# ANÁLISIS DEL RETO

*Estudiante 1, código 1, email 1*

*Estudiante 2, código 2, email 2*

*Estudiante 3, código 3, email 3*

## Requerimiento 1

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 se implementó y quien lo hizo. |

### 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

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).

| **Entrada** | **Tiempo (s)** |
|---|---|
| | |
| | |
| | |

### Tablas de datos

Las tablas con la recopilación de datos de las pruebas.

## Graficas
Las gráficas con la representación de las pruebas realizadas.

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

# 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 |

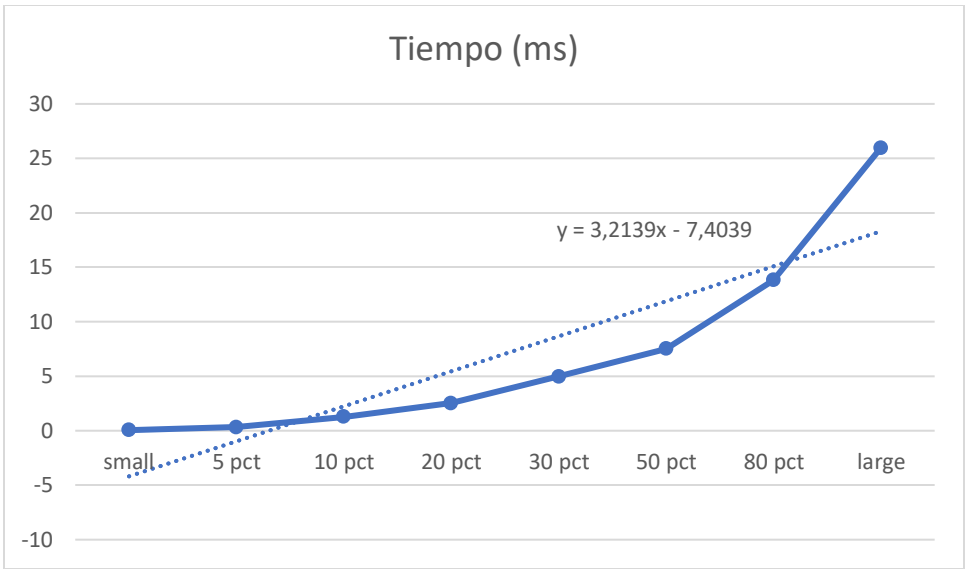
## Tablas de datos

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.

# Requerimiento 1

```python
def req_1(data_structs, Number, country_code, level):
    """
    Función que soluciona el requerimiento 1
    """
    # TODO: Realizar el requerimiento 1
    oferta=[]
    for item in data_structs["jobs"]["elements"]:
        if country_code in item["country_code"] and level in item["experience_level"]:
            oferta.append(item)
            if len(oferta) == Number:
                return oferta
    if len(oferta)<Number:
        return oferta
```

Este requerimiento nos permite añadir las ofertas de trabajo que compartan con los parametros que nos otorga el usuario. Al completar el numero de ofertas la funcion retorna la lista donde se almacenan y si no cumple el numero devuelve las que alcanzaron.

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, numero de ofertas, código de país, nivel de experticia |
| **Salidas** | La lista con las ofertas |
| **Implementado (Sí/No)** | Si. Implementado por Juan Eduardo Briceño |

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Primer for | O(n) |
| Cualquier comparacion | O(1) |
| ***TOTAL*** | ***O(n)*** |

## Pruebas Realizadas

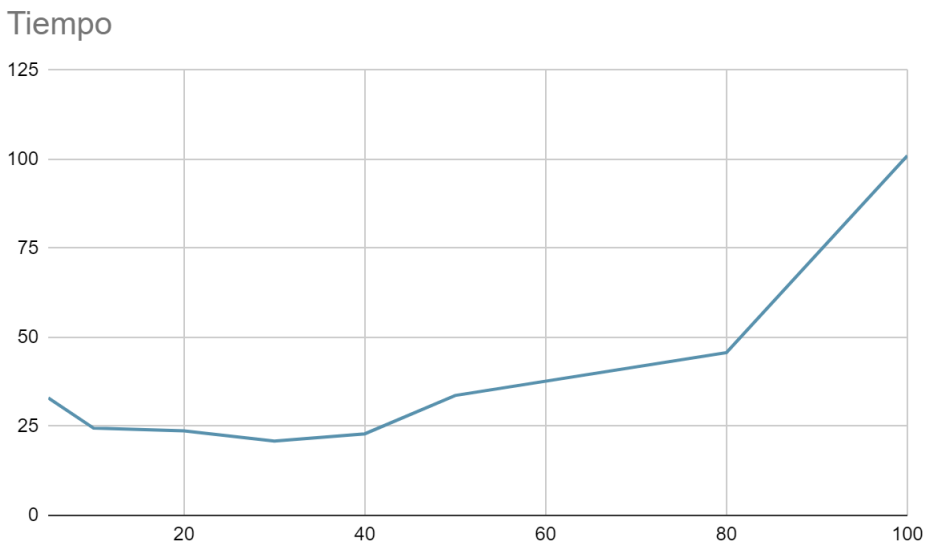
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Numero de ofertas 100, SK, junior

| **Procesadores** | **AMD Ryzen 5 3600 6-Core Processor** |
|---|---|
| **Memoria RAM** | 16 GB |
| **Sistema Operativo** | Windows 10 |

| **Entrada** | **Tiempo (ms)** |
|---|---|
| small | 33.00 |
| 10 pct | 24.50 |
| 20 pct | 23.72 |
| 30 pct | 20.87 |
| 40 pct | 22.91 |

| | |
|---|---|
| 50 pct | 33.67 |
| 80 pct | 45.70 |
| large | 101.1405 |

Grafica




Analisis

Es una funcion que no demora mucho mas al saber que solo es busqueda y no siempre encontrara los numeros de los datos que se le pide, de igual forma tiene una comlpejidad de O(N) al tener que recorrer toda la lista

# Requerimiento 3

```python
def req_3(nombre_empresa, fecha_inicial, fecha_final, data_structs):
    """
    Función que soluciona el requerimiento 3
    """
    # TODO: Realizar el requerimiento 3
    ofertas = []
    def nueva_oferta(fecha, empleo, habilidad, ciudad, pais, tamaño, tipo_lugar, ucranianos):
        dict_oferta = {"fecha": fecha,
                       "empleo": empleo,
                       "habilidad": habilidad,
                       "ciudad": ciudad,
                       "pais": pais,
                       "tamaño": tamaño,
                       "tipo_lugar": tipo_lugar,
                       "ucranianos": ucranianos}
        return dict_oferta
    anio_inicial = int(fecha_inicial[:4])
    mes_inicial = int(fecha_inicial[6:7])
    dia_inicial = int(fecha_inicial[9:10])
    anio_final = int(fecha_final[:4])
    mes_final = int(fecha_final[6:7])
    dia_final = int(fecha_final[9:10])
    for item in data_structs["jobs"]["elements"]:
        fecha_oferta = item["published_at"]
        anio_oferta = int(fecha_oferta[:4])
        mes_oferta = int(fecha_oferta[6:7])
        dia_oferta = int(fecha_oferta[9:10])
        if (nombre_empresa == item["company_name"] and anio_oferta >= anio_inicial and anio_oferta <= anio_final and
            mes_oferta >= mes_inicial and mes_oferta <= mes_final and dia_oferta >= dia_inicial and dia_oferta <= dia_final):
            oferta = nueva_oferta(fecha_oferta, item["title"], item["experience_level"], item["city"], item["country_code"], item["company_size"],
                  item["workplace_type"], item["open_to_hire_ukrainians"])
            ofertas.append(oferta)
    ofertas_ordenadas = sorted(ofertas, key=lambda x: (x["fecha"], x["pais"]))
    return ofertas_ordenadas
```

Este requerimiento nos permite ver la cantidad de ofertas de trabajo que ocurrieron en una empresa en un periodo de tiempo, ademas de decir cuales son de que nivel de experiencia y la cantidad total de ofertas

