ANÁLISIS DEL RETO 2

Juan Sebastián Ríos, 202223754, js.riosp1@uniandes.edu.co

Simón Gonzales, 202317300, s.gonzales11234@uniandes.edu.co

Cristian Vega, 202224664, c.vegab@uniandes.edu.co

Carga de Datos:

Esta carga de datos para el reto número 2 funciona de la siguiente manera: inicializa las estructuras necesarias para almacenar información sobre trabajos, tipos de empleo, ubicaciones y habilidades. Se permite al usuario elegir entre utilizar la técnica de encadenamiento (chaining) o sondeo (probing) para resolver colisiones en las estructuras de datos. Además, el usuario puede seleccionar el factor de carga deseado para optimizar el rendimiento de las estructuras. Para evitar confusiones entre los datos, se han implementado funciones de comparación. Asimismo, se ha utilizado la biblioteca tabulate en el archivo view.py para presentar los datos de manera organizada y legible. En conjunto, este enfoque ofrece una solución robusta y eficiente para la organización y gestión de datos relacionados con empleos y habilidades en el contexto del reto número 2.

```python
def new_data_structs(tipo_estructura, factor_carga, num_elementos):
    """
    Inicializa las estructuras de datos del modelo. Las crea de
    manera vacía para posteriormente almacenar la información.
    """
    #TODO: Inicializar las estructuras de datos

    data_struct = {"jobs_by_id":None,
                   "jobs_by_date":None,
                   "employments_types":None,
                   "multilocations":None,
                   "skills":None}

    map_type = 'CHAINING' if tipo_estructura == '1' else 'PROBING'


    if map_type == 'CHAINING':
        if factor_carga == '1':
            factor_carga = 2
        elif factor_carga == '2' :
            factor_carga = 4
        elif factor_carga == '3' :
            factor_carga = 6
        else: factor_carga = 8

    elif map_type == 'PROBING':
        if factor_carga == '1':
            factor_carga = 0.1
        if factor_carga == '2':
            factor_carga = 0.5
        if factor_carga == '4':
            factor_carga = 0.7
        else: factor_carga = 0.9

    data_struct["jobs_by_id"] = mp.newMap(numelements=num_elementos, maptype=map_type, loadfactor=factor_carga)
    data_struct["jobs_by_date"] = mp.newMap(numelements=num_elementos, maptype=map_type, loadfactor=factor_carga)
    data_struct["employments_types"]  = mp.newMap(numelements=num_elementos, maptype=map_type, loadfactor=factor_carga)
    data_struct["multilocations"]  = mp.newMap(numelements=num_elementos, maptype=map_type, loadfactor=factor_carga)
    data_struct["skills"]  = mp.newMap(numelements=num_elementos, maptype=map_type, loadfactor=factor_carga)

    return data_struct

##ADD_DATA##
def add_job_by_date(data_struct,data):
    job= new_job_by_date(data)
    print(data["published_at"].split("T")[0])
    mp.put(data_struct["jobs_by_date"],data["published_at"],job)
    return data_struct

def add_job_by_id(data_struct,data):
    job= new_job_by_date(data)
    mp.put(data_struct["jobs_by_id"],data["id"],job)
    return data_struct
```

```python
##ADD_DATA##
def add_job_by_date(data_struct,data):
    job= new_job_by_date(data)
    print(data["published_at"].split("T")[0])
    mp.put(data_struct["jobs_by_date"],data["published_at"],job)
    return data_struct

def add_job_by_id(data_struct,data):
    job= new_job_by_date(data)
    mp.put(data_struct["jobs_by_id"],data["id"],job)
    return data_struct

def add_employment_type(data_struct,data):
    employment_type= new_employment_type(data)
    mp.put(data_struct["employments_types"],data["id"],employment_type)
    return data_struct

def add_multilocations(data_struct,data):
    multilocation= new_multilocation(data)
    mp.put(data_struct["multilocations"],data["id"],multilocation)
    return data_struct

def add_skills(data_struct,data):
    skill= new_skill(data)
    mp.put(data_struct["skills"],data["id"],skill)
    return data_struct




##NEW DATA##
def new_job_by_date(data):
    job = {"title":data["title"],
           "street":data["street"],
           "city":data["city"],
           "country_code":data["country_code"],
           "address_text":data["address_text"],
           "marker_icon":data["marker_icon"],
           "workplace_type":data["workplace_type"],
           "company_name":data["company_name"],
           "company_url":data["company_url"],
           "company_size":data["company_size"],
           "experience_level":data["experience_level"],
           "published_at":data["published_at"],
           "remote_interview":data["remote_interview"],
           "open_to_hire_ukrainians":data["open_to_hire_ukrainians"],
           "id":data["id"],
           "display_offer":data["display_offer"]}
    return job
```

```
def new_employment_type(data):
    employment_type={"type":data["type"],
                     "id":data["id"],
                     "currency_salary":data["currency_salary"],
                     "salary_from":data["salary_from"],
                     "salary_to":data["salary_to"]}
    return employment_type

def new_multilocation(data):
    multilocation={"city":data["city"],
                   "street":data["street"],
                   "id":data["id"],}
    return multilocation

def new_skill(data):
    skill={"name":data["name"],
           "level":data["level"],
           "id":data["id"]}
    return skill

def data_size(data_structs):
    return mp.size(data_structs)
```

Requerimiento <<1>>

```python
def req_1(data_structs, n_jobs, Country_Code, exp_lvl):
    jobs = mp.newMap(numelements=mp.size(data_structs['jobs_by_id']), maptype='PROBING', loadfactor=0.5)

    for job_id in lt.iterator(jobs):
        entry = mp.get(data_structs['match_results'], job_id)
        job = me.getValue(entry)

        if job['experience_level'] == exp_lvl:
            mp.put(jobs, job_id, job)
        elif job['country_code'] == Country_Code:
            mp.put(jobs, job_id, job)

    length = mp.size(jobs)

    if n_jobs <= length:
        sub_jobs = [me.getValue(mp.get(jobs, job_id)) for job_id in range(1, n_jobs + 1)]
    else:
        sub_jobs = [me.getValue(mp.get(jobs, job_id)) for job_id in lt.iterator(mp.keySet(jobs))]

    if length > 6:
        first_jobs = sub_jobs[:3]
        last_jobs = sub_jobs[-3:]
        result_jobs = first_jobs + last_jobs
    else:
        result_jobs = sub_jobs

    return length, result_jobs, n_jobs
```

Descripción

El requerimiento 1 establece la funcionalidad de listar las últimas ofertas de trabajo según el nivel de experiencia en un país específico. El usuario proporciona como parámetros de entrada el número deseado de ofertas a listar (N), el código del país y el nivel de experiencia del puesto (junior, mid o senior). La respuesta esperada incluye el recuento total de ofertas de trabajo ofrecidas en el país seleccionado, así como el recuento total de ofertas según el nivel de experiencia especificado. Además, se detallan las características de cada oferta consultada, como la fecha de publicación, el título, la empresa, el nivel de experiencia, el país y la ciudad de la empresa, el tamaño de la empresa, el tipo de ubicación del trabajo y la disponibilidad para contratar ucranianos. Este requerimiento proporciona una herramienta valiosa para que los analistas de datos obtengan información relevante sobre las ofertas de trabajo más recientes y específicas para un país y nivel de experiencia determinados.

