


# Parcial Final Estructuras de Datos

CC. 1083864045 - Santiago Alexander Losada Muñoz

CC. 1001370984 - Efraín García Valencia

1. Definir para el sistema mencionado abajo y un conjunto de datos de al menos 200 elementos, cuál sería el método de ordenamiento por el que usted recomendaría ordenar la data para las consultas propuestas.

*Sistemas de gestión de inventario: Ordenar productos en un almacén por código de producto, nombre o cantidad.*

Código de apoyo:  Punto 1 - InventorySort.ipynb

Para un sistema de gestión de inventario que requiere ordenar productos en un almacén según el código de producto, nombre o cantidad, el método de ordenamiento recomendado dependerá de los requisitos específicos y del tipo de consultas que se realicen con mayor frecuencia; a continuación se presenta un análisis detallado para cada uno de los algoritmos considerados, con el fin de encontrar el que mejor se ajuste a las particularidades y necesidades del sistema:

## Quicksort

### Ventajas:

- Eficiente en la práctica y generalmente más rápido que otros algoritmos de ordenamiento.
- Requiere menos memoria adicional, ya que opera in situ (no requiere memoria adicional para almacenar copias temporales de los datos).

### Desventajas:

- No es estable: puede cambiar el orden relativo de elementos iguales.
- Peor caso de complejidad temporal de  $O(n^2)$  si la elección del pivote es mala, aunque en la práctica esto es raro.

### Consideraciones:

- Apropriado para conjuntos de datos grandes y aleatorios.
- Adecuado cuando la memoria es limitada ya que opera in situ.
- Útil cuando se necesita un algoritmo de ordenamiento rápido en promedio.

## Mergesort

### Ventajas:

- Estable: mantiene el orden relativo de elementos iguales.
- Mejor caso, peor caso y caso promedio tienen una complejidad temporal de  $O(n \log n)$ .
- Adecuado para datos enlazados y listas.

### Desventajas:

- Requiere memoria adicional para almacenar las listas temporales durante el proceso de mezcla.

### Consideraciones:

- Apropiado para conjuntos de datos grandes y cuando la estabilidad es esencial.
- Útil cuando la memoria adicional no es un problema.

## Heapsort

### Ventajas:

- In situ: no requiere memoria adicional para almacenar copias temporales de los datos.
- Complejidad temporal de  $O(n \log n)$  en todos los casos.

### Desventajas:

- No es estable.
- No es tan eficiente en la práctica como Quicksort y Mergesort debido a su constante multiplicativa más grande.

### Consideraciones:

- Apropiado cuando se necesita ordenar grandes conjuntos de datos in situ y la estabilidad no es una preocupación.
- Útil cuando la memoria es limitada.

## Resumen

### Velocidad:

- Quicksort tiende a ser más rápido en la práctica, especialmente para conjuntos de datos grandes y aleatorios.
- Mergesort y Heapsort tienen una complejidad temporal más consistente en todos los casos,  $O(n \log n)$ .

### Uso de Memoria:

- Quicksort y Heapsort son in situ y requieren menos memoria adicional.
- Mergesort necesita memoria adicional para almacenar listas temporales.

### Estabilidad:

- Mergesort es estable, lo que significa que conserva el orden relativo de elementos iguales.
- Quicksort y Heapsort no son estables.

## Pruebas

Las pruebas se realizaron con un conjunto de datos de 200 elementos, se probó a ordenar con cada uno de los algoritmos de ordenamiento propuestos y con los diferentes tipos (Nombre, Cantidad y Código), además el tiempo obtenido en milisegundos es el tiempo promedio al ejecutarse 10000 veces.

```
Tiempo de ejecución (Nombre,Quicksort): 13740.344ms  
Tiempo de ejecución (Nombre,Mergesort): 9332.055ms  
Tiempo de ejecución (Nombre,Heapsort): 14129.197ms
```

```
Tiempo de ejecución (Cantidad,Quicksort): 10926.333ms  
Tiempo de ejecución (Cantidad,Mergesort): 9070.015ms  
Tiempo de ejecución (Cantidad,Heapsort): 13667.502ms
```

```
Tiempo de ejecución (Código,Quicksort): 13648.591ms  
Tiempo de ejecución (Código,Mergesort): 7842.547ms  
Tiempo de ejecución (Código,Heapsort): 13716.623ms
```

## Conclusiones

- El algoritmo de MergeSort proporciona una gran rapidez en el ordenamiento de los datos, además, como tiene un tiempo de  $O(n \log n)$  en todos los casos, su rendimiento se mantendrá estable a medida que aumente el número de datos, sin embargo al ser recursivo y necesitar almacenar listas temporales este algoritmo no será adecuado para cantidades limitadas de memoria, pero sí cuando la velocidad sea prioritaria.
- El algoritmo de HeapSort por otro lado hace uso mínimo de la memoria, pero en la práctica es el más lento en todos los casos, además de que su eficiencia sólo empeorará a medida que aumente el número de datos, esto lo hace ideal para los casos en que la memoria sea muy limitada y la velocidad de ordenamiento no sea la prioridad.
- Por último, el algoritmo de QuickSort nos presenta un punto medio entre la velocidad del MergeSort y el bajo uso de memoria del HeapSort, además, a medida que aumente el número de datos, el tiempo promedio de ejecución se acercará más a  $O(n \log n)$  pues habrá menor probabilidad de escoger un mal pivote, por lo que es perfecto para situaciones en las que tengamos grandes cantidades de datos y queramos un buen tiempo de ejecución pero nuestra memoria no sea suficiente para el MergeSort.

Para finalizar, cabe destacar que la elección del algoritmo dependerá de las características específicas del problema, como el tamaño del conjunto de datos, los requisitos de memoria, la necesidad de estabilidad, si la lista de productos es estática o dinámica y de cuánto énfasis se coloca en el rendimiento versus la simplicidad de implementación.

**Importante:** Se tendrá un ordenamiento preestablecido, éste dependerá de cuál será el que tenga mayor número de consultas, en éste caso por decisión nuestra será por orden de código de producto, ya que se asume será el que tenga mayor número de consultas además de ser el de mayor complejidad de realizar.

## 2. Definir para el sistema mencionado abajo y un conjunto de datos de al menos 200 elementos. Definir, justificar y graficar cual seria la mejor estructura jerárquica para darle solución a los problemas planteados

*Control de accesos y seguridad informática: Árboles para representar jerarquías de permisos y roles.*

### **Prefacio**

En este contexto, un árbol de permisos y roles es una estructura jerárquica donde los nodos representan roles o permisos, y las aristas indican las relaciones jerárquicas entre ellos. Cada usuario está asociado a uno o varios roles, y cada rol tiene asignados ciertos permisos.

Por norma general se espera que el número de roles en un sistema **no** sea muy elevado (*Menos de 30 roles y subroles*) y que en un caso extremo no se tengan más de 50 roles, esta consideración será útil para determinar la escalabilidad y eficiencia en la búsqueda de datos para la estructura dada, además, se entiende que en la estructura que define a los usuarios se encuentre una referencia al rol (o los roles) que este posee dentro del sistema (Ej: {UserName, **Role**, Job, OtherData}), este identificador del rol será usado para encontrar los permisos correspondientes a cada usuario y de este modo evitar guardar una lista de permisos individuales para cada uno de los usuarios del sistema.

La estructura a usar dependerá enormemente de los requerimientos específicos del sistema y las reglas de negocio establecidas para su funcionamiento, a continuación se presenta un análisis detallado de las diferentes estructuras de árbol consideradas, teniendo en cuenta las características propias de cada estructura además de las situaciones en las que estas estructuras serían más o menos apropiadas:

### **Árbol binario de roles con permisos**

#### **Descripción**

En esta estructura cada uno de los nodos del árbol representará a uno de los roles presentes en el sistema, estos nodos tendrán dentro de sí un identificador numérico único, relacionado al rol y un apuntador hacia la lista de permisos propia de ese rol guardada en la base de datos, el árbol será construido de modo que los roles se encuentren ordenados del más común al menos común formando una estructura binaria en la cual el rol más común del sistema será la raíz y los roles menos comunes se encontrarán en los últimos niveles del árbol.

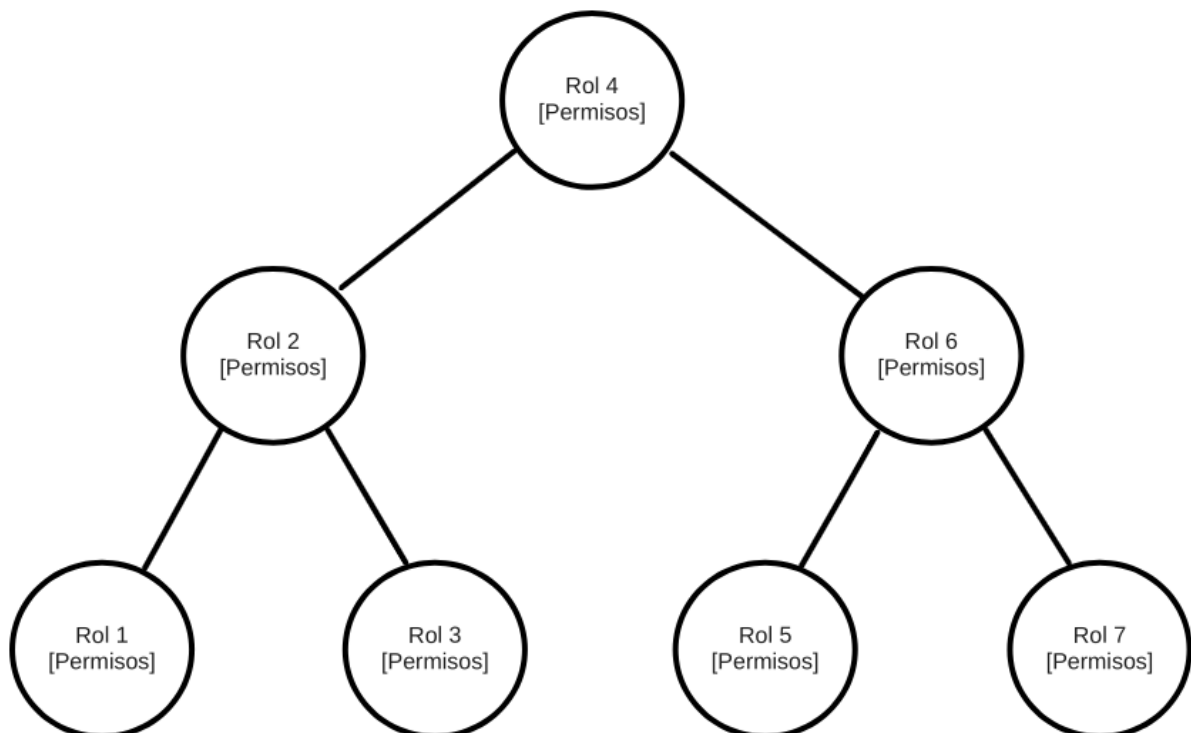
## Ventajas

- Por la forma en la que se encuentra ordenado el árbol los roles más comunes y por tanto los que más se suelen buscar, siempre estarán en la raíz o cerca de la raíz del árbol, lo que asegurará para la mayoría de las consultas una gran eficiencia.
- Guardar las listas con los permisos en la base de datos y solo referenciarlas mediante un apuntador hace que el árbol sea simple, ligero y seguro pues no contendrá como tal la información sensible de los permisos, además, puedo cambiar fácilmente los permisos de un rol directamente desde la base de datos sin necesidad de hacerle cambios al árbol o a sus nodos.
- Al ser un árbol binario la estructura es fácil de implementar y aumentar.

## Desventajas

- Debido a que el árbol tendrá tantos nodos como roles existan en el sistema, la eficiencia en las consultas disminuirá a medida que aumente el número de roles
- Dado que la información de los permisos no está contenida dentro del mismo árbol será necesario hacer constantes llamadas a la base de datos para extraer dicha información durante las consultas.
- Ya que cada rol tiene su propia lista de permisos y es probable que varios de estos permisos sean compartidos entre los roles, se podría redundar en la información presente en las listas.

## Gráfico



## Árbol binario de roles con permisos heredados

### Descripción

En esta estructura cada uno de los nodos del árbol representará a uno de los roles presentes en el sistema, estos nodos tendrán dentro de sí un identificador numérico único, relacionado al rol y un apuntador hacia las listas de permisos en la base de datos tanto del nodo actual como de sus predecesores efectivamente creando una especie de “herencia” descendente para los permisos, el árbol será construido de modo que los roles se encuentren ordenados descendentemente del que tiene la menor cantidad de permisos al que más, formando una estructura binaria en la cual el rol con la menor cantidad de permisos será la raíz y aquellos nodos situados en las hojas del árbol tendrán la mayor cantidad de permisos.

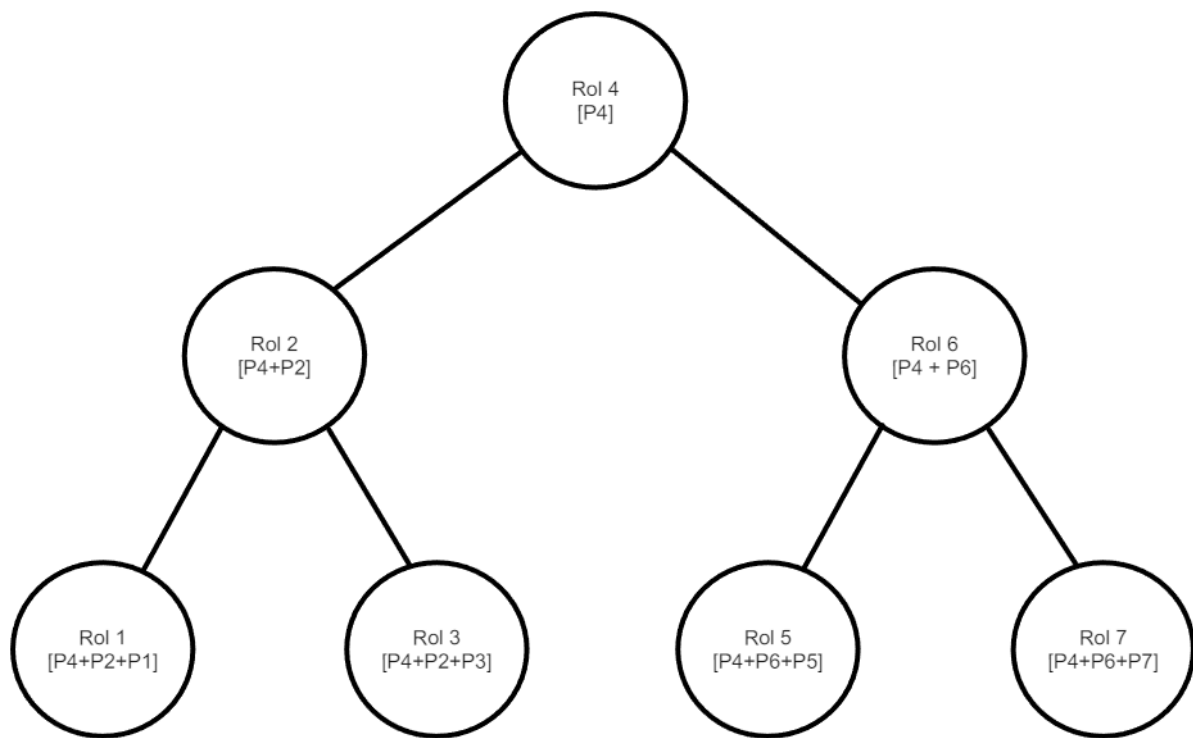
### Ventajas

- Al implementar una forma de heredar los permisos a través del árbol se reduce de manera significativa la redundancia en las listas de permisos.
- Debido a la estructura del árbol y la herencia de permisos se introduce la idea de unos *pseudo-sub-roles* y *roles generalizados* pues los permisos propios de los nodos hijos serán una versión especializada de la de sus padres, y así mismo los roles de los nodos padres tendrán permisos generalizados para con sus nodos hijos.

### Desventajas

- Dada la obligatoriedad de la herencia se extiende la consecuencia lógica de que **TODOS** los roles del sistema compartirán aquellos permisos concedidos en el nodo raíz, lo que hará imposible crear roles que no tengan estos permisos.
- Teniendo en cuenta que se heredan los permisos de todos los predecesores la única forma de crear roles que tengan ciertas combinaciones de permisos es modificando la estructura a un árbol m-way y redundando en los roles.

## Gráfico



Adicionalmente, el modelo *Role based access control* (**RBAC**) se puede tomar en cuenta para éste tipo de casos, es un modelo que se usa como alternativa de **ACL** o lista de control de acceso, puesto que ésta última posee un inconveniente en éste contexto, cuanto mayor es el número de usuarios, más trabajo de mantenimiento conlleva y más errores se pueden producir al asignar las autorizaciones individuales.

Una lista de control de acceso o **ACL** (del inglés, *access control list*) es un concepto de seguridad informática usado para fomentar la separación de privilegios.

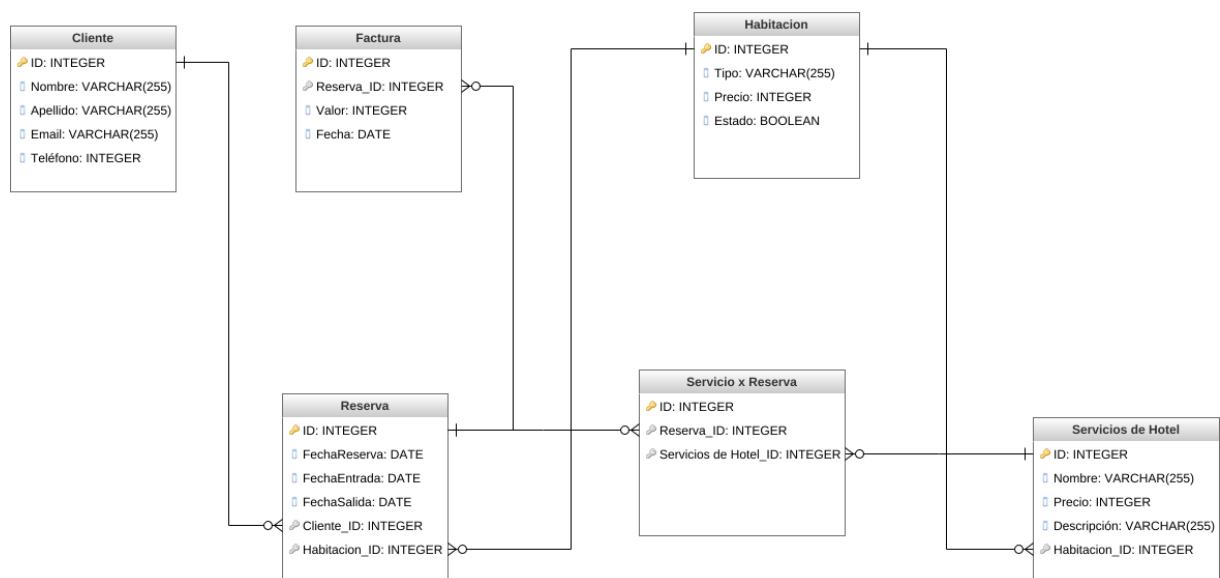
El modelo de seguridad **RBAC** permite con roles y autorizaciones previamente creadas, asignar uno o más roles por usuario, se pueden también asignar autorizaciones de acceso dentro del modelo de rol de forma individual. El objetivo de esta asignación es asegurar que los accesos permitan a los usuarios llevar a cabo todas sus actividades sin necesidad de realizar más ajustes, sin embargo, ésto a su vez impide que se pueda tomar como un árbol para representar su estructura jerárquica, en vez de eso se busca un árbol que pueda acoplarse o asimilar su modelo sin perder de vista las reglas o características que lo definan como árbol.



### 3. Definir un modelo entidad relación que incluya al menos los valores definidos en la parte inferior

*Sistema de Reservas de Hotel: Desarrolla un modelo ER que incluya clientes, habitaciones, reservas, servicios del hotel y facturación.*

[Puedes encontrar el modelo aquí](#)



#### Entidades:

##### 1. Cliente:

- a. Atributos: ClienteID (clