='darkpurple', icon='cloud', prefix='fa', icon_color='darkred', angle=180, spin=True)).add_to(marker_cluster)
    for dato in lt.iterator(respuesta_req3):
        folium.Marker([dato['latitude'], dato['longitude']], popup=dato,tooltip=dato['title'], icon=folium.Icon(color='lightgreen', icon='cog', prefix='fa', icon_color='darkred', angle=180, spin=True)).add_to(marker_cluster)
    for dato in lt.iterator(respuesta_req4):
        folium.Marker([dato['latitude'], dato['longitude']], popup=dato,tooltip=dato['title'], icon=folium.Icon(color='cadetblue', icon='diamond', prefix='fa', icon_color='darkred', angle=180, spin=True)).add_to(marker_cluster)
    for dato in lt.iterator(respuesta_req5):
        folium.Marker([dato['latitude'], dato['longitude']], popup=dato,tooltip=dato['title'], icon=folium.Icon(color='pink', icon='heart', prefix='fa', icon_color='darkred', angle=180, spin=True)).add_to(marker_cluster)
    for city in tqdm(lt.iterator(respuesta_req6),colour='red'):
        for ciudad, trabajo in city.items():
            for dato in trabajo:
                folium.Marker([dato['latitude'], dato['longitude']], popup=dato,tooltip=dato['title'],icon=folium.Icon(color='green', icon='key', prefix='fa', icon_color='darkred', angle=180, spin=True)).add_to(marker_cluster)

    for dato in lt.iterator(respuesta_req7):
        folium.Marker([dato['latitude'], dato['longitude']], popup=dato,tooltip=dato['title'], icon=folium.Icon(color='red', icon='fa-life-ring', prefix='fa', icon_color='darkred', angle=180, spin=True)).add_to(marker_cluster)

    titulo_mapa = f"todos_los_requerimientos.html"
    m.save(titulo_mapa)

    subprocess.Popen(['start', titulo_mapa], shell=True)
    webbrowser.open_new_tab(titulo_mapa)
```

visualizar gráficamente en un mapa interactivo TODOS los requerimientos previamente implementados.

Entrada	No hay un input especifico
Salidas	Grafica de todos los requerimientos anteriores
Implementado (Sí/No)	Si se implementó, Zair Montoya

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
TOTAL (Suma de la complejidad de todos los req)	***O(req_1 + req_2 + ….. + req_8)***

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	**Apple M1**
Memoria RAM	8 GB
Sistema Operativo	macOS Sonoma 14.4.1

Entrada	Tiempo (ms)
small	69240,09
10 pct	89484,36
30 pct	133125,49
80 pct	280105,41
large	510826,05

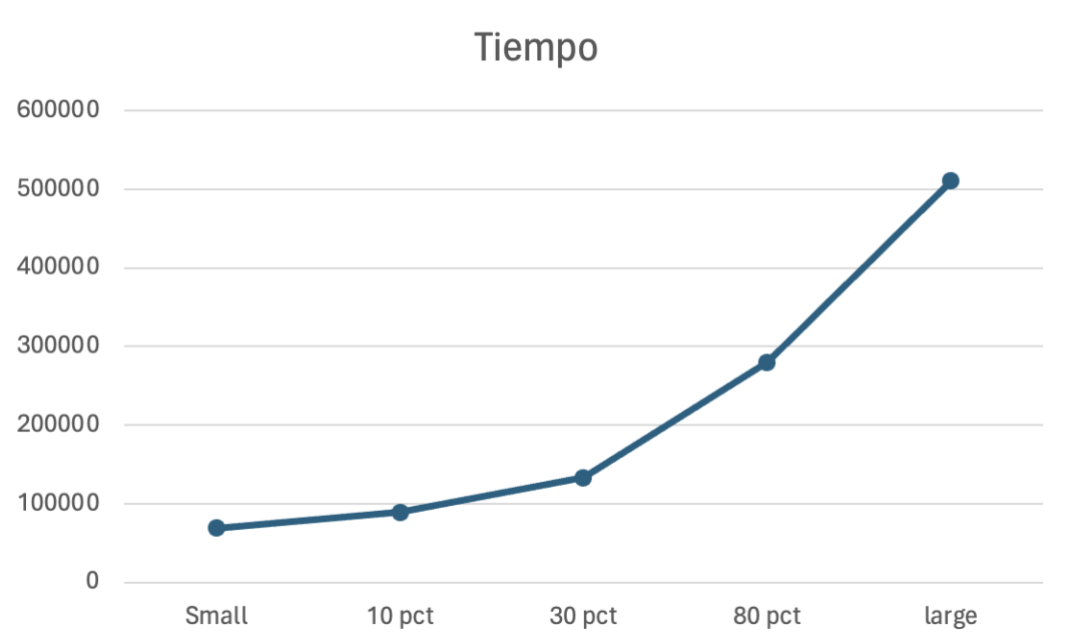
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	69240,09
10 pct	Dato2	89484,36
30 pct	Dato3	133125,49
80 pct	Dato4	280105,41
large	Dato5	510826,05

Graficas

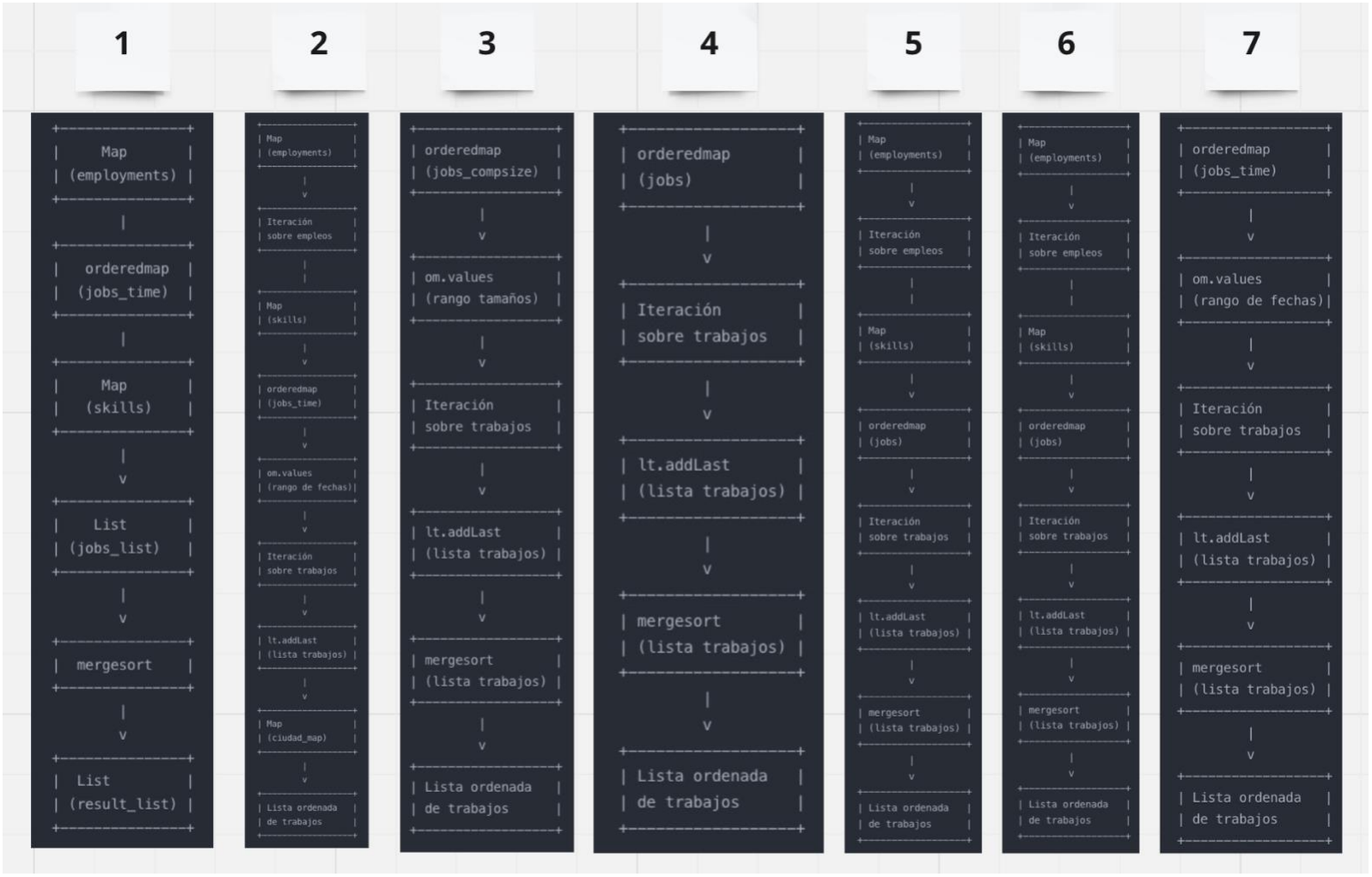
Las gráficas con la representación de las pruebas realizadas.