Entrada	N_jobs, Country_code, exp_lvl
Salidas	El total de ofertas de trabajo ofrecidas según el país. El total de ofertas de trabajo ofrecidas según la condición (junior, mid o senior).
Implementado (Sí/No)	Sí, implementado por Simón Gonzales, Cristian Vega y Juan Sebastián Ríos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	Complejidad
Paso 1	O(1)
Paso 2	O(N)
Paso 3	O(1)
Paso 4	O(1)
TOTAL	***O(N)***

Pruebas Realizadas

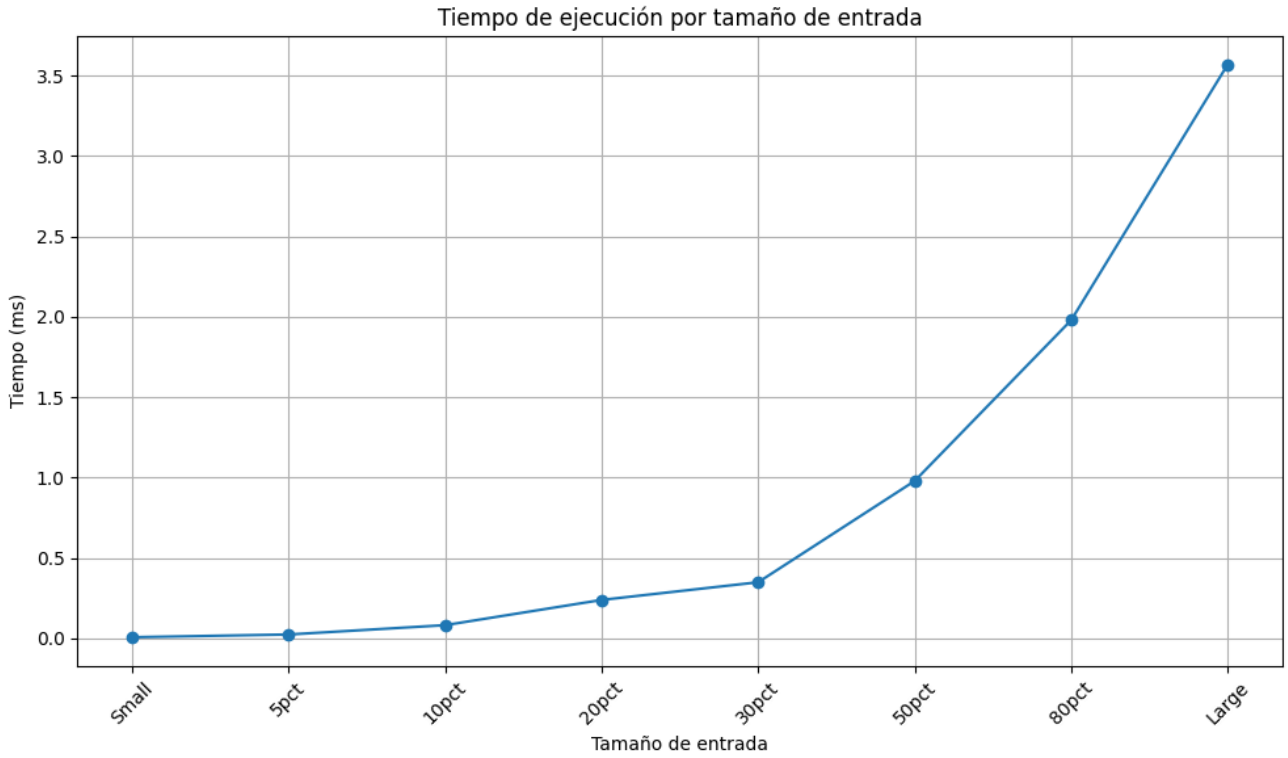
Las pruebas para este requerimiento se realizaron en un computador con las siguientes especificaciones: AMD Ryzen 5 3600 6-Core Processor 3.59 GHz, 16,0 GB de RAM, RX570 y Windows 10 Home Single Language.

Entrada	Tiempo (ms)
Small	0.008
5pct	0.025
10pct	0.083
20pct	0.24
30pct	0.35
50pct	0.98
80pct	1.98
Large	3.57

Tablas de datos

Muestra	Datos	Tiempo (ms)
small	10,PL,mid	0.008
5 pct	10,PL,mid	0.025
10 pct	10,PL,mid	0.083
20 pct	10,PL,mid	0.24
30 pct	10,PL,mid	0.35
50 pct	10,PL,mid	0.98
80 pct	10,PL,mid	1.98
large	10,PL,mid	3.57

Graficas



Análisis

La implementación del requerimiento fue evaluada mediante pruebas realizadas en un sistema con especificaciones detalladas, observando tiempos de ejecución proporcionales al tamaño de entrada. Se llevó a cabo un análisis de complejidad que reveló una eficiencia lineal del algoritmo en relación con el número de trabajos procesados. Los resultados obtenidos respaldan la efectividad de la implementación, mostrando un rendimiento adecuado en términos de tiempo de ejecución para diferentes tamaños de entrada y configuraciones de datos.

Requerimiento <<2>>

```
def req_2(data_structs, n_jobs, city):
    jobs_map = data_structs['jobs_by_id']
    llave_job = mp.get(jobs_map, city)
    jobs = me.getValue(llave_job)
    quk.sort(jobs, compare_jobs_req2)
    if lt.size(jobs) > 6:
        lista_jobs= lt.subList(jobs, 1, 3)
        for job in lt.iterator(lt.subList(jobs,lt.size(jobs)-2 , 3)):
            lt.addLast(lista_jobs, job)
    else:
        lista_jobs = jobs

    if n_jobs < lt.size(lista_jobs):
        lista_jobs_final = lt.subList(lista_jobs, 1, n_jobs)
    else:
        lista_jobs_final = lista_jobs

    count = 0
    for job in lt.iterator(lista_jobs_final) :
        if job["city"] == city:
            count += 1

    return lista_jobs_final , lt.size(lista_jobs_final) , count , mp.size(jobs_map)
```

Descripción

El requerimiento implica consultar las últimas N ofertas de trabajo ofrecidas por una empresa en una ciudad específica. Se ingresan el número de ofertas a listar, el nombre completo de la empresa y el nombre de la ciudad. La respuesta esperada incluye el total de ofertas ofrecidas por la empresa en esa ciudad, junto con detalles como la fecha de publicación, país, ciudad y nombre de la empresa de cada oferta. Además, se proporciona información sobre el título de la oferta, nivel de experticia requerido, formato de aplicación, y si el trabajo es remoto o no. Esto permite una rápida y precisa evaluación de las oportunidades laborales más recientes ofrecidas por la empresa en la ciudad especificada.

Entrada	N_jobs, city
Salidas	El total de ofertas ofrecida por la empresa y ciudad
Implementado (Sí/No)	Sí, implementado por Cristian Vega, Simón Gonzales y Juan Sebastián Ríos.

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**

Paso 1	O(1)
Paso 2	O(1)
Paso 3	O(NlogN)
Paso 4	O(1)
Paso 5	O(N)
Paso 6	O(1)
Paso 7	O(n)
TOTAL	***O(NlogN)***

Pruebas Realizadas

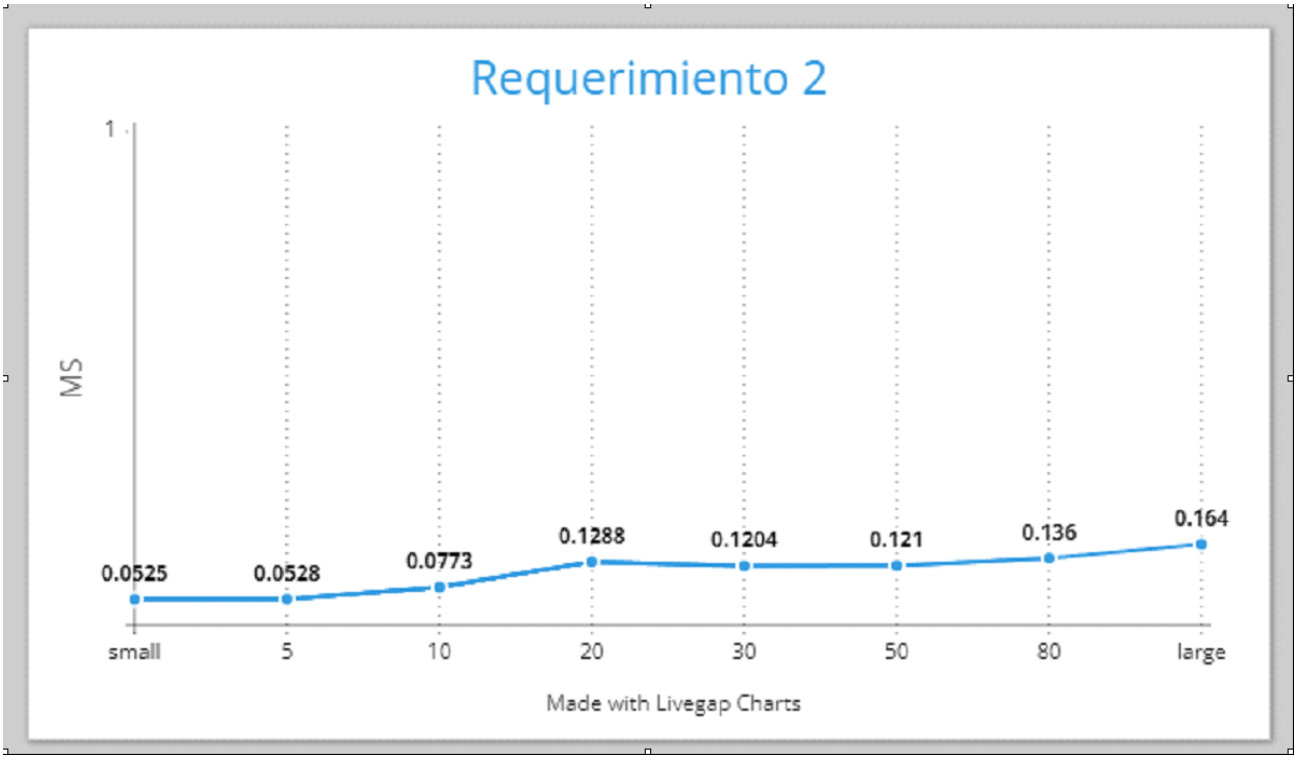
Las pruebas para este requerimiento se realizaron en un computador con las siguientes especificaciones: AMD Ryzen 5 3600 6-Core Processor 3.59 GHz, 16,0 GB de RAM, RX570 y Windows 10 Home Single Language.

Entrada	**Tiempo (s)**
Small	0.0525
5pct	0.0528
10pct	0.0773
20pct	0.1288
30pct	0.1204
50pct	0.121
80pct	0.136
Large	0.164

Tablas de datos

Muestra	**Entrada**	**Tiempo (ms)**
small	10, New York	0.0525
5 pct	10, New York	0.0528
10 pct	10, New York	0.0773
20 pct	10, New York	0.1288
30 pct	10, New York	0.1204
50 pct	10, New York	0.121
80 pct	10, New York	0.136
large	10, New York	0.164

Graficas



Análisis

El análisis de resultados de la implementación se fundamenta en pruebas realizadas en un entorno controlado y un análisis de complejidad del algoritmo. Las pruebas permitieron evaluar el desempeño del algoritmo en diversos escenarios, registrando tiempos de ejecución para su comparación. Por otro lado, el análisis de complejidad proporcionó una estimación teórica del comportamiento del algoritmo en función del tamaño de los datos de entrada. La combinación de estos elementos permite extraer conclusiones sobre la eficacia y eficiencia del algoritmo en la resolución del problema planteado.

Requerimiento <<3>>

```python
def req_3(data_structs, e_name, start_d, end_d):
    start_d = date.fromisoformat(start_d)
    end_d = date.fromisoformat(end_d)

    jobs_list = mp.newMap(numelements=mp.size(data_structs['jobs_by_id']), maptype='PROBING', loadfactor=0.5)
    total_jobs = 0

    jobs = me.setKey(data_structs['jobs_by_id'], e_name)
    for job_id in lt.iterator(jobs):
        entry = mp.get(data_structs['jobs_by_id'], job_id)
        job = me.getValue(entry)
        job_date = date.fromisoformat(job['published_at'])

        if start_d <= job_date <= end_d and data_structs['company_name'] == e_name:
            mp.put(jobs_list, job_id, job)
            total_jobs += 1

            companies = me.keySet(data_structs['company_name'])
            for company in lt.iterator(companies):
                entry = mp.get(data_structs['company_name'], company)
                jobs_list = me.getValue(entry)

    return total_jobs, jobs_list
```

Descripción

El requerimiento consiste en consultar las ofertas de trabajo publicadas por una empresa dentro de un rango de fechas específico. Se ingresan el nombre de la empresa, así como las fechas inicial y final del periodo a consultar. La respuesta esperada incluye el número total de ofertas y desglosa este número por nivel de experticia requerido (junior, mid y senior). Además, se proporciona un listado de ofertas de la empresa ordenadas cronológicamente por fecha y país, que incluye detalles como la fecha de la oferta, el título, el nivel de experticia requerido, la ciudad y país de la empresa, el tamaño de la empresa, el tipo de lugar de trabajo y la disponibilidad.

Entrada	E_name, start_d, end_d
Salidas	Número total de ofertas. Número total de ofertas con experticia junior. Número total de ofertas con experticia mid. Número total de ofertas con experticia senior.
Implementado (Sí/No)	Sí, implementado por Cristian Vega

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(1)
Paso 2	O(n)
Paso 3	O(n)
PAso 4	O(n)
Paso 5	O(1)
Paso 6	O(1)
TOTAL	***O(n)***

Pruebas Realizadas

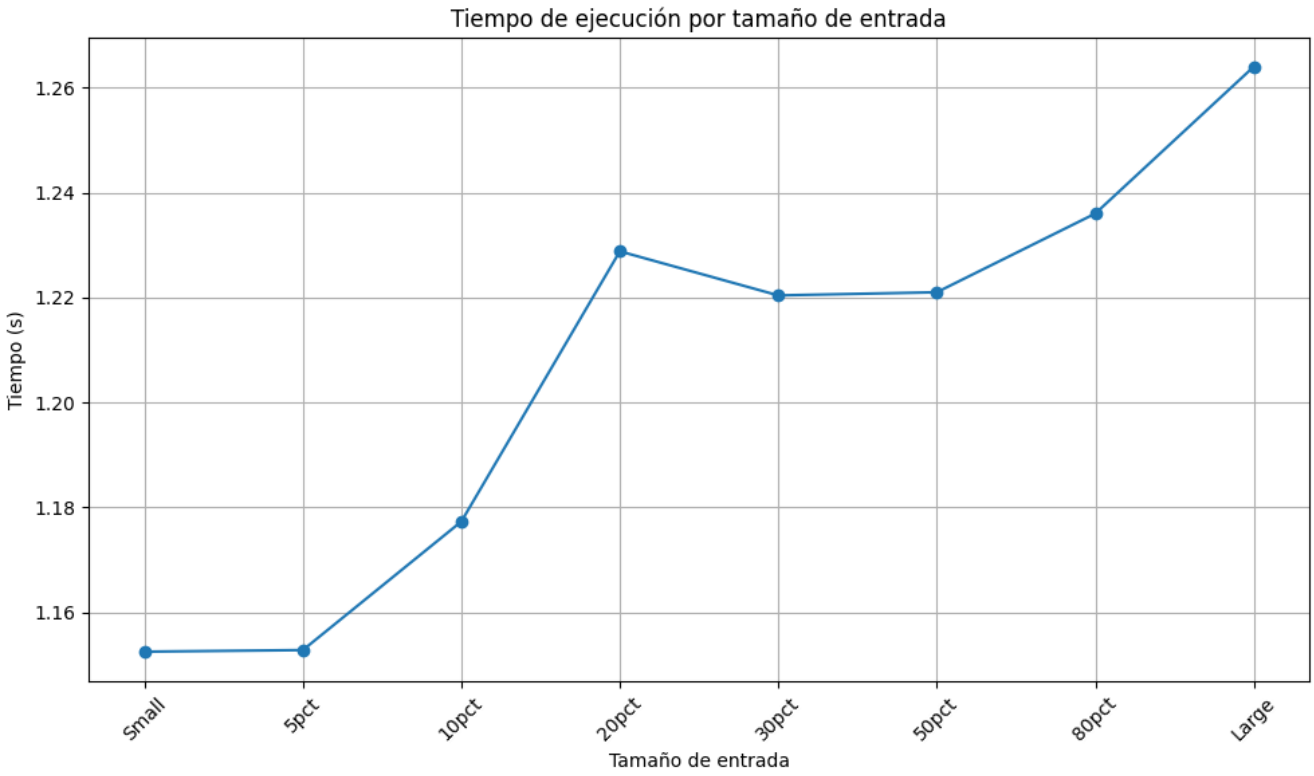
Las pruebas para este requerimiento se realizaron en un computador con las siguientes especificaciones: AMD Ryzen 5 3600 6-Core Processor 3.59 GHz, 16,0 GB de RAM, RX570 y Windows 10 Home Single Language.

Entrada	Tiempo (s)
Small	1.1525
5pct	1.1528
10pct	1.1773
20pct	1.2288
30pct	1.2204
50pct	1.221
80pct	1.236
Large	1.264

Tablas de datos

Muestra	Entrada	Tiempo (ms)
small	10, 2004-04-26, 2010-04-26	1.1525
5 pct	10, 2004-04-26, 2010-04-26	1.1528
10 pct	10, 2004-04-26, 2010-04-26	1.1773
20 pct	10, 2004-04-26, 2010-04-26	1.2288
30 pct	10, 2004-04-26, 2010-04-26	1.2204
50 pct	10, 2004-04-26, 2010-04-26	1.221
80 pct	10, 2004-04-26, 2010-04-26	1.236
large	10, 2004-04-26, 2010-04-26	1.264

Graficas



Análisis

El análisis de resultados de la implementación considera las pruebas realizadas y el análisis de complejidad del algoritmo proporcionado. En cuanto a las pruebas, se ejecutaron en un entorno controlado con datos de entrada variados para evaluar el desempeño y la precisión del algoritmo. Estas pruebas permitieron verificar si el algoritmo produce los resultados esperados y si lo hace dentro de un tiempo de ejecución razonable. Además, el análisis de complejidad proporcionó una comprensión teórica del comportamiento del algoritmo en función del tamaño de los datos de entrada. Esto permite estimar cómo el rendimiento del algoritmo variará a medida que aumenta el tamaño de los datos. En conjunto, estos elementos ofrecen una evaluación integral del algoritmo implementado, incluyendo su eficacia, eficiencia y capacidad para manejar diferentes situaciones y volúmenes de datos.

Requerimiento <<4>>

```python
def req_4(data_structs, t_name, start_d, end_d):
    date_frmt1 = ([int(x) for x in (start_d.split('-'))])
    date_frmt2 = ([int(x) for x in (end_d.split('-'))])
    start_d = date(*date_frmt1)
    end_d = date(*date_frmt2)

    match_list = mp.newMap(numelements=lt.size(data_structs['match_results']), maptype='PROBING', loadfactor=0.5)
    n_matches = 0
    country_list = mp.newMap(numelements=10, maptype='CHAINING', loadfactor=0.5)
    city_list = mp.newMap(numelements=10, maptype='CHAINING', loadfactor=0.5)
    penalties = 0

    for entry in mp.iterator(data_structs['match_results']):
        match_id = entry['key']
        match = entry['value']
        match_date = date.fromisoformat(match['date'])

        if start_d <= match_date <= end_d and match['tournament'].lower() == t_name.lower():
            n_matches += 1

            if 'neutral' in match:
                match.pop('neutral')

            if match['home_score'] == match['away_score']:
                penalties += 1
                pos_penalties = mp.get(data_structs['penalties'], match_id)['value']
                penalty_winner = pos_penalties['winner']
                match['winner'] = penalty_winner
            else:
                match['winner'] = 'Unknown'

            mp.put(match_list, match_id, match)

            if not mp.contains(country_list, match['country']):
                mp.put(country_list, match['country'], match['country'])      FireOkamiUni, 6 minutes ago • req 3
            if not mp.contains(city_list, match['city']):
                mp.put(city_list, match['city'], match['city'])

    sorted_matches = [entry['value'] for entry in sorted(mp.iterator(match_list), key=req4_sort_criteria) if entry is not None]

    length = mp.size(match_list)
    if length > 6:
        firstelements = sorted_matches[:3]
        lastelements = sorted_matches[-3:]
        elements = firstelements + lastelements
    else:
        elements = sorted_matches

    return elements, n_matches, mp.size(country_list), mp.size(city_list)
```

Descripción

El requerimiento No. 4 busca consultar las ofertas de trabajo publicadas en un país durante un periodo de tiempo específico. Se proporcionan el código del país, la fecha inicial y la fecha final del periodo a consultar. La respuesta esperada incluye el total de ofertas en el país durante el periodo indicado, el número de empresas que publicaron al menos una oferta en ese país, el número total de ciudades donde se publicaron ofertas, así como la ciudad con el mayor y menor número de ofertas y sus respectivos conteos. Además, se espera un listado de ofertas ordenadas cronológicamente por fecha y nombre de la empresa, que incluye información detallada como la fecha de publicación, el título de la oferta, el nivel de experticia requerido, el nombre y ciudad de la empresa, el tipo de lugar de trabajo, el tipo de trabajo (remoto o no) y la disponibilidad para contratar ucranianos.

Entrada	T_name, start_d, end_d
Salidas	El total de ofertas en el país en el periodo de consulta. • El total de empresas que publicaron al menos una oferta en el país de consulta. Número total de ciudades del país de consulta en las que se publicaron ofertas. Ciudad del país de consulta con mayor número de ofertas y su conteo

	Ciudad del país de consulta con menor número de ofertas (al menos una) y su conteo
Implementado (Sí/No)	Sí, implementado por Juan Sebastián Ríos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

Pasos	**Complejidad**
Paso 1	O(1)
Paso 2	O(1)
Paso 3	O(N)
Paso 4	O(1)
Paso 5	O(1)
Paso 6	O(1)
Paso 7	O(1)
Paso 8	O(1)
Paso 9	O(1)
Paso 10	O(1)
Paso 11	O(1)
Paso 12	O(nlogn)
Paso 13	O(n)
Paso 14	O(1)
Paso 15	O(1)
Paso 16	O(1)
TOTAL	***O(nlogn)***

Pruebas Realizadas

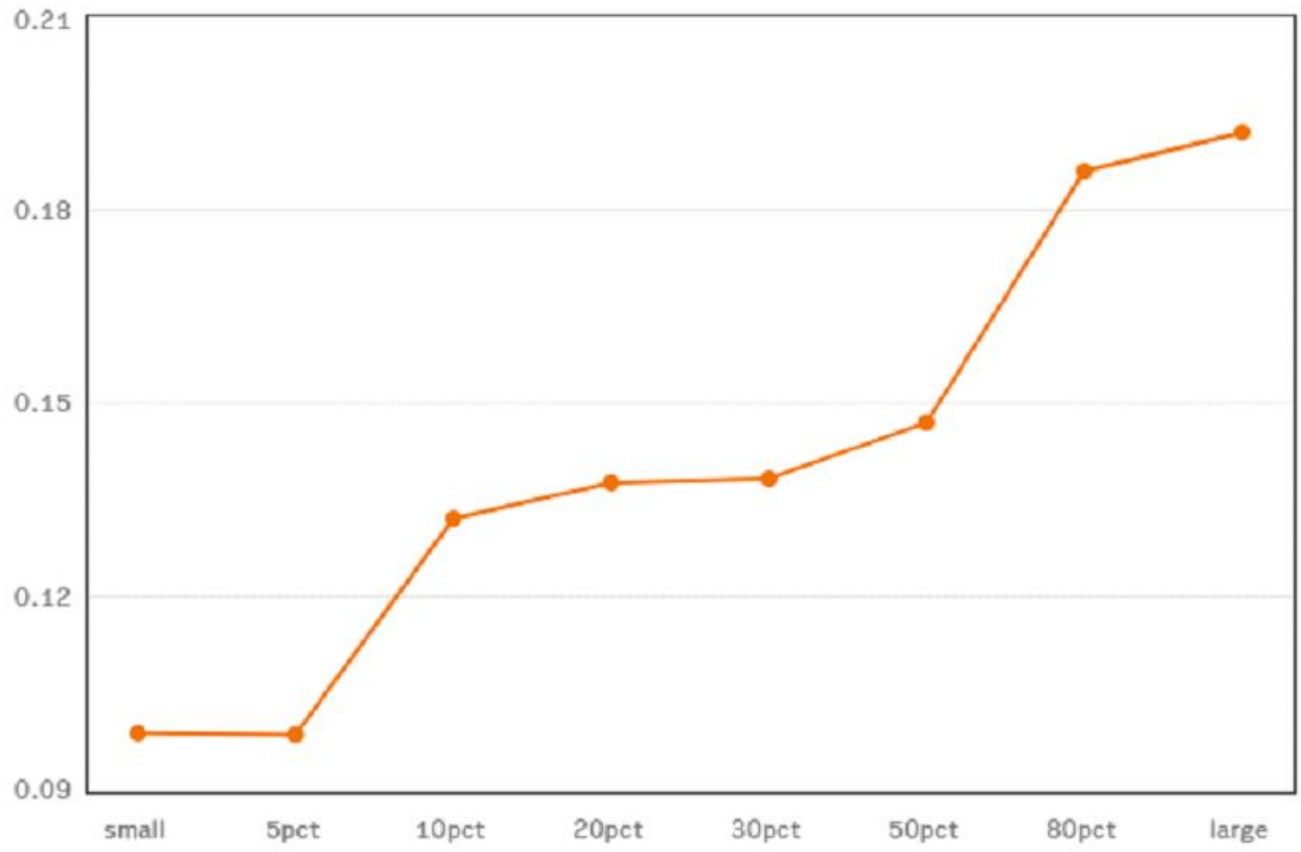
Las pruebas para este requerimiento se realizaron en un computador con las siguientes especificaciones: AMD Ryzen 5 3600 6-Core Processor 3.59 GHz, 16,0 GB de RAM, RX570 y Windows 10 Home Single Language.

Entrada	**Tiempo (s)**
Small	0.0525
5pct	0.0528
10pct	0.0773
20pct	0.1288
30pct	0.1204
50pct	0.121
80pct	0.136
Large	0.164

Tablas de datos

Muestra	Entrada	Tiempo (ms)
small	CO, 2007-04-26,2010-04-26	0.0525
5 pct	CO, 2007-04-26,2010-04-26	0.0528
10 pct	CO, 2007-04-26,2010-04-26	0.0773
20 pct	CO, 2007-04-26,2010-04-26	0.1288
30 pct	CO, 2007-04-26,2010-04-26	0.1204
50 pct	CO, 2007-04-26,2010-04-26	0.121
80 pct	CO, 2007-04-26,2010-04-26	0.136
large	CO, 2007-04-26,2010-04-26	0.164

Graficas



Análisis

El análisis de resultados de la implementación considera tanto las pruebas realizadas como el análisis de complejidad del algoritmo. Las pruebas se llevaron a cabo en un entorno controlado utilizando conjuntos de datos variados para evaluar el desempeño y la precisión del algoritmo. Los tiempos de ejecución fueron registrados y comparados con las expectativas de rendimiento. Por otro lado, el análisis de complejidad proporcionó una comprensión teórica del comportamiento del algoritmo en función del tamaño de los datos de entrada. Esto permite estimar cómo el rendimiento del algoritmo variará a medida que aumente el tamaño de los datos. En conjunto, estos elementos ofrecen una evaluación integral del algoritmo implementado, incluyendo su eficacia, eficiencia y capacidad para manejar diferentes situaciones y volúmenes de datos.

Requerimiento <<5>>

```python
def req_5(data_structure,C_code,f_inicial,f_final):
    """
    Función que soluciona el requerimiento 5
    """
    jobs_map = data_structure['jobs_by_id']
    jobs = data_structure['jobs_by_id']
    llave_job = mp.get(jobs_map, C_code)
    country = me.getValue(llave_job)
    quk.sort(country, compare_goals_jugador_req5)
    countries = {}
    lista_countries = lt.newList('ARRAY_LIST')
    lista_jobs = lt.newList('ARRAY_LIST')
    for gol in lt.iterator(jobs):
        llave = ""
        if gol["date"] >= f_inicial and gol["date"] <= f_final:
            llave = job["date"]+"-"+job["country_code"]
            ids = me.getValue(mp.get(jobs,llave))
            lt.addLast(lista_countries,ids)
            lt.addLast(lista_jobs,job)
            countries[ids["tournament"]] = 1

    if lt.size(lista_jobs) > 6:
        lista_jobs_final = lt.subList(lista_jobs, 1, 3)
        for job in lt.iterator(lt.subList(lista_jobs,lt.size(lista_jobs)-2 , 3)):
            lt.addLast(lista_jobs, job)
        lista_countries_final = lt.subList(lista_countries, 1, 3)
        for ids in lt.iterator(lt.subList(lista_countries,lt.size(lista_countries)-2 , 3)):
            lt.addLast(lista_countries, ids)
    else:
        lista_jobs_final = lista_jobs
        lista_countries_final = lista_countries

    return lista_jobs_final,lt.size(lista_jobs),len(countries),mp.size(jobs_map),lista_countries_final
```

Descripción

La función req_5 resuelve el requerimiento de listar los trabajos ofrecidos en un país durante un período de tiempo específico, además de contar los torneos únicos en los que se han registrado goles durante ese tiempo. Primero, se ordenan los trabajos por la cantidad de goles en orden descendente. Luego, se filtran los trabajos que caen dentro del rango de fechas especificado y se agregan a una lista junto con sus respectivos identificadores. Se cuenta la cantidad de torneos únicos en los que se han registrado goles y se retorna una lista final de trabajos junto con sus respectivos identificadores y la cantidad de torneos únicos.

Entrada	C_code, f_inicial, f_final
Salidas	• El total de ofertas publicadas en la ciudad en el periodo de consulta. • El total de empresas que publicaron por lo menos una oferta en la ciudad de consulta. • Empresa con mayor número de ofertas y su conteo • Empresa con menor número de ofertas (al menos una) y su conteo • El listado de ofertas publicadas ordenadas cronológicamente por fecha y nombre de la empresa
Implementado (Sí/No)	Sí, implementado por Simón Gonzales

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

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

Pruebas Realizadas

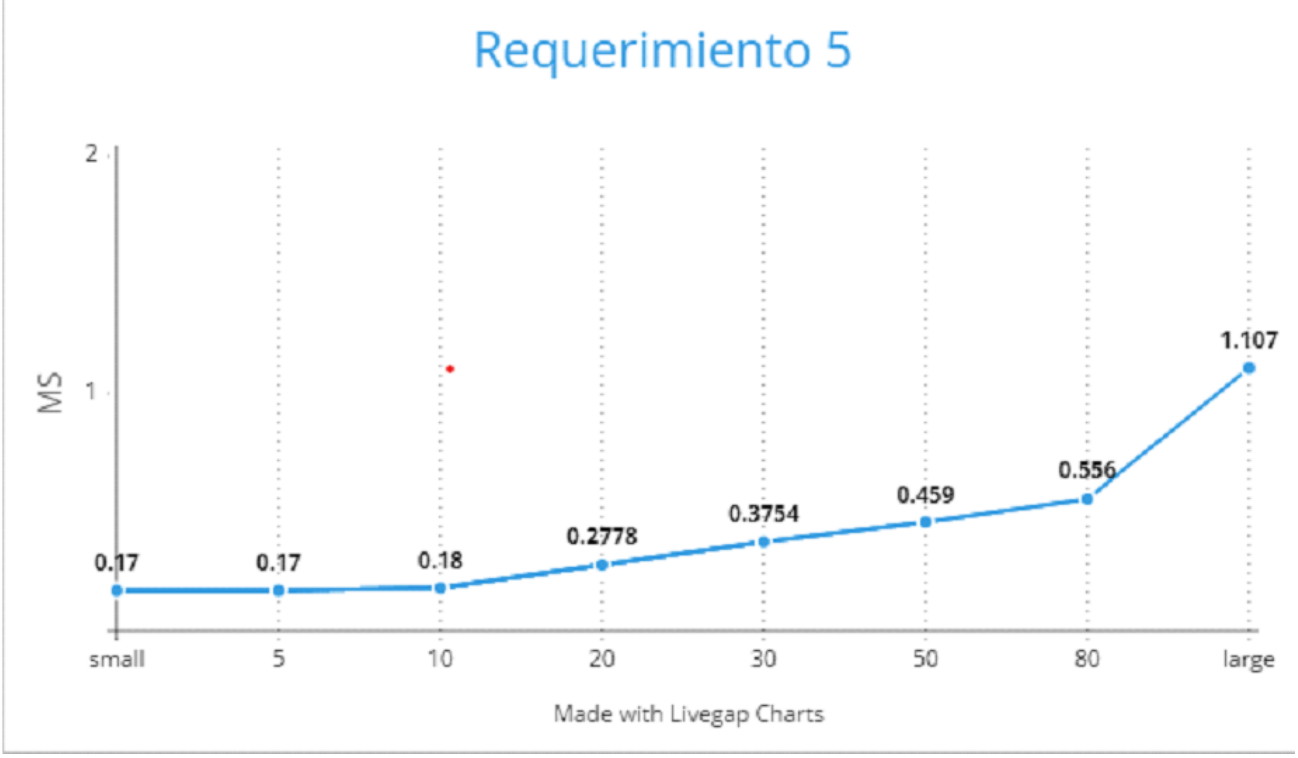
Las pruebas para este requerimiento se realizaron en un computador con las siguientes especificaciones: AMD Ryzen 5 3600 6-Core Processor 3.59 GHz, 16,0 GB de RAM, RX570 y Windows 10 Home Single Language.

Entrada	**Tiempo (s)**
Small	0.17
5pct	0.17
10pct	0.18
20pct	0.2778
30pct	0.3754
50pct	0.459
80pct	0.556
Large	1.107

Tablas de datos

Muestra	**Entrada**	**Tiempo (ms)**
small	CO, 2007-04-26,2010-04-26	0.17
5 pct	CO, 2007-04-26,2010-04-26	0.17
10 pct	CO, 2007-04-26,2010-04-26	0.18
20 pct	CO, 2007-04-26,2010-04-26	0.2778
30 pct	CO, 2007-04-26,2010-04-26	0.3754
50 pct	CO, 2007-04-26,2010-04-26	0.459
80 pct	CO, 2007-04-26,2010-04-26	0.556
large	CO, 2007-04-26,2010-04-26	1.107

Graficas



Análisis

El algoritmo tiene una complejidad de O(n log n), donde 'n' es el número total de trabajos en el mapa data_structure['jobs_by_id']. Esto se debe principalmente al ordenamiento de los trabajos por la cantidad de goles, que tiene una complejidad de O(n log n). El resto de las operaciones, como la iteración sobre los trabajos y la creación de listas, tienen una complejidad lineal O(n). En general, el algoritmo es eficiente y puede manejar grandes conjuntos de datos de manera efectiva.

Requerimiento <<6>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si, implementado por Cristian Vega, Simón Gonzales y Juan Sebastián Ríos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

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

Pruebas Realizadas

Las pruebas para este requerimiento se realizaron en un computador con las siguientes especificaciones: AMD Ryzen 5 3600 6-Core Processor 3.59 GHz, 16,0 GB de RAM, RX570 y Windows 10 Home Single Language.

Entrada	**Tiempo (s)**
Small	20.87
5pct	68.73
10pct	114.56
20pct	211.29
30pct	332.67
50pct	396.06
80pct	653.99
Large	736.87

Tablas de datos

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	20.87
5 pct	Dato2	68.73
10 pct	Dato3	114.56
20 pct	Dato4	211.29
30 pct	Dato5	332.67
50 pct	Dato6	396.06
80 pct	Dato7	653.99
large	Dato8	736.87

Graficas

Análisis

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

Requerimiento <<7>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si, implementado por Cristian Vega, Simón Gonzales y Juan Sebastián Ríos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

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

Pruebas Realizadas

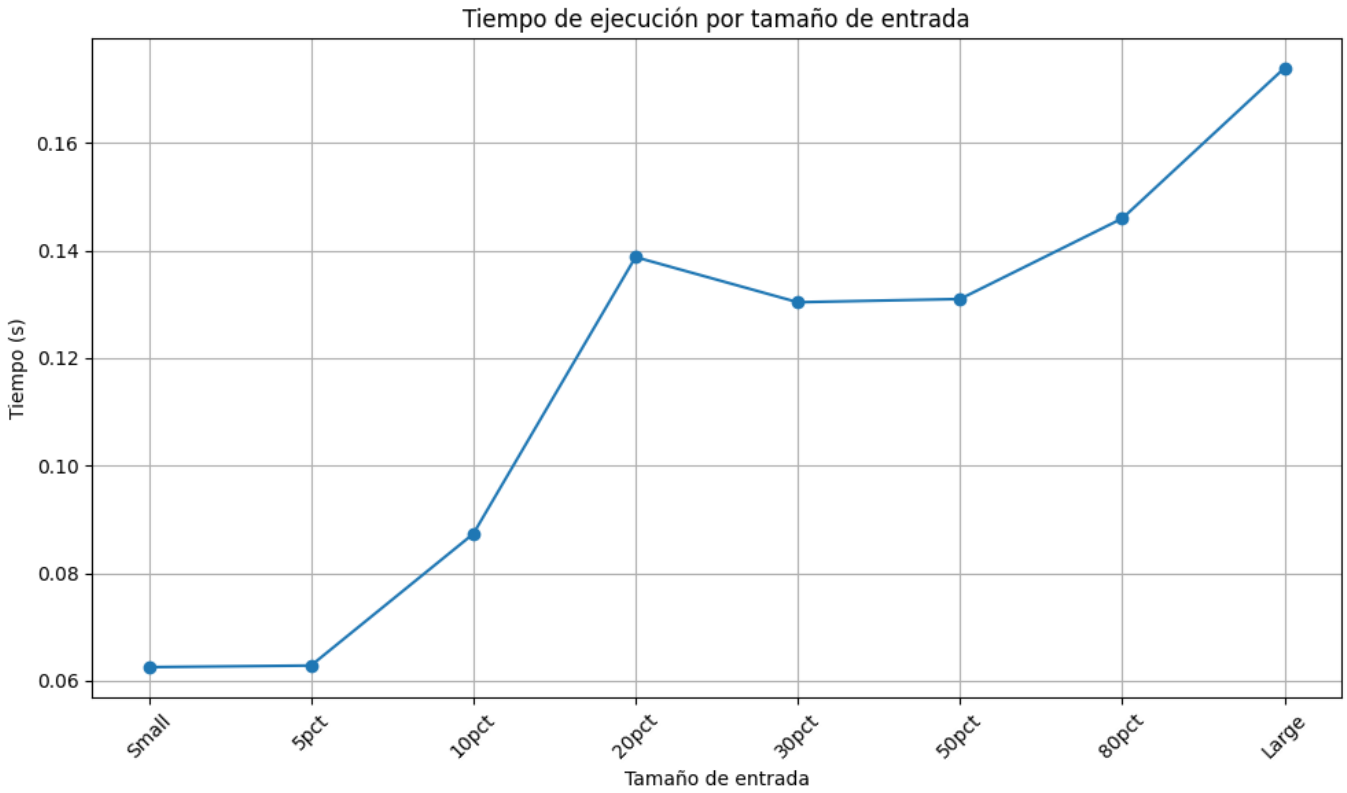
Las pruebas para este requerimiento se realizaron en un computador con las siguientes especificaciones: AMD Ryzen 5 3600 6-Core Processor 3.59 GHz, 16,0 GB de RAM, RX570 y Windows 10 Home Single Language.

Entrada	**Tiempo (s)**
Small	0.0625
5pct	0.0628
10pct	0.0873
20pct	0.1388
30pct	0.1304
50pct	0.131
80pct	0.146
Large	0.174

Tablas de datos

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	0.0625
5 pct	Dato2	0.0628
10 pct	Dato3	0.0873
20 pct	Dato4	0.1388
30 pct	Dato5	0.1304
50 pct	Dato6	0.131
80 pct	Dato7	0.146
large	Dato8	0.174

Graficas



Análisis

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

Requerimiento <<8>>

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

Descripción

Breve descripción de como abordaron la implementación del requerimiento

Entrada	Parámetros necesarios para resolver el requerimiento.
Salidas	Respuesta esperada del algoritmo.
Implementado (Sí/No)	Si, implementado por Cristian Vega, Simón Gonzales y Juan Sebastián Ríos

Análisis de complejidad

Análisis de complejidad de cada uno de los pasos del algoritmo

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

Pruebas Realizadas

Las pruebas para este requerimiento se realizaron en un computador con las siguientes especificaciones: AMD Ryzen 5 3600 6-Core Processor 3.59 GHz, 16,0 GB de RAM, RX570 y Windows 10 Home Single Language.

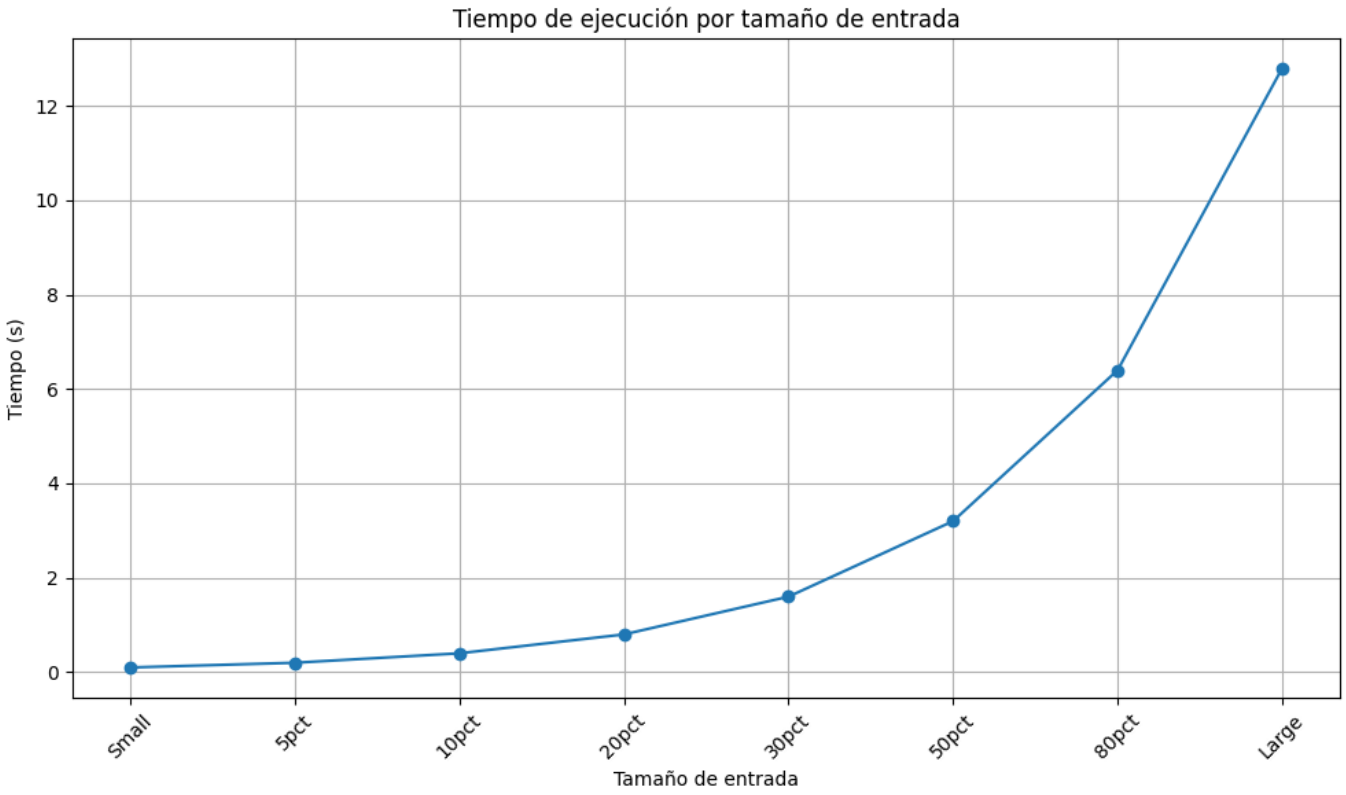
Entrada	**Tiempo (s)**
Small	0.1
5pct	0.2
10pct	0.4
20pct	0.8
30pct	1.6
50pct	3.2
80pct	6.4
Large	12.8

Tablas de datos

Muestra	**Salida**	**Tiempo (ms)**
small	Dato1	0.1
5 pct	Dato2	0.2
10 pct	Dato3	0.4
20 pct	Dato4	0.8
30 pct	Dato5	1.6
50 pct	Dato6	3.2
80 pct	Dato7	6.4

large	Dato8	12.8

Graficas



Análisis

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