RETO 1

ESTRUCTURA DE DATOS Y ALGORITMOS

Integrantes:

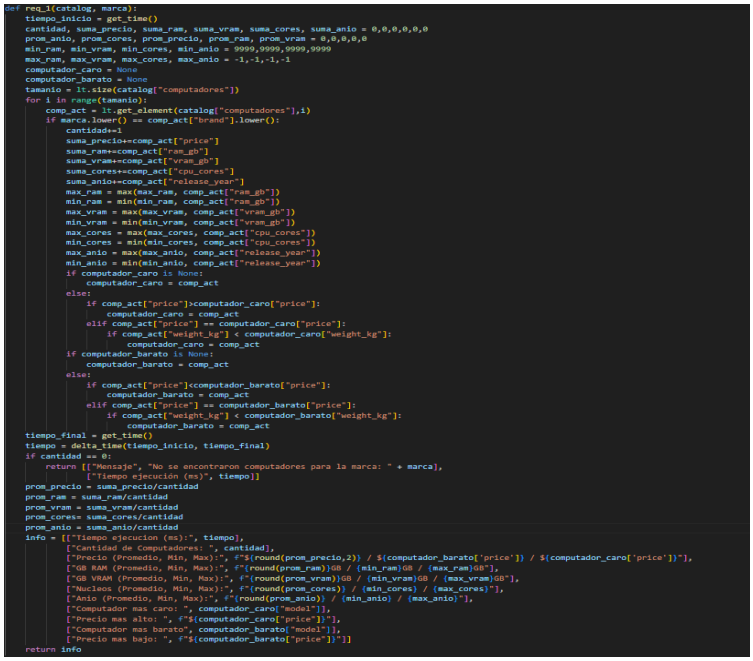
Gabriel Andrés Moya Caravaca - g.moyac@uniandes.edu.co - 202616964

Gerardo Rocha Linares - g.rochal2@uniandes.edu.co - 202513388

Edgar Daniel González Martínez - ed.gonzalezm1@uniandes.edu.co - 202524741

REQUERIMIENTO 1

Implementado por: Gabriel Andrés Moya Caravaca



```
def req_1(catalog, marca):
    tiempo_inicio = get_time()
    cantidad, suma_precio, suma_ram, suma_vram, suma_cores, suma_anio = 0,0,0,0,0,0
    prom_anio, prom_cores, prom_precio, prom_ram, prom_vram = 0,0,0,0,0
    min_ram, min_vram, min_cores, min_anio = 9999,9999,9999,9999
    max_ram, max_vram, max_cores, max_anio = -1,-1,-1,-1
    computador_caro = None
    computador_barato = None
    tamanio = lt.size(catalog["computadores"])
    for i in range(tamanio):
        comp_act = lt.get_element(catalog["computadores"],i)
        if marca.lower() == comp_act["brand"].lower():
            cantidad+=1
            suma_precio+=comp_act["price"]
            suma_ram+=comp_act["ram_gb"]
            suma_vram+=comp_act["vram_gb"]
            suma_cores+=comp_act["cpu_cores"]
            suma_anio+=comp_act["release_year"]
            max_ram = max(max_ram, comp_act["ram_gb"])
            min_ram = min(min_ram, comp_act["ram_gb"])
            max_vram = max(max_vram, comp_act["vram_gb"])
            min_vram = min(min_vram, comp_act["vram_gb"])
            max_cores = max(max_cores, comp_act["cpu_cores"])
            min_cores = min(min_cores, comp_act["cpu_cores"])
            max_anio = max(max_anio, comp_act["release_year"])
            min_anio = min(min_anio, comp_act["release_year"])
            if computador_caro is None:
                computador_caro = comp_act
            else:
                if comp_act["price"]>computador_caro["price"]:
                    computador_caro = comp_act
                elif comp_act["price"] == computador_caro["price"]:
                    if comp_act["weight_kg"] < computador_caro["weight_kg"]:
                        computador_caro = comp_act
            if computador_barato is None:
                computador_barato = comp_act
            else:
                if comp_act["price"]<computador_barato["price"]:
                    computador_barato = comp_act
                elif comp_act["price"] == computador_barato["price"]:
                    if comp_act["weight_kg"] < computador_barato["weight_kg"]:
                        computador_barato = comp_act
    tiempo_final = get_time()
    tiempo = delta_time(tiempo_inicio, tiempo_final)
    if cantidad == 0:
        return [["Mensaje", "No se encontraron computadores para la marca: " + marca],
                ["Tiempo ejecución (ms)", tiempo]]
    prom_precio = suma_precio/cantidad
    prom_ram = suma_ram/cantidad
    prom_vram = suma_vram/cantidad
    prom_cores= suma_cores/cantidad
    prom_anio = suma_anio/cantidad
    info = [["Tiempo ejecucion (ms):", tiempo],
            ["Cantidad de Computadores: ", cantidad],
            ["Precio (Promedio, Min, Max):", f"${round(prom_precio,2)} / ${computador_barato['price']} / ${computador_caro['price']}"],
            ["GB RAM (Promedio, Min, Max):", f"{round(prom_ram)}GB / {min_ram}GB / {max_ram}GB"],
            ["GB VRAM (Promedio, Min, Max):", f"{round(prom_vram)}GB / {min_vram}GB / {max_vram}GB"],
            ["Nucleos (Promedio, Min, Max):", f"{round(prom_cores)} / {min_cores} / {max_cores}"],
            ["Anio (Promedio, Min, Max):", f"{round(prom_anio)} / {min_anio} / {max_anio}"],
            ["Computador mas caro: ", computador_caro["model"]],
            ["Precio mas alto: ", f"${computador_caro['price']}"],
            ["Computador mas barato", computador_barato["model"]],
            ["Precio mas bajo: ", f"${computador_barato['price']}"]]
    return info
```

El requerimiento 1 recibe una marca, busca todos los computadores del catálogo que sean de esa marca y con esos datos deberia retornar: cuánto se demoró en ejecutarse, cuántos computadores encontró y, para precio, RAM, VRAM, núcleos de CPU y año de lanzamiento, calcula el promedio, el mínimo y el máximo. Además, debe decir cuál es el modelo más caro de esa marca y su precio (si hay empate, se queda con el de menor peso) y cuál es el modelo más barato y su precio (si hay empate, también el de menor peso).

COMPLEJIDAD

Este requerimiento recorre una sola vez toda la lista de computadores y va revisando si la marca coincide. Cuando coincide, suma datos y actualiza mínimos/máximos, y también va guardando cuál es el más caro y el más barato. Como ese recorrido es de principio a fin y cada paso hace cosas básicas (sumas, comparaciones, min/max), el tiempo es O(n), donde n es la cantidad de computadores en catalog["computadores"].

En memoria extra es O(1) porque no crea listas nuevas ni estructuras grandes: solo usa variables acumuladoras y dos referencias (caro/barato). Adicionalmente, lo que está fuera del for (divisiones para promedios y armar la respuesta) se hace una sola vez y no cambia la complejidad.

REQUERIMIENTO 2

Implementado por: Gerardo Rocha Linares

```
def req_2(catalog,min,max):
    inicio=get_time()
    menor_precio=None
    mayor_precio=None
    moderno=None
    cumplen=0
    suma_ram=0
    suma_vram=0
    suma_precios=0
    min = float(min)
    max = float(max)
    filtrados=lt.new_list()
    n=lt.size(catalog["computadores"])

    for i in range(n):
        element=lt.get_element(catalog["computadores"],i)
        if element["price"] <= max and element["price"] >= min:
            cumplen+=1
            suma_vram+=element["vram_gb"]
            suma_precios+=element["price"]
            suma_ram+=element["ram_gb"]
            lt.add_last(filtrados,element)

            if moderno is None:
                moderno = element

            else:
                if element["release_year"] > moderno["release_year"]:
                    moderno=element

                elif element["release_year"] == moderno["release_year"] and element["price"] > moderno["price"]:
                    moderno=element

            if mayor_precio is None:
                mayor_precio=element
            else:
                if element["price"] > mayor_precio["price"]:
                    mayor_precio=element

            if menor_precio is None:
                menor_precio=element
            else:
                if element["price"] < menor_precio["price"]:
                    menor_precio=element
    if cumplen == 0:
        final = get_time()
        tiempo = delta_time(inicio, final)

        return [
            ["Mensaje", f"No se encontraron computadores en el rango {min} - {max}"],
            ["Tiempo ejecución (ms)", tiempo]
        ]

    promedio_ram = suma_ram / cumplen
    promedio_vram = suma_vram / cumplen
    promedio_precios = suma_precios / cumplen

    final = get_time()
    tiempo = delta_time(inicio, final)

    info = [
        ["Tiempo ejecución (ms)", tiempo],
        ["Cantidad", cumplen],
        ["Precio promedio", round(promedio_precios, 2)],
        ["RAM promedio", round(promedio_ram)],
        ["VRAM promedio", round(promedio_vram)],
    ]

    if moderno:
        info.append(["Más moderno",
                     f"{moderno['model']} | {moderno['brand']} | "
                     f"{moderno['release_year']} | "
                     f"CPU: {moderno['cpu_brand']} | GPU: {moderno['gpu_brand']} | "
                     f"${moderno['price']}"])

    if mayor_precio:
        info.append(["Más caro",
                     f"{mayor_precio['model']} | {mayor_precio['brand']} | "
                     f"CPU: {mayor_precio['cpu_brand']} | GPU: {mayor_precio['gpu_brand']} | "
                     f"${mayor_precio['price']}"])

    if menor_precio:
        info.append(["Más barato",
                     f"{menor_precio['model']} | {menor_precio['brand']} | "
                     f"CPU: {menor_precio['cpu_brand']} | GPU: {menor_precio['gpu_brand']} | "
                     f"${menor_precio['price']}"])

    return info
```

El requerimiento 2 recibe un precio mínimo y un precio máximo, filtra todos los computadores que estén dentro de ese rango y con los que cumplen calcula un resumen: tiempo de ejecución, cuántos computadores quedaron, y los promedios de RAM, VRAM y precio. Además, debe identificar el computador más moderno del rango (el de mayor release_year; si hay empate en el año, devolver el más costoso) y también reportar el computador de menor precio y el de mayor precio dentro del rango, mostrando para cada uno modelo, marca, año, CPU, GPU y precio.

