ANÁLISIS DEL RETO

Tomas Aponte, 202420148, t.aponte@uniandes.edu.co

Juan Diego García, 202423575jd.garcia12@uniandes.edu.co

Carga de datos

```python
def new_logic():
    """
    Crea el catalogo para almacenar las estructuras de datos
    """
    #TODO: Llama a las funciónes de creación de las estructuras de datos
    catalog = {
        "crimes": None,
        "date_tree": None,
        "area_map": None,
        "area_name_map": None
    }
    catalog["crimes"] = al.new_list()
    catalog["date_tree"] = bst.new_map()
    catalog["area_map"] = mp.new_map(21, 0.7)
    catalog["area_name_map"] = mp.new_map(21, 0.7)

    return catalog


def load_data(catalog, filename=(data_dir+"Crime_in_LA_20.csv")):
    """
    Carga los datos del reto
    """
    # TODO: Realizar la carga de datos
    file = open(filename, encoding="utf-8")
    reader = csv.DictReader(file)
    for row in reader:
        crime={
            "DR_NO": row["DR_NO"],
            "Date Rptd": datetime.strptime(row["Date Rptd"], "%m/%d/%Y %I:%M:%S %p"),
            "DATE OCC": datetime.strptime(row["DATE OCC"], "%m/%d/%Y %I:%M:%S %p"),
            "TIME OCC": row["TIME OCC"],
            "AREA": row["AREA"] if "AREA" in row else "Unknown",
            "AREA NAME": row["AREA NAME"],
            "Rpt Dist No": row["Rpt Dist No"],
            "Part 1-2": row["Part 1-2"],
            "Crm Cd": row["Crm Cd"],
            "Crm Cd Desc": row["Crm Cd Desc"],
            "Vict Age": row["Vict Age"],
            "Vict Sex": row["Vict Sex"],
            "Vict Descent": row["Vict Descent"],
            "Premise Cd": row["Premise Cd"] if "Premise CD" in row else "Unknown",
            "Premise Desc": row["Premise Desc"] if "Premise Desc" in row else "Unknown",
            "Status": row["Status"],
            "Status Desc": row["Status Desc"],
            "LOCATION": row["LOCATION"],
            "LAT": row["LAT"],
            "LON": row["LON"]
        }
        al.add_last(catalog["crimes"], crime)

    file.close()
    return catalog["crimes"]["size"]

def create_tree(catalog):
    """
    Crea el árbol para las fechas
    """
    for i in range(catalog["crimes"]["size"]):
        crime = catalog["crimes"]["elements"][i]
        bst.put(catalog["date_tree"], crime["Date Rptd"], crime["Vict Age"])
    return catalog["date_tree"]

def create_area_map(catalog):
    """
    Crea el mapa para las áreas
    """
    for i in range(1, 21):
        if not mp.contains(catalog["area_map"], i):
            area_list = al.new_list()
            mp.put(catalog["area_map"], i, area_list)
        else:
            area_list = mp.get(catalog["area_map"], i)
        for j in range(catalog["crimes"]["size"]):
            crime = catalog["crimes"]["elements"][j]
            if crime["AREA"] is not "Unknown":
                if int(crime["AREA"]) == i:
                    al.add_last(area_list, crime)
```

La carga de datos está dividida en 3 funciones: new_logic(), load_data(), create_tree(), create_map(). New_logic() se encarga de construir un catálogo general donde estarán guardadas una lista de arreglos (crimes), un árbol binario de búsqueda (date_tree) y un mapa linear probing (area_map). Load_data() se encarga de llenar la lista de arreglos, generando un diccionario por cada fila del archivo csv, donde la llave del diccionario se nombra después de la columna de donde viene el dato. Una vez se tenga el diccionario con todos los datos, se agrega al catálogo a la lista de arreglos, y se procede a llenar el diccionario de la siguiente fila, quedando así la totalidad de los datos cargados. Por otro lado, create_tree() se encarga de construir el árbol del catálogo, donde las llaves son la fecha del reporte de un crimen dado, y su valor la edad de la víctima del crimen; mientras que create_map() se encarga de generar un mapa donde cada llave corresponde con el código de un área dada, y el valor de cada llave son los diccionarios asociados al área. Estas estructuras se crean por aparte debido a la complejidad temporal de la creación del árbol. Juntar todas las estructuras en una carga de datos aumentaría significativamente el tiempo de respuesta para cargar los primeros y últimos 5 registros del catálogo, que solo necesitan de la lista de diccionarios.

Descripción

Entrada	N/A
Salidas	Total de reportes, primeros y últimos 5 datos
Implementado (Sí/No)	Si, Juan Diego García

Análisis de complejidad

Pasos	**Complejidad**
Crea el catalogo para todas las estructuras de datos	O(1)
Extrae todos los registros a una lista	O(n)
Crea y llena las estructuras de datos necesarias	O(n)
Selecciona las primeros y últimos 5	O(n)
TOTAL	***O(n)***