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, el nombre de |
| **Salidas** | Una lista de diccionarios con cada oferta |
| **Implementado (Sí/No)** | Si. Implementado por Nicolas Mario Romero Colmenares |

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Primer for | O(n) |
| Cualquier comparacion | O(1) |
| ***TOTAL*** | ***O(n)*** |

## Pruebas Realizadas

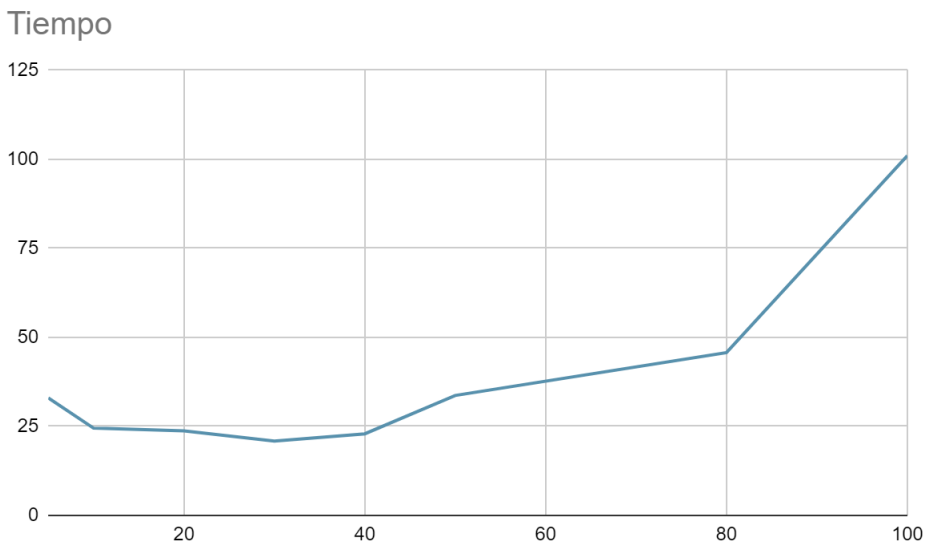
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Numero de ofertas 100, SK, junior

| **Procesadores** | **AMD Ryzen 5 3600 6-Core Processor** |
|---|---|
| **Memoria RAM** | 16 GB |
| **Sistema Operativo** | Windows 10 |

| **Entrada** | **Tiempo (ms)** |
|---|---|

| | |
|---|---|
| small | 33.00 |
| 10 pct | 24.50 |
| 20 pct | 23.72 |
| 30 pct | 20.87 |
| 40 pct | 22.91 |
| 50 pct | 33.67 |
| 80 pct | 45.70 |
| large | 101.1405 |

Grafica




Analisis

Es una funcion que no demora mucho mas al saber que solo es busqueda y no siempre encontrara los numeros de los datos que se le pide, de igual forma tiene una comlpejidad de O(N) al tener que recorrer toda la lista

# Requerimiento 4

```python
def req_4(data_structs, country_code, fecha_inicial, fecha_final):
    """
    Función que soluciona el requerimiento 4
    """
    # TODO: Realizar el requerimiento 4
    oferta=[]
    fecha_inicial=datetime.strptime(fecha_inicial, "%Y-%m-%dT%H:%M:%S.%fZ")
    fecha_final=datetime.strptime(fecha_final, "%Y-%m-%dT%H:%M:%S.%fZ")
    for item in data_structs["jobs"]["elements"]:
        if country_code in item["country_code"]:
            a=item["published_at"]
            a=datetime.strptime(a, "%Y-%m-%dT%H:%M:%S.%fZ")
            if a>fecha_inicial and a<fecha_final:
                oferta.append(item)
    ciudades={}
    empresas={}
    a=len(oferta)
    for item in oferta:
        if item["city"] not in ciudades:
            ciudades[item["city"]]=1
        else:
            ciudades[item["city"]]+=1
    for item in oferta:
        if item["company_name"] not in empresas:
            empresas[item["company_name"]]=1
    num_empresas=len(empresas)
    num_ciudades=len(ciudades)
    mayor= max(ciudades, key=ciudades.get)
    menor= min(ciudades, key=ciudades.get)
    return oferta , a, ciudades, num_ciudades, num_empresas, mayor, menor
```