COMPLEJIDAD

Este requerimiento recorre una sola vez toda la lista catalog["computadores"] y revisa si el precio de cada computador está entre min y max; cuando cumple, acumula RAM, VRAM y precio para luego calcular los promedios, y al mismo tiempo va actualizando cuál es el más moderno del rango por año. si hay empate en el año, Se decidira por el mas caro. Además, se extraera informacion del más barato y el más caro del rango. Por eso el tiempo es O(n), ya que solo hay un for y no existen ciclos anidados. En memoria adicional sí aumenta porque se crea la lista filtrados con todos los computadores que cumplen el rango, lo que implica O(k), siendo k la cantidad filtrada, y en el peor caso O(n). Fuera del for únicamente se calculan los promedios y se construyen los diccionarios de salida, lo cual es O(1). Los if no cambian la complejidad porque el filtro se evalúa n veces, pero el bloque interno solo se ejecuta cuando el computador cumple, es decir k veces, y el resto de if son comparaciones y asignaciones de tiempo constante, por lo que el orden de crecimiento se mantiene en O(n).

REQUERIMIENTO 3

Implementado por: Edgar Daniel González Martínez

El requerimiento 3 recibe un CPU brand y un CPU tier, filtra el catálogo para quedarse solo con los computadores que cumplan esas dos condiciones y, con ese grupo, debe reportar el tiempo de ejecución, cuántos computadores coincidieron y los promedios de precio, RAM, VRAM y número de hilos; además, debe identificar cuál es la marca de GPU más frecuente entre los filtrados y cuál es el año de lanzamiento más frecuente.

```
def req_3(catalog,cpu_brand,cpu_tier):
    """
    Retorna el resultado del requerimiento 3
    """
    # TODO: Modificar el requerimiento 3
    inicio = get_time()
    lista_nueva = lt.new_list()
    suma_precio = 0
    suma_ram = 0
    suma_vram = 0
    suma_hilos = 0
    contador = 0
    tamaño = lt.size(catalog["computadores"])

    for i in range(tamaño):
        computador = lt.get_element(catalog["computadores"], i)
        if computador["cpu_brand"].lower() == cpu_brand.lower() and computador["cpu_tier"].lower() == cpu_tier.lower():
            lt.add_last(lista_nueva, computador)
            suma_precio += computador["price"]
            suma_ram += computador["ram_gb"]
            suma_vram += computador["vram_gb"]
            suma_hilos += computador["cpu_threads"]
            contador += 1

    if contador == 0:
        return [["Mensaje", "No se encontraron computadores para la marca: " + cpu_brand + " y el tier: " + cpu_tier]]

    prom_precio = suma_precio / contador
    prom_ram = suma_ram / contador
    prom_vram = suma_vram / contador
    prom_hilos = suma_hilos / contador

    año_frecuente = 0
    max_frecuencia = 0
    numero_computadores= lt.size(lista_nueva)

    for i in range(numero_computadores):
        computador = lt.get_element(lista_nueva, i)
        año_actual = computador["release_year"]
        contador_año = 0

        for j in range(numero_computadores):
            computador_año = lt.get_element(lista_nueva, j)

            if computador_año["release_year"] == año_actual:
                contador_año += 1

        if contador_año > max_frecuencia:
            max_frecuencia = contador_año
            año_frecuente = año_actual

    gpu_frecuente = 0
    max_frecuencia_gpu = 0

    for i in range(numero_computadores):
        computador = lt.get_element(lista_nueva, i)
        gpu_actual = computador["gpu_brand"]
        contador_gpu = 0

        for j in range(numero_computadores):
            computador_gpu = lt.get_element(lista_nueva, j)
        if computador_gpu["gpu_brand"] == gpu_actual:
            contador_gpu += 1
    if contador_gpu > max_frecuencia_gpu:
        max_frecuencia_gpu = contador_gpu
        gpu_frecuente = gpu_actual

        final = get_time()
        tiempo = final-inicio

    return  (contador,prom_precio,prom_ram,prom_vram,prom_hilos,año_frecuente,gpu_frecuente,tiempo)
```