Pruebas Realizadas

Procesador: Ryzen 7 7730U Radeon Graphics

RAM: 16GB

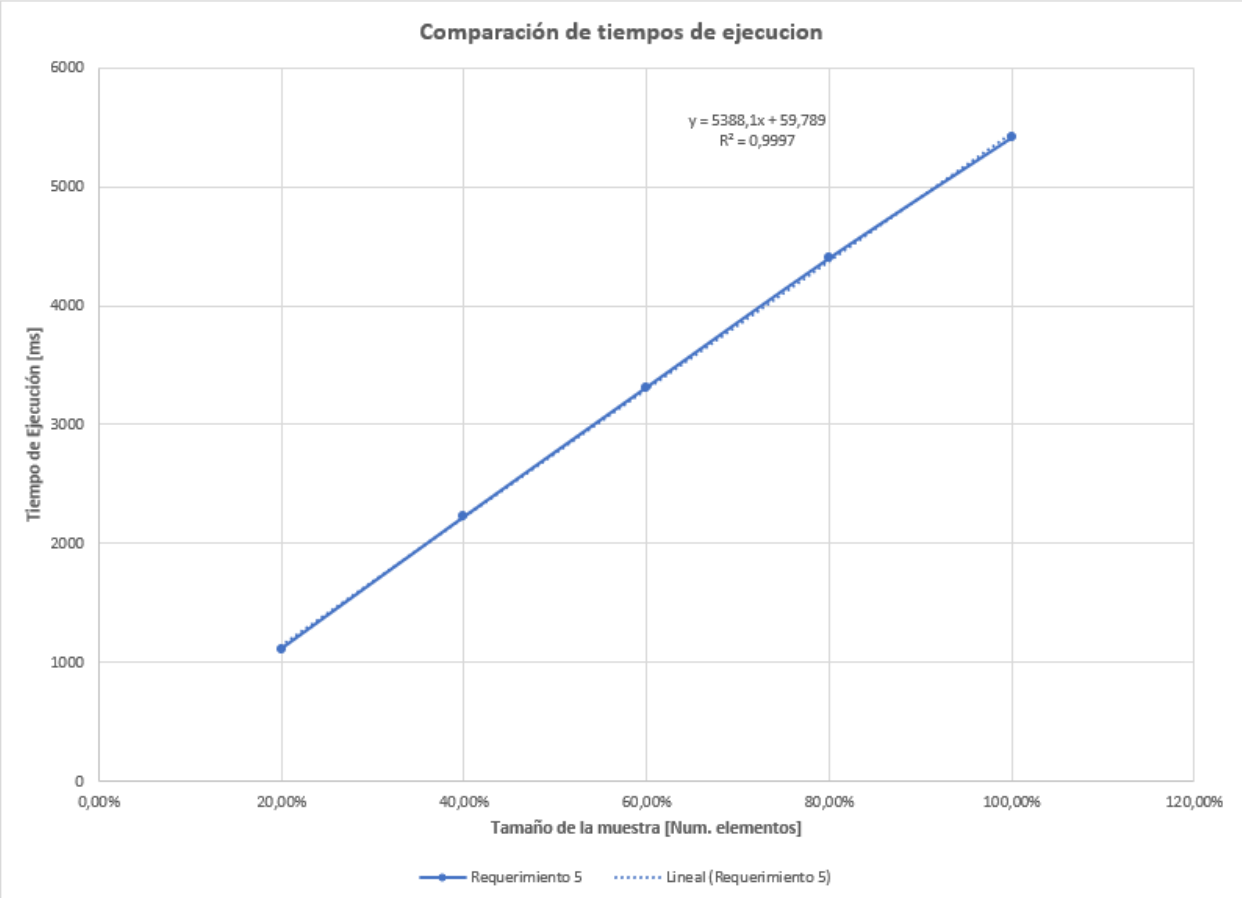
SO: Windows 11

Entrada	**Tiempo (ms)**
20%	1113,62
40%	2223,77
60%	3312,88
80%	4398,92
100%	5414,19

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Carga de datos
20,00%	63088,00	1113,62
40,00%	126176,00	2223,77
60,00%	189264,00	3312,88
80,00%	252352,00	4398,92
100,00%	315440,00	5414,19

Graficas



Análisis

La grafica confirma la complejidad temporal propuesta anteriormente O(n). Esto se debe a que la longitud de su ejecución depende de la cantidad de datos que contienen los registros para cargarse en su estructura asignada.

Requerimiento 1

```python
def req_1(catalog, anno_0, anno_1):
    """
    Retorna el resultado del requerimiento 1
    """
    # TODO: Modificar el requerimiento 1
    start_time = get_time()
    year0 = datetime.strptime(anno_0, "%Y-%m-%d")
    year1 = datetime.strptime(anno_1, "%Y-%m-%d")
    if year0 > year1:
        print("El año inicial no puede ser mayor al año final")
        return None
    filtro = bst.keys(catalog["date_tree"], year0, year1)
    lista_retorno = al.new_list()
    sl.merge_sort(filtro, True)
    current_node = filtro["first"]
    while current_node is not None:
        for j in range(catalog["crimes"]["size"]):
            value = current_node["info"]
            crime = catalog["crimes"]["elements"][j]
            if crime["Date Rptd"] == value:
                info = {
                    "DR_NO": crime["DR_NO"],
                    "DATE OCC": crime["DATE OCC"],
                    "TIME OCC": crime["TIME OCC"],
                    "AREA NAME": crime["AREA NAME"],
                    "Crm Cd": crime["Crm Cd"],
                    "Crm Cd Desc": crime["Crm Cd Desc"],
                    "LOCATION": crime["LOCATION"],
                }
                al.add_last(lista_retorno, info)
        current_node = current_node["next"]
        if lista_retorno["size"] > 10:
            new_list = al.new_list()
            for i in range(0, 5):
                al.add_last(new_list, lista_retorno["elements"][i])
            for i in range(-6, -1):
                al.add_last(new_list, lista_retorno["elements"][i])
            lista_retorno = new_list
    end_time = get_time()
    elapsed_time = delta_time(start_time, end_time)
    return lista_retorno, filtro["size"], elapsed_time
```

Este requerimiento se encarga de recibir dos fechas para las cuales retorna todos los registros que están dentro de este periodo de tiempo. Primero se formatean las fechas, después se descartan fechas invalidas, se filtran y organizan los datos, se crea una lista de retorno la cual extrae los registros relevantes para el criterio, finalmente se acorta la lista si es necesario.

Descripción

Entrada	Catálogo, fecha inicio, fecha fin
Salidas	Reportes dentro del periodo de tiempo
Implementado (Sí/No)	Si, Juan Diego García

Análisis de complejidad

Pasos	**Complejidad**
Formatea los valores de fecha y verifica validez	O(1)
Extrae las llaves del mapa	O(n)
Organiza los datos	O(n log n)
Extrae reportes relevantes a una lista	O(n)
TOTAL	***O(n log n)***

Pruebas Realizadas

Procesador: Ryzen 7 7730U Radeon Graphics

RAM: 16GB

SO: Windows 11

Fecha inicio: 2020-01-01

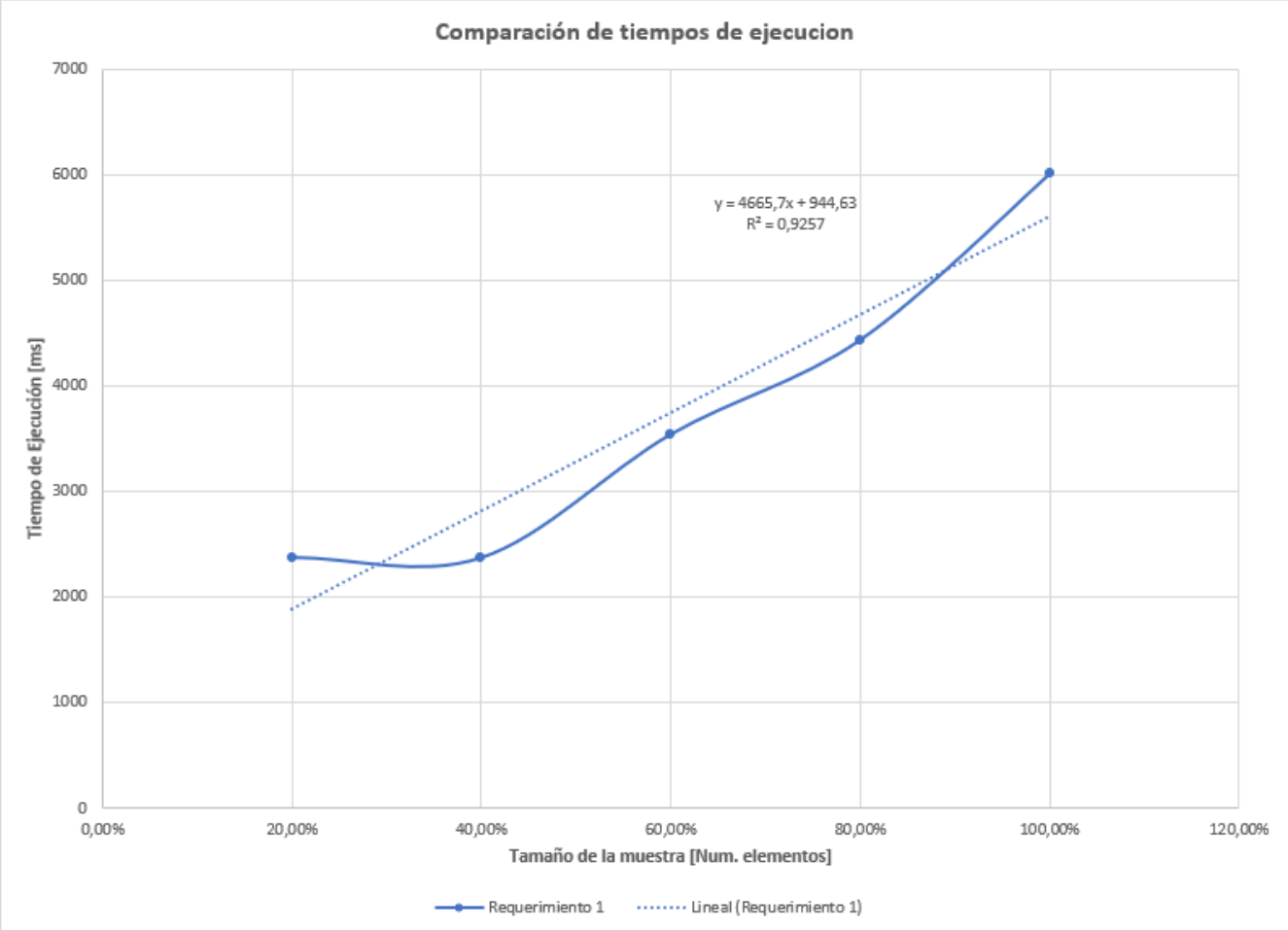
Fecha fin: 2020-05-02

Entrada	**Tiempo (ms)**
20%	2372,80
40%	2369.22
60%	3538.62
80%	4432.99
100%	6006.62

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 1
20,00%	63088,00	2372,8
40,00%	126176,00	2369,22
60,00%	189264,00	3538,62
80,00%	252352,00	4432,99
100,00%	315440,00	6006,62

Graficas



Análisis

Es posible evidenciar en la grafica unos cambios notables en el comportamiento de el tiempo de ejecución en cuanto a la cantidad de datos. Esto puede deberse a la complejidad O(n log n) de merge_sort y al recorrido que se le da al mapa para extraer las llaves de sus respectivos valores y así realizar comparaciones sencillamente.