Análisis

Este requerimiento es el único con una complejidad de tiempo lineal distinta, ya que al utilizar el resto de requerimientos para su desarrollo se llega a la conclusión que la complejidad de este se convierte en la sumatoria de la complejidad de los otros 7 requerimientos.

Diagramas desarrollo de los requerimientos:



Requerimiento Ejemplo

Descripción

```python
def get_data(data_structs, id):
    """
    Retorna un dato a partir de su ID
    """
    pos_data = lt.isPresent(data_structs["data"], id)
    if pos_data > 0:
        data = lt.getElement(data_structs["data"], pos_data)
        return data
    return None
```

Este requerimiento se encarga de retornar un dato de una lista dado su ID. Lo primero que hace es verificar si el elemento existe. Dado el caso que exista, retorna su posición, lo busca en la lista y lo retorna. De lo contrario, retorna None.

Entrada	Estructuras de datos del modelo, ID.
Salidas	El elemento con el ID dado, si no existe se retorna None
Implementado (Sí/No)	Si. Implementado por Juan Andrés Ariza

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Buscar si el elemento existe (isPresent)	O(n)
Obtener el elemento (getElement)	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Los datos de entrada fueron el ID 1.

Procesadores	**AMD Ryzen 7 4800HS with Radeon Graphics**
Memoria RAM	8 GB
Sistema Operativo	Windows 10

Entrada	**Tiempo (ms)**
small	0.05
5 pct	0.33
10 pct	1.28

20 pct	2.54
30 pct	4.98
50 pct	7.51
80 pct	13.81
large	25.97

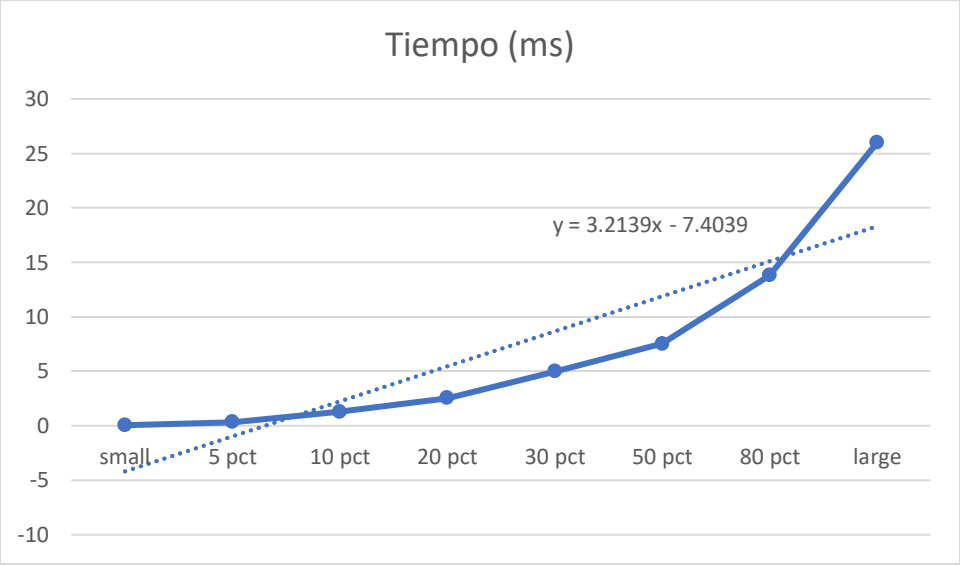
Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

Muestra	Salida	Tiempo (ms)
small	Dato1	0.05
5 pct	Dato2	0.33
10 pct	Dato3	1.28
20 pct	Dato4	2.54
30 pct	Dato5	4.98
50 pct	Dato6	7.51
80 pct	Dato7	13.81
large	Dato8	25.97

Graficas

Las gráficas con la representación de las pruebas realizadas.



Análisis

A pesar de que obtener un elemento en un *ArrayList,* dada su posición, tiene complejidad constante, la implementación de este requerimiento tiene un orden lineal O(n). Esto debido a que, lo primero que se

hace es verificar si el elemento hace parte de la lista. Específicamente, a la hora de buscar un elemento en una lista, en el peor de los casos es necesario recorrer toda la lista, es decir, complejidad lineal.

Este comportamiento se puede evidenciar experimentalmente en la gráfica. Ya que, gracias a que los datos no se encuentran tan dispersos con respecto a la línea de tendencia, la curva coincide con el comportamiento lineal esperado.