La complejidad del req_3 está deterinada por los dos bloques que buscan “el año más frecuente” y “la GPU más frecuente”, porque ambos hacen un conteo con ciclos anidados sobre la lista filtrada. Definiendo n como la cantidad total de computadores en catalog["computadores"] y m como la cantidad que pasa el filtro (m = tamaño de lista_nueva), primero hay un recorrido lineal para filtrar y acumular sumas que cuesta O(n). Luego, para hallar año_frecuente, hay un for externo de m y dentro otro for de m, así que ese bloque cuesta O(m^2). Después viene el bloque de gpu_frecuente. En términos de complejidad igual ejecuta un for externo de m y un for interno de m con llamadas a get_element, así que también aporta O(m^2). Por eso, el tiempo total es O(n) y o(m^2), y en el peor caso cuando casi todos cumplen el filtro (m ≈ n) se convierte en O(n^2). En memoria adicional, se crea lista_nueva para guardar los m filtrados, así que el espacio extra es O(m) (peor caso O(n)), mientras que el resto son solo variables y acumuladores O(1).

REQUERIMIENTO 4

Implementado por: Gabriel Andrés Moya Caravaca

```
def req_4(catalog, cpu_brand, gpu_model):
    """
    Retorna el resultado del requerimiento 4
    """
    # TODO: Modificar el requerimiento 4
    tiempo_inicio = get_time()

    cantidad, suma_precio, suma_vram, suma_ram, suma_boost = 0,0,0,0,0
    prom_precio, prom_vram, prom_ram, prob_boost_cpu = 0,0,0,0
    mas_caro1 = None
    mas_caro2 = None
    lista_filtrada = filtrar_por_cpubrand_gpumodel(catalog,gpu_model, cpu_brand)
    actual = lista_filtrada["first"]
    cantidad = sl.size(lista_filtrada)
    if cantidad == 0:
        tiempo_final = get_time()
        tiempo = delta_time(tiempo_inicio, tiempo_final)
        return [["Mensaje", "No se encontraron computadores que cumplan el filtro de cpu: "+ cpu_brand+" y gpu: " + gpu_model],
                ["Tiempo ejecución (ms)", tiempo]]
    while actual != None:
        comp_act = actual["info"]
        suma_precio += comp_act["price"]
        suma_vram += comp_act["vram_gb"]
        suma_ram += comp_act["ram_gb"]
        suma_boost += comp_act["cpu_boost_ghz"]

        if mas_caro1 is None:
            mas_caro1 = comp_act

        elif (comp_act["price"] > mas_caro1["price"] or
              (comp_act["price"] == mas_caro1["price"] and
              comp_act["weight_kg"] < mas_caro1["weight_kg"])):

            mas_caro2 = mas_caro1
            mas_caro1 = comp_act

        elif mas_caro2 is None:

            mas_caro2 = comp_act

        elif (comp_act["price"] > mas_caro2["price"] or
              (comp_act["price"] == mas_caro2["price"] and
              comp_act["weight_kg"] < mas_caro2["weight_kg"])):
            mas_caro2 = comp_act

        actual = actual["next"]
    prom_precio = suma_precio/cantidad
    prom_vram = suma_vram/cantidad
    prom_ram = suma_ram/cantidad
    prob_boost_cpu = suma_boost/cantidad
    tiempo_final = get_time()
    tiempo = delta_time(tiempo_inicio, tiempo_final)

    info = [
        ["Tiempo ejecución (ms)", tiempo],
        ["Cantidad", cantidad],
        ["Precio promedio", round(prom_precio, 2)],
        ["VRAM promedio (GB)", round(prom_vram)],
        ["RAM promedio (GB)", round(prom_ram)],
        ["Boost promedio (GHz)", round(prob_boost_cpu, 2)],
    ]

    if mas_caro1:
        info.append(["Computadora mas costosa 1:",
                     f"{mas_caro1['model']} | {mas_caro1['brand']} | "
                     f"{mas_caro1['release_year']} | "
                     f"{mas_caro1['cpu_model']} | "
                     f"${mas_caro1['price']}"])

    if mas_caro2:
        info.append(["Computadora mas costosa 2:",
                     f"{mas_caro2['model']} | {mas_caro2['brand']} | "
                     f"{mas_caro2['release_year']} | "
                     f"{mas_caro2['cpu_model']} | "
                     f"${mas_caro2['price']}"])

    return info
```

El requerimiento 4 recibe una marca de CPU y un modelo de GPU, recorre el catálogo para filtrar solo los computadores que coinciden con esa combinación y con ese conjunto calcula el tiempo de ejecución, cuántos coincidieron y los promedios de precio, VRAM, RAM y cpu_boost_ghz. Además, debe identificar y mostrar los 2 computadores más costosos dentro de los filtrados, reportando para cada uno modelo, marca, año, CPU model y precio, y si hay empate en el precio se usa el menor peso como criterio para decidir el orden entre empatados.

COMPLEJIDAD

La complejidad temporal del requerimiento es O(n+k), porque primero filtrar_por_cpubrand_gpumodel recorre todo el catálogo para construir la lista_filtrada, lo cual cuesta O(n), y después el while recorre esa lista filtrada completa para acumular promedios y mantener los 2 computadores más costosos, lo cual cuesta O(k), donde k es la cantidad de computadores que cumplen el filtro. Como k puede ser a lo mucho n, en el peor caso la complejidad se simplifica a O(n). La complejidad espacial adicional es O(k), ya que se crea y almacena lista_filtrada con los computadores que cumplen; el resto del requerimiento solo usa variables acumuladoras y dos referencias (mas_caro1 y mas_caro2), que ocupan O(1).

REQUERIMIENTO 5

Implementado por: Gerardo Rocha Linares

```
def req_5(catalog,min,max,resolucion,solicitud):
    """
    Retorna el resultado del requerimiento 5
    """
    # TODO: Modificar el requerimiento 5
    filtrados=lt.new_list()
    inicio=get_time()
    barato=None
    caro=None
    cumplieron=0
    precio_suma=0
    resolucion_suma=0
    gpu_tier_suma=0
    min = int(min)
    max = int(max)
    n=lt.size(catalog["computadores"])
    for i in range(n):
        element=lt.get_element(catalog["computadores"],i)

        if min < element["release_year"] < max and resolucion == element["resolution"]:
            cumplieron+=1
            precio_suma+=element["price"]
            resolucion_suma+=element["display_size_in"]
            gpu_tier_suma+=element["gpu_tier"]
            lt.add_last(filtrados,element)
            if barato is None:
                barato = element
            else:
                if element["price"] < barato["price"]:
                    barato=element
                elif element["price"] == barato["price"]:
                    if element["weight_kg"] < barato["weight_kg"]:
                        barato=element
            if caro is None:
                caro=element
            else:
                if element["price"] > caro["price"]:
                    caro=element
                elif element["price"] == caro["price"]:
                    if element["weight_kg"] < caro["weight_kg"]:
                        caro=element
    if cumplieron > 0:
        precio_promedio=precio_suma/cumplieron
        promedio_resolucion=resolucion_suma/cumplieron
        promedio_gpu_tier=gpu_tier_suma/cumplieron
    else:

        promedio_gpu_tier = promedio_resolucion = precio_promedio =0
    if barato is not None:
        barato_info={"brand":caro["brand"],"model":caro["model"],
                     "price":barato["price"],
                     "display_size_in":barato["display_size_in"],
                     "gpu_tier":barato["gpu_tier"],
                     "display_type":barato["display_type"],
                     "release_year":barato["release_year"],
                     "weight_kg":barato["weight_kg"]}
    else:
        barato_info=None
    if caro is not None:
        caro_info={"brand":caro["brand"],"model":caro["model"],
                     "price":caro["price"],
                     "display_size_in":caro["display_size_in"],
                     "gpu_tier":caro["gpu_tier"],
                     "display_type":caro["display_type"],
                     "release_year":caro["release_year"],
                     "weight_kg":caro["weight_kg"]}
    else:
        caro_info=None
    info = None

    if solicitud.lower() == "barato":
        info=barato_info
    if solicitud.lower() =="caro":
        info=caro_info
    final=get_time()
    tiempo=final-inicio
    resultado = [
        ["Tiempo ejecución (ms)", tiempo],
        ["Filtro de selección", solicitud.upper()],
        ["Cantidad", cumplieron],
        ["Precio promedio", round(precio_promedio, 2)],
        ["GPU tier promedio", round(promedio_gpu_tier, 2)],
        ["Tamaño pantalla promedio", round(promedio_resolucion, 2)],
    ]
    if info:
        resultado.append([
            f"Más {solicitud.lower()}:",
            f"{info['brand']} | "
            f"{info['model']} | "
            f"${info['price']} | "
            f"{info['display_size_in']} in | "
            f"GPU Tier {info['gpu_tier']} | "
            f"Tipo {info['display_type']} | "
            f"Año {info['release_year']} | "
            f"Peso {info['weight_kg']} kg"
        ])
    return resultado
```

El requerimiento 5 recibe un tipo de filtro (BARATO o CARO), una resolución y un rango de años, y con eso recorre el catálogo para quedarse solo con los computadores que tengan esa resolución y estén entre el año mínimo y el año máximo. Con los que cumplen debe reportar el tiempo de ejecución, el filtro usado y cuántos computadores coincidieron, calcular los promedios de precio, tamaño de pantalla y GPU tier, y además escoger el computador que responde a la consulta según el filtro: si es BARATO, el de menor precio; si es CARO, el de mayor precio. Para ese computador elegido se muestran precio, resolución, año, tamaño de pantalla, GPU tier, tipo de display y peso. Si hay empates en precio dentro de la misma resolución, se desempata usando el menor peso. COMPLEJIDAD En este requerimiento, la complejidad temporal es O(n) porque se recorre una sola vez catalog["computadores"] (n elementos) y en cada iteración solo se hacen comparaciones y operaciones básicas de suma, además de actualizar las referencias a barato y caro, todo en tiempo constante; agregar a filtrados con lt.add_last también se asume O(1). La complejidad espacial adicional sí aumenta porque se crea la lista filtrados y se guardan allí todos los computadores que cumplen el filtro. Fuera del ciclo solo se calculan promedios y se arman diccionarios de salida, lo cual es O(1) y no afecta el orden.

REQUERIMIENTO 6

Implementado por: Edgar Daniel Gonzalez Martinez

El requerimiento 6 toma un año inicial y un año final, y utiliza este rango para explorar el catálogo de computadores, filtrando solo

aquellos cuyo año de lanzamiento (release_year) se encuentre dentro del intervalo especificado. Con los computadores seleccionados, el requerimiento debe informar sobre el tiempo de ejecución y cuántos registros cumplen con el rango. Además, agrupa los computadores según el sistema operativo (OS) y para cada grupo, calcula el número de computadores, el total recaudado (suma de precios), el precio y el peso promedio. También determina, para cada sistema operativo, cuál es el computador más caro y el más económico. En términos generales, el requerimiento debe señalar cuál es el sistema operativo más utilizado (el que tiene más computadores en el rango) y cuál genera la mayor recaudación dentro del intervalo de años analizado.

```
def req_6(catalog, año_inicial, año_final):
    """
    Retorna el resultado del requerimiento 6
    """
    tiempo_inicio = get_time()
    lista_filtrada = sl.new_list()
    os_unicos = sl.new_list()
    resultados_por_os = sl.new_list()
    cantidad_total = 0
    os_mas_usado = None
    os_mayor_recaudo = None
    max_cantidad = 0
    max_recaudo = 0
    max_cantidad_recaudo = 0
    cantidad_os = 0
    suma_precio = 0
    suma_peso = 0
    mas_caro = None
    mas_barato = None
    precio = 0
    peso = 0
    info_os = None
    encontrado = False
    os_actual = None

    for i in range(lt.size(catalog["computadores"])):
        comp = lt.get_element(catalog["computadores"], i)
        if año_inicial <= comp["release_year"] <= año_final:
            sl.add_last(lista_filtrada, comp)

    cantidad_total = sl.size(lista_filtrada)

    if cantidad_total == 0:
        tiempo_final = get_time()
        tiempo = delta_time(tiempo_inicio, tiempo_final)
        return [["Mensaje", f"No se encontraron computadores entre {año_inicial} y {año_final}"],
                ["Tiempo ejecución (ms)", tiempo]]

    actual = lista_filtrada["first"]
    while actual is not None:
        os_actual = actual["info"]["os"]
        encontrado = False
        nodo_os = os_unicos["first"]
        while nodo_os is not None:
            if nodo_os["info"] == os_actual:
                encontrado = True
                break
            nodo_os = nodo_os["next"]
        if not encontrado:
            sl.add_last(os_unicos, os_actual)
        actual = actual["next"]


    nodo_os = os_unicos["first"]
    while nodo_os is not None:
        os_actual = nodo_os["info"]
        cantidad_os = 0
        suma_precio = 0
        suma_peso = 0
        mas_caro = None
        mas_barato = None

        actual = lista_filtrada["first"]
        while actual is not None:
            comp = actual["info"]
            if comp["os"] == os_actual:
                cantidad_os += 1
                precio = comp["price"]
                peso = comp["weight_kg"]
                suma_precio += precio
                suma_peso += peso

                if mas_caro is None or precio > mas_caro["price"] or \
                   (precio == mas_caro["price"] and peso < mas_caro["weight_kg"]):
                    mas_caro = comp
                if mas_barato is None or precio < mas_barato["price"] or \
                   (precio == mas_barato["price"] and peso < mas_barato["weight_kg"]):
                    mas_barato = comp
            actual = actual["next"]

        if cantidad_os > max_cantidad:
            max_cantidad = cantidad_os
            os_mas_usado = os_actual
        if suma_precio > max_recaudo:
            max_recaudo = suma_precio
            os_mayor_recaudo = os_actual
            max_cantidad_recaudo = cantidad_os

        info_os = {
            "os": os_actual,
            "cantidad": cantidad_os,
            "recaudo": round(suma_precio, 2),
            "precio_promedio": round(suma_precio / cantidad_os, 2),
            "peso_promedio": round(suma_peso / cantidad_os, 2),
            "mas_caro": mas_caro,
            "mas_barato": mas_barato
        }
        sl.add_last(resultados_por_os, info_os)
        nodo_os = nodo_os["next"]

    tiempo_final = get_time()
    tiempo = delta_time(tiempo_inicio, tiempo_final)
```

```
def req_6(catalog, año_inicial, año_final):

    tiempo_final = get_time()
    tiempo = delta_time(tiempo_inicio, tiempo_final)

    info = [
        ["Tiempo ejecución (ms)", round(tiempo, 2)],
        ["Total computadores en rango", cantidad_total],
        ["OS más usado", os_mas_usado],
        ["  Cantidad de registros", max_cantidad],
        ["  Recaudo total", round(max_recaudo, 2)],
        ["OS mayor recaudo", os_mayor_recaudo],
        ["  Cantidad de registros", max_cantidad_recaudo],
        ["  Recaudo total", round(max_recaudo, 2)],
    ]

    nodo = resultados_por_os["first"]
    while nodo is not None:
        datos = nodo["info"]
        info.append([f"--- OS: {datos['os']} ---", ""])
        info.append(["  Precio promedio", datos["precio_promedio"]])
        info.append(["  Peso promedio (kg)", datos["peso_promedio"]])
        if datos["mas_caro"]:
            c = datos["mas_caro"]
            info.append(["  Más costoso",
                         f"{c['model']} | {c['brand']} | {c['release_year']} | "
                         f"{c['cpu_model']} | {c['gpu_model']} | ${c['price']}"])
        if datos["mas_barato"]:
            b = datos["mas_barato"]
            info.append(["  Más barato",
                         f"{b['model']} | {b['brand']} | {b['release_year']} | "
                         f"{b['cpu_model']} | {b['gpu_model']} | ${b['price']}"])
        nodo = nodo["next"]

    return info
```