Requerimiento 3

```python
def req_3(catalog, N: int, area_name: str):
    """
    Requerimiento 3: Obtener N crimenes más recientes en un área especifica.
    """
    start_time = get_time()

    # Convertimos el nombre del área a minúsculas para buscar en el mapa
    area_name_map = catalog["area_name_map"]
    crimes_list_entry = mp.get(area_name_map, area_name.lower())

    if crimes_list_entry is None:
        print(f"No se encontraron crimenes reportados en el área: {area_name}")
        return None

    crimes_list = crimes_list_entry
    total_crimes = al.size(crimes_list)

    if total_crimes == 0:
        print(f"No se encontraron crimenes reportados en el área: {area_name}")
        return None

    # Ordenamos por fecha descendente
    sorted_crimes = al.merge_sort(crimes_list, cmp_function=compare_by_date_desc)

    # Recortamos la lista a los N crimenes más recientes
    if al.size(sorted_crimes) > N:
        sorted_crimes["elements"] = sorted_crimes["elements"][:N]
        sorted_crimes["size"] = N

    # Creamos la lista final con la información requerida
    recent_crimes = al.new_list()
    for i in range(al.size(sorted_crimes)):
        crime = sorted_crimes["elements"][i]
        crime_info = {
            "DR_NO": crime["DR_NO"],
            "DATE OCC": crime["DATE OCC"],
            "TIME OCC": crime["TIME OCC"],
            "AREA NAME": crime["AREA NAME"],
            "Rpt Dist No": crime["Rpt Dist No"],
            "Part 1-2": crime["Part 1-2"],
            "Crm Cd": crime["Crm Cd"],
            "Status": crime["Status"],
            "LOCATION": crime["LOCATION"]
        }
        al.add_last(recent_crimes, crime_info)

    end_time = get_time()
    elapsed_time = (end_time - start_time)

    return recent_crimes, total_crimes, elapsed_time
```

Este requerimiento se encarga de retornar una cantidad de registros especificada en un área específica desde el catálogo. Se guardan en una variable todos los registros que coinciden con el área deseada por el usuario. Se organiza la lista con merge_sort y se corta a la longitud deseada. Finalmente se guardan todos los datos del registro que coinciden con el criterio en una lista final.

Descripción

Entrada	Catálogo, numero de registros deseado, nombre de un área.
Salidas	Numero de reportes deseado para el área elegida.
Implementado (Sí/No)	Si, Tomas Aponte

Análisis de complejidad

Pasos	**Complejidad**
Obtiene los valores del mapa	O(n)
Organiza los datos	O (n log n)
Acorta la lista	O(1)
Extrae reportes relevantes a una lista	O(n)
TOTAL	***O (n log n)***

Pruebas Realizadas

Procesador: Ryzen 7 7730U Radeon Graphics

RAM: 16GB

SO: Windows 11

Nombre de área = West LA

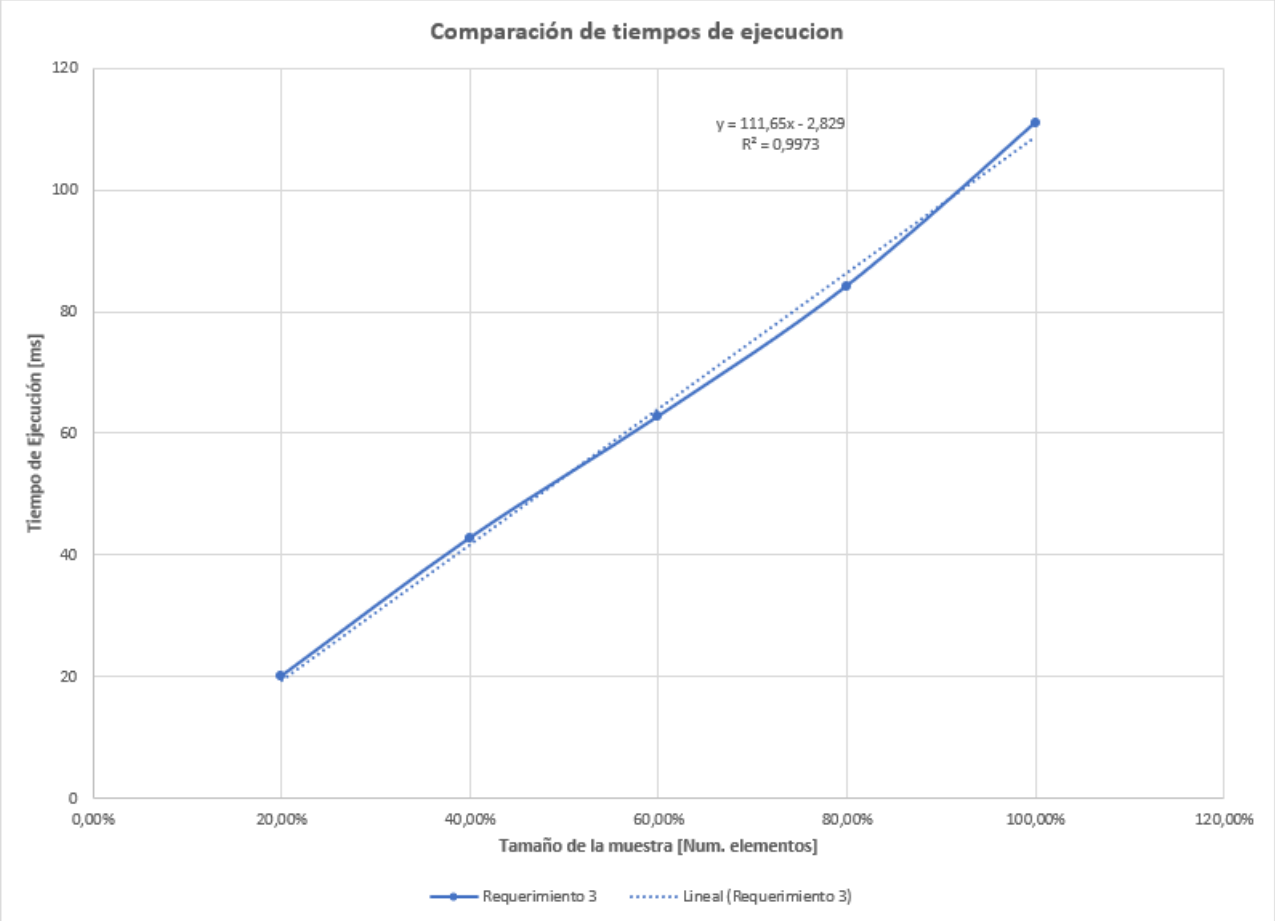
Numero de registros = 10

Entrada	**Tiempo (ms)**
20%	20,07
40%	42,77
60%	62,76
80%	84,18
100%	111,01

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 3
20,00%	63088,00	20,07
40,00%	126176,00	42,77
60,00%	189264,00	62,76
80,00%	252352,00	84,18
100,00%	315440,00	111,01

Graficas



Análisis

En esta grafica se puede evidenciar un claro comportamiento O(n) esto se debe a que la mayoría de los procesos llevados a cabo en el algoritmo son de esta complejidad temporal. Al ser merge_sort estable no afecta mucho la complejidad temporal y garantiza resultados constantes.

Requerimiento 5

```python
def req_5(catalog, n, anno_0, anno_1):
    """
    Retorna el resultado del requerimiento 5
    """
    # TODO: Modificar el requerimiento 5
    start_time = get_time()
    year0 = datetime.strptime(anno_0, "%Y-%m-%d")
    year1 = datetime.strptime(anno_1, "%Y-%m-%d")
    if year0 > year1:
        print("El año inicial no puede ser mayor al año final")
        return None
    areas = catalog["area_map"]
    stats = al.new_list()
    n = int(n)
    for code in range(1, 21):
        crimes = mp.get(areas, code)
        unresolved = al.new_list()
        for i in range(crimes["size"]):
            crime = crimes["elements"][i]
            if year0 <= crime["Date Rptd"] <= year1 and crime["Status"] == "IC":
                if crime not in unresolved["elements"]:
                    al.add_last(unresolved, crime)
        if unresolved["size"] > 0:
            unresolved = sort_unresolved(unresolved)
            first = unresolved["elements"][-1]["DATE OCC"]
            last = unresolved["elements"][0]["DATE OCC"]
            info ={
                "AREA NAME": unresolved["elements"][0]["AREA NAME"],
                "AREA": code,
                "Crimes": unresolved["size"],
                "first": first,
                "last": last,
            }
            al.add_last(stats, info)
    stats = sort_stats(stats)
    if stats["size"] > n:
        new_stats = al.new_list()
        for i in range(0, n):
            al.add_last(new_stats, stats["elements"][i])
        stats = new_stats
    endtime = get_time()
    elapsed_time = delta_time(start_time, endtime)
    return stats, elapsed_time
```

Este requerimiento se encarga de retornar una cantidad de áreas definida por el usuario dentro de un margen de tiempo especificado. El código formatea los datos de fecha, verifica su validez, obtiene los datos relevantes del mapa y los inserta en una lista. Después, se organizan los datos y finalmente se acorta al tamaño deseado.

Descripción

Entrada	Catálogo, numero de registros deseado, nombre de un área.
Salidas	Numero de reportes deseado para el área elegida.
Implementado (Sí/No)	Si, Tomas Aponte

Análisis de complejidad

Pasos	**Complejidad**
Formatea y valida la fecha	O(1)
Extrae los elementos útiles del mapa	O(n)
Recorre la lista en búsqueda de coincidencias	O (n log n)

Extrae reportes relevantes a una lista	O(n)
Organiza y recorta los registros	O(n log n)
TOTAL	***O (n log n)***

Pruebas Realizadas

Procesador: Ryzen 7 7730U Radeon Graphics

RAM: 16GB

SO: Windows 11

Fecha inicio: 2020-01-10

Fecha fin: 2020-06-06

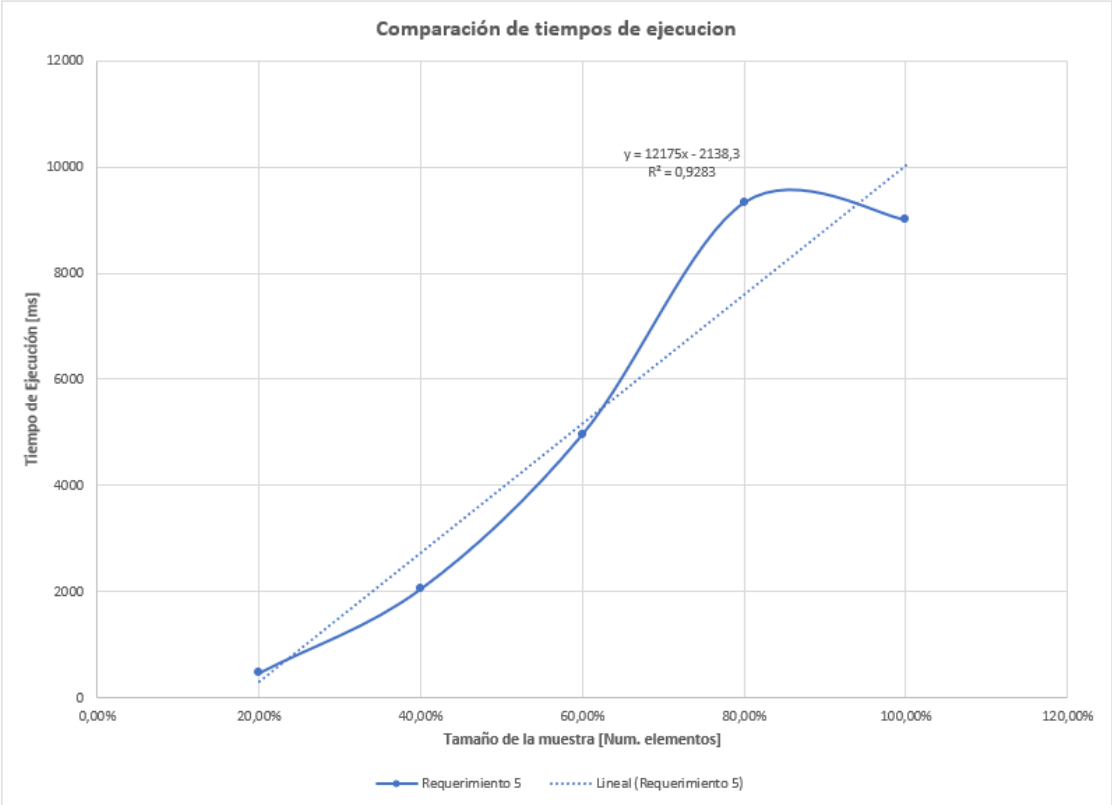
Numero de áreas: 20

Entrada	**Tiempo (ms)**
20%	470,57
40%	2055,12
60%	4968,79
80%	9331,31
100%	9007,40

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 5
20,00%	63088,00	470,57
40,00%	126176,00	2055,12
60,00%	189264,00	4968,79
80,00%	252352,00	9331,31
100,00%	315440,00	9007,4

Graficas



Análisis

Esta es la grafica que tiene el comportamiento más ambiguo en la creación de los requisitos. Se podría atribuir el recorrido de la lista a su comportamiento, así como el ordenamiento de datos pero no parece haber nada fuera de lo normal que pudiera causar este cambio repentino en su estructura.

Requerimiento 6

```python
def req_5(catalog, n, anno_0, anno_1):
    """
    Retorna el resultado del requerimiento 5
    """
    # TODO: Modificar el requerimiento 5
    start_time = get_time()
    year0 = datetime.strptime(anno_0, "%Y-%m-%d")
    year1 = datetime.strptime(anno_1, "%Y-%m-%d")
    if year0 > year1:
        print("El año inicial no puede ser mayor al año final")
        return None
    areas = catalog["area_map"]
    stats = al.new_list()
    n = int(n)
    for code in range(1, 21):
        crimes = mp.get(areas, code)
        unresolved = al.new_list()
        for i in range(crimes["size"]):
            crime = crimes["elements"][i]
            if year0 <= crime["Date Rptd"] <= year1 and crime["Status"] == "IC":
                if crime not in unresolved["elements"]:
                    al.add_last(unresolved, crime)
        if unresolved["size"] > 0:
            unresolved = sort_unresolved(unresolved)
            first = unresolved["elements"][-1]["DATE OCC"]
            last = unresolved["elements"][0]["DATE OCC"]
            info ={
                "AREA NAME": unresolved["elements"][0]["AREA NAME"],
                "AREA": code,
                "Crimes": unresolved["size"],
                "first": first,
                "last": last,
            }
            al.add_last(stats, info)
    stats = sort_stats(stats)
    if stats["size"] > n:
        new_stats = al.new_list()
        for i in range(0, n):
            al.add_last(new_stats, stats["elements"][i])
        stats = new_stats
    endtime = get_time()
    elapsed_time = delta_time(start_time, endtime)
    return stats, elapsed_time
```

El requerimiento recibe un numero de áreas y un margen de tiempo para el que debe obtener los registros que cumplan el criterio propuesto. Se encarga de hacer esto obteniendo los valores oportunos de un mapa con la información de los registros, creando una lista para probar los requisitos, guardando los registros que aciertan con el criterio, organizando los registros y retornando el resultado acortado por área.

Descripción

Entrada	Catálogo, numero de área deseados, rango de tiempo.
Salidas	Numero de áreas deseado, información de los registros acordes.
Implementado (Sí/No)	Si, Juan Diego García

Análisis de complejidad

Pasos	**Complejidad**
Extrae las llaves del mapa	O(n)
Recorre la lista en busca de coincidencias	O(n)

Organiza los datos	O (n log n)
Extrae reportes relevantes a una lista y la acorta	O(n)
TOTAL	***O (n log n)***