Primero convierte los inputs del usuario en estilo DATETIME para luego poder hacer la comparacion tambien conviertiendo el del archivos csv. Entonces primero se busca el country code y luego por rango de fechas si cumple se añade a la lista oferta

| | |
|---|---|
| **Entrada** | Estructuras de datos del modelo, codigo pais, fecha inicial, fecha final |
| **Salidas** | La lista con las ofertas, tamaño de la lista, numero de ciudades, numero de empresas, mayor ciudad con ofertas, menor ciudad con empresas |
| **Implementado (Sí/No)** | Si. Implementado por Juan Eduardo Briceño |

Análisis de complejidad de cada uno de los pasos del algoritmo

| **Pasos** | **Complejidad** |
|---|---|
| Cualquier for | O(n) |
| Cualquier comparacion | O(1) |
| ***TOTAL*** | ***O(n)*** |

Al no haber en ningun momento un for dentro de un for no cae en un O(n^2)

## Pruebas Realizadas

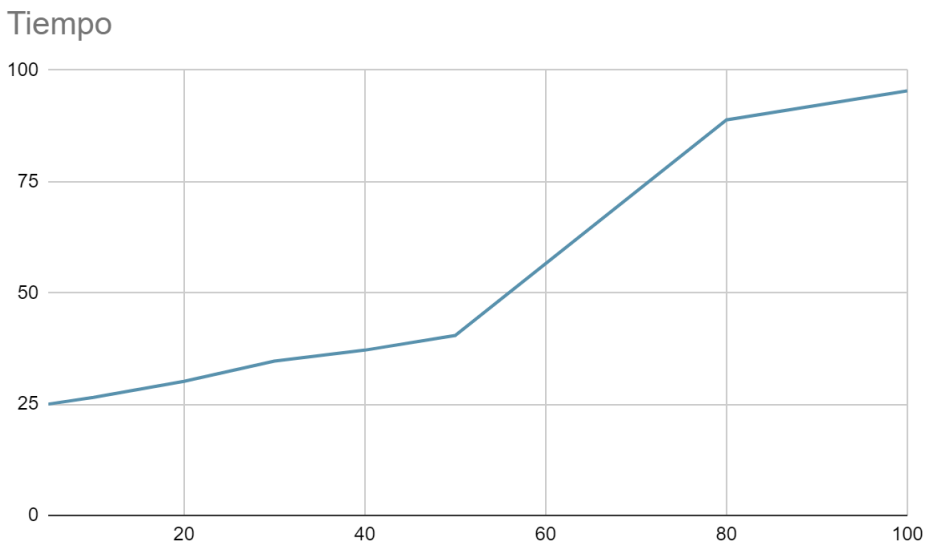
Las pruebas realizadas fueron realizadas en una maquina con las siguientes especificaciones. Pais SK, fecha incial 2022-01-01 fecha final 2023-05-20

| **Procesadores** | **AMD Ryzen 5 3600 6-Core Processor** |
|---|---|

| | |
|---|---|
| **Memoria RAM** | 16 GB |
| **Sistema Operativo** | Windows 10 |

| **Entrada** | **Tiempo (ms)** |
|---|---|
| small | 25 |
| 10 pct | 26.5 |
| 20 pct | 30.12 |
| 30 pct | 34.67 |
| 40 pct | 37.12 |
| 50 pct | 40.41 |
| 80 pct | 88.65 |
| large | 95.38 |

Grafica




Analisis

La funcion al nunca llegar a un for dentro de un for no se elevara potencialmente entones no se vera mucha demoria ni aumento a la hora de hacerlo. De igual forma la funcion requiere de busquedas en diccionarios y comparaciones para encontrar todos los datos necesarios.