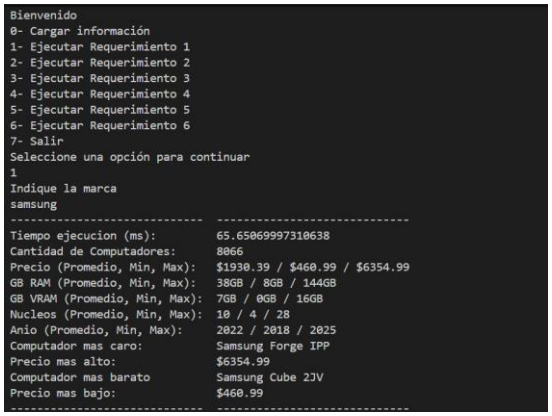
COMPLEJIDAD:

El requerimiento 6 tiene complejidad temporal O(n + k·u) debido a que primero recorre todo el catálogo para filtrar por rango de años (n computadores) y luego, sobre los k computadores filtrados, repite recorridos por cada uno de los u sistemas operativos distintos, donde u cuenta únicamente los sistemas operativos únicos. En el peor caso, si cada computador filtrado tiene un sistema operativo diferente, u = k y el tiempo queda O(n + k^2); y si además k se parece a n (es decir, casi todo el catálogo cae en el rango), entonces k^2 es equivalente a n^2, por lo que la complejidad se puede expresar como O(n^2). En memoria, el requerimiento usa espacio adicional lineal respecto a los computadores filtrados porque almacena la lista filtrada y los resultados agrupados por cada sistema operativo distinto.

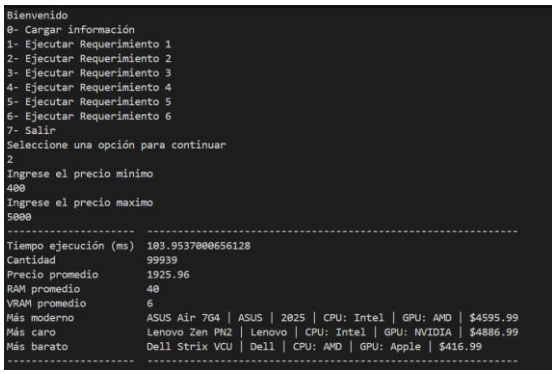
REQUERIMIENTOS INDIVIDUALES

PRUEBAS REQUERIMIENTOS:

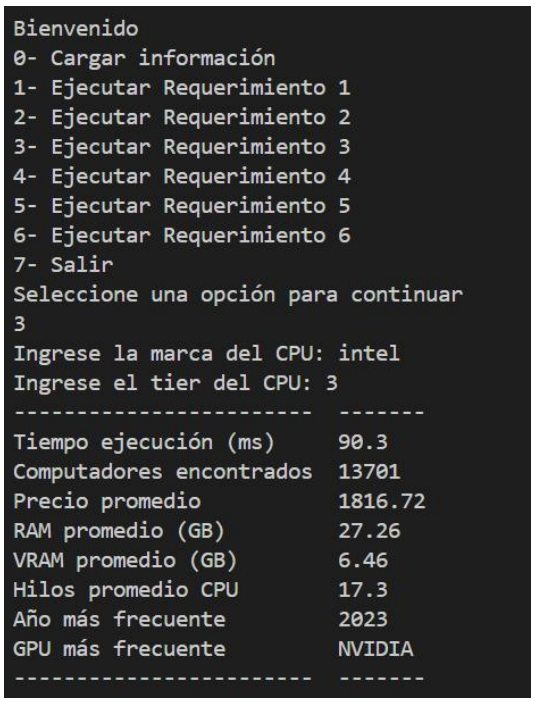
Req 1 Implementado por: Gabriel Andrés Moya Caravaca



Req 2 Implementado por: Gerardo Rocha Linares



Req 3 Implementado por: Edgar Daniel Gonzalez Martinez



REQUERIMIENTOS GRUPALES

PRUEBAS REQUERIMIENTOS:

Req 4 Implementado por: Gabriel Andrés Moya Caravaca

```
Bienvenido
0- Cargar información
1- Ejecutar Requerimiento 1
2- Ejecutar Requerimiento 2
3- Ejecutar Requerimiento 3
4- Ejecutar Requerimiento 4
5- Ejecutar Requerimiento 5
6- Ejecutar Requerimiento 6
7- Salir
Seleccione una opción para continuar
4
Ingrese la marca de cpu
amd
Ingrese el modelo de gpu
rtx 40 80
-------------------------  ------------------------------------------------------------
Tiempo ejecución (ms)      43.36839997768402
Cantidad                   1858
Precio promedio            2198.96
VRAM promedio (GB)         10
RAM promedio (GB)          57
Boost promedio (GHz)       3.71
Computadora mas costosa 1: Dell Pro TA4 | Dell | 2025 | AMD Ryzen 7 5676 | $8091.99
Computadora mas costosa 2: Lenovo Think CHH | Lenovo | 2022 | AMD Ryzen 7 6395 | $6138.99
-------------------------  ------------------------------------------------------------
```

Req 5 Implementado por: Gerardo Rocha Linares

```
2- Ejecutar Requerimiento 2
3- Ejecutar Requerimiento 3
4- Ejecutar Requerimiento 4
5- Ejecutar Requerimiento 5
6- Ejecutar Requerimiento 6
7- Salir
Seleccione una opción para continuar
5
Ingrese el anio minimo
2019
Ingrese el anio maximo
2022
Ingrese la resolucion de pantalla(widthxheight)
1920x1080
Deseea que sea BARATO o CARO
caro
-----------------------  ---------------------------------------------------------------------------------------------
Tiempo ejecución (ms)    53.953400015830994
Filtro de selección      CARO
Cantidad                 9538
Precio promedio          1792.61
GPU tier promedio        2.96
Tamaño pantalla promedio 19.08
Más caro:                Apple | Apple Blade NW3 | $9529.99 | 15.6 in | GPU Tier 5 | Tipo OLED | Año 2021 | Peso 1.1 kg
-----------------------  ---------------------------------------------------------------------------------------------
```

Req 6 Implementado por: Edgar Daniel Gonzalez Martinez

```
6- Ejecutar Requerimiento 6
7- Salir
Seleccione una opción para continuar
6
Ingrese el anio inicial
2022
Ingrese el anio final
2025
--------------------------  ------------------------------------------------------------------------------
Tiempo ejecución (ms)       245.0
Total computadores en rango 67878
OS más usado                Windows
Cantidad de registros       48706
Recaudo total               93319861.94
OS mayor recaudo            Windows
Cantidad de registros       48706
Recaudo total               93319861.94
--- OS: Windows ---
Precio promedio             1915.98
Peso promedio (kg)          4.3
Más costoso                 HP Think 1FC | HP | 2022 | AMD Ryzen 7 7131 | Arc A770 | $9633.99
Más barato                  Dell Zen IL9 | Dell | 2022 | AMD Ryzen 3 3564 | RX 6000 50 | $372.99
--- OS: macOS ---
Precio promedio             2188.35
Peso promedio (kg)          4.25
Más costoso                 Apple Pro XVG | Apple | 2025 | Apple M1 Pro | Apple Integrated | $9772.99
Más barato                  Samsung Cube 2JV | Samsung | 2025 | AMD Ryzen 5 3687 | RX 6000 50 | $460.99
--- OS: Linux ---
Precio promedio             1873.91
Peso promedio (kg)          4.29
Más costoso                 MSI Legion ZSP | MSI | 2023 | Intel i5-13435 | RX 6000 60 | $5479.99
Más barato                  Dell Zen 7JI | Dell | 2022 | Intel i3-11875 | RTX 40 50 | $479.99
--- OS: ChromeOS ---
Precio promedio             1827.0
Peso promedio (kg)          4.28
Más costoso                 Lenovo Pro KMZ | Lenovo | 2025 | Intel i7-10176 | RTX 40 90 | $10984.99
Más barato                  Acer Cube XBM | Acer | 2023 | Intel i5-11703 | Apple Integrated | $593.99
--------------------------  ------------------------------------------------------------------------------
```