Pruebas Realizadas

Procesador: Ryzen 7 7730U Radeon Graphics

RAM: 16GB

SO: Windows 11

Mes: 05

Género: F

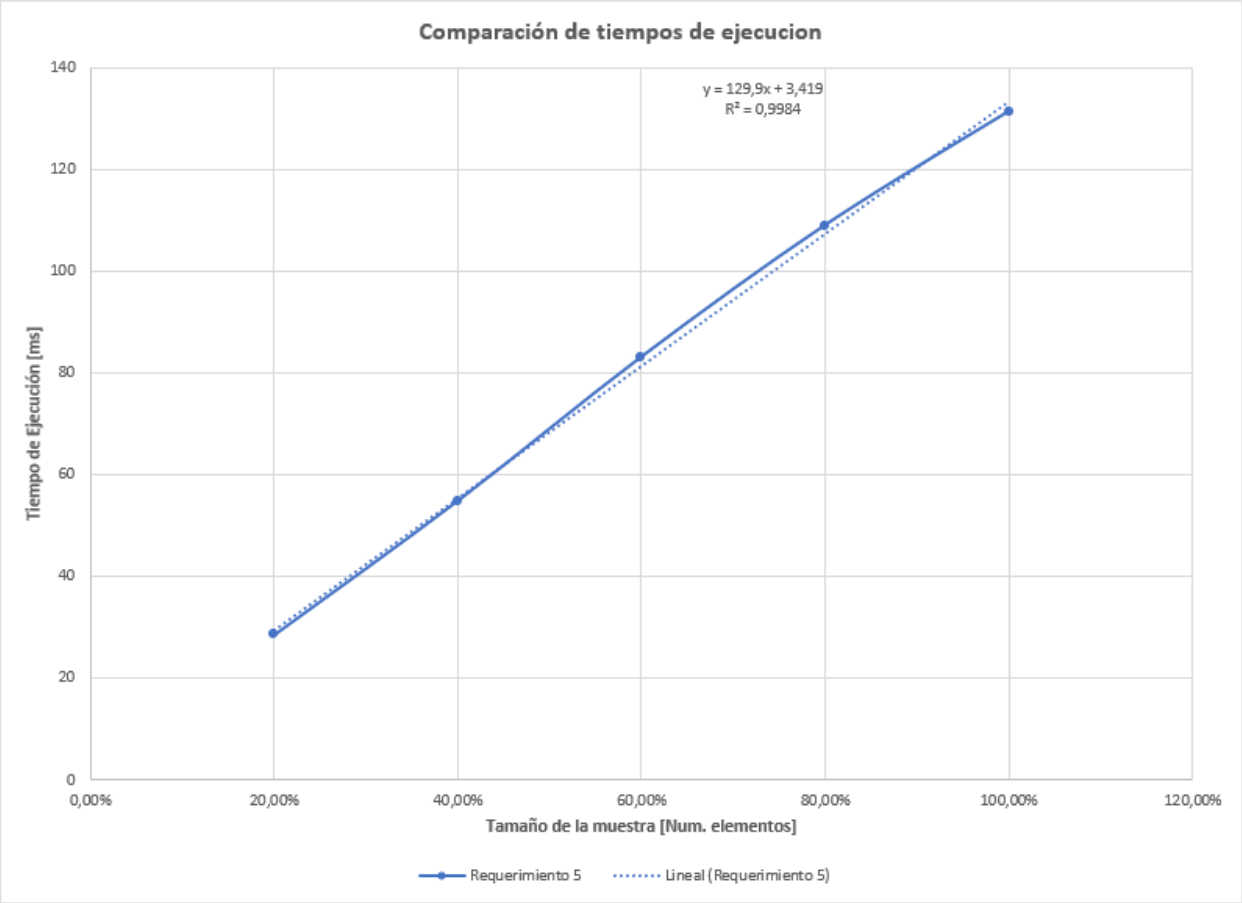
Numero de áreas: 15

Entrada	**Tiempo (ms)**
20%	28,49
40%	54.85
60%	83.14
80%	108.98
100%	131.32

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 6
20,00%	63088,00	28,49
40,00%	126176,00	54,85
60,00%	189264,00	83,14
80,00%	252352,00	108,98
100,00%	315440,00	131,32

Graficas



Análisis

La grafica demuestra el comportamiento definido por los procesos que se ejecutan dentro del código del requisito. La leve curva que se observa en la grafica puede ser gracias a el recorrido de el diccionario en búsqueda de registros que coincidieran con el criterio, la extracción de llaves del mapa o la organización de los datos. Pero pareciera tender a O(n) mas que otra cosa.

Requerimiento 7

```
def req_7(catalog, N, sex, age_min, age_max):
    """
    Determina los N crímenes más comunes para víctimas de un sexo específico en un rango de edad dado.
    """
    start_time = get_time()

    # Usaremos un mapa para contar por código de crimen
    crime_stats = {}

    # Recorrer todos los crímenes
    crimes_list = catalog["crimes"]
    size = al.size(crimes_list)

    for i in range(size):
        crime = al.get_element(crimes_list, i)

        try:
            # Verificar campos necesarios
            if (not crime["Vict Sex"] or not crime["Vict Age"] or
                not crime["Crm Cd"] or not crime["DATE OCC"]):
                continue

            victim_sex = crime["Vict Sex"].upper()
            victim_age = int(crime["Vict Age"])
            crime_code = crime["Crm Cd"]
            crime_year = crime["DATE OCC"].year

            # Filtrar por sexo y rango de edad
            if victim_sex == sex.upper() and age_min <= victim_age <= age_max:
                # Inicializar estructura si no existe
                if crime_code not in crime_stats:
                    crime_stats[crime_code] = {
                        "code": crime_code,
                        "total": 0,
                        "ages": {},
                        "years": {}
                    }

                # Actualizar estadísticas
                crime_stats[crime_code]["total"] += 1

                # Actualizar conteo por edad
                if victim_age in crime_stats[crime_code]["ages"]:
                    crime_stats[crime_code]["ages"][victim_age] += 1
                else:
                    crime_stats[crime_code]["ages"][victim_age] = 1

                # Actualizar conteo por año
                if crime_year in crime_stats[crime_code]["years"]:
                    crime_stats[crime_code]["years"][crime_year] += 1
                else:
                    crime_stats[crime_code]["years"][crime_year] = 1

        except (ValueError, TypeError, AttributeError, KeyError):
            continue

    # Convertir a lista
    crime_list = al.new_list()
    for code in crime_stats:
        crime_data = crime_stats[code]

        # Convertir diccionarios a listas de tuplas
        ages_list = [(age, count) for age, count in crime_data["ages"].items()]
        years_list = [(year, count) for year, count in crime_data["years"].items()]

        al.add_last(crime_list, {
            "code": crime_data["code"],
            "total": crime_data["total"],
            "ages": ages_list,
            "years": years_list
        })

    # Ordenar la lista por total de crímenes (descendente)
    def compare_crimes(crime1, crime2):
        return crime2["total"] - crime1["total"]

    sorted_crimes = al.merge_sort(crime_list, sort_criteria=False, cmp_function=compare_crimes)

    # Tomar los primeros N elementos
    result_size = min(N, al.size(sorted_crimes))
    result = al.sub_list(sorted_crimes, 0, result_size)

    end_time = get_time()
    return result, delta_time(start_time, end_time)
```

Este requerimiento debe retornar los crímenes mas comunes para las víctimas de un cierto genero y edad.

Descripción

Entrada	Catálogo, edad, genero.
Salidas	Crimen más recurrente según el criterio.
Implementado (Sí/No)	Si, Tomas Aponte

Análisis de complejidad

Pasos	**Complejidad**
Crea listas y variables necesarias	O(1)
Extrae las llaves del mapa	O(n)
Recorre la lista en busca de coincidencias	O(n)
Organiza los datos	O (n log n)
Extrae reportes relevantes a una lista y la acorta	O(n)
TOTAL	***O (n log n)***

Pruebas Realizadas

Procesador: Ryzen 7 7730U Radeon Graphics

RAM: 16GB

SO: Windows 11

N: 10

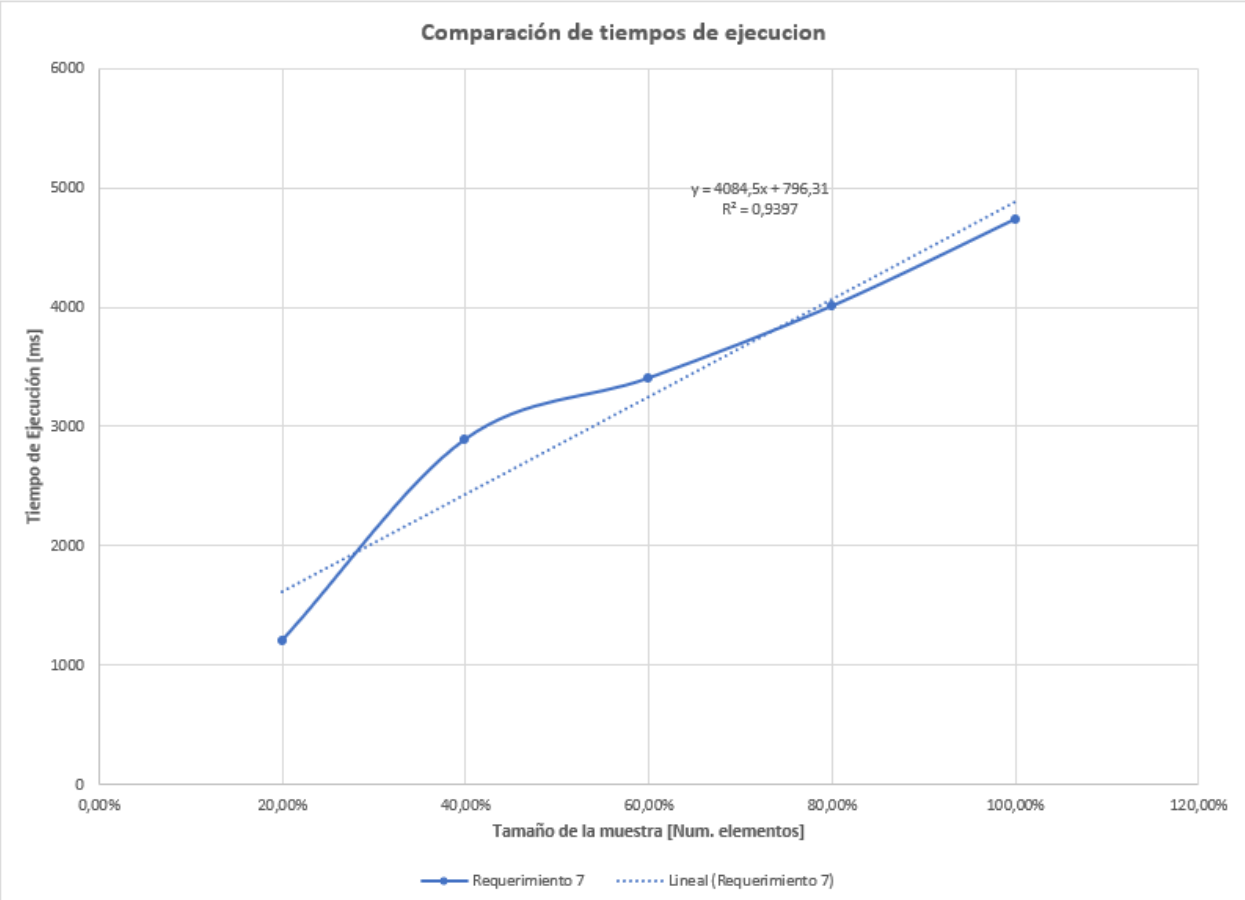
Género: M

Entrada	**Tiempo (ms)**
20%	1204,90
40%	2890,89
60%	3401,98
80%	4004,76
100%	4732,45

Tablas de datos

Porcentaje de la muestra [pct]	Tamaño de la muestra	Requerimiento 7
20,00%	63088,00	1204,9
40,00%	126176,00	2890,89
60,00%	189264,00	3401,98
80,00%	252352,00	4004,76
100,00%	315440,00	4732,45

Graficas



Análisis

La grafica demuestra el comportamiento definido por los procesos que se ejecutan dentro del código del requisito. La leve curva que se observa en la gráfica puede ser gracias a el recorrido del diccionario en búsqueda de registros que coincidieran con el criterio, la extracción de llaves del mapa o la organización de los datos. Al fin no pudo ser posible la solución de errores dentro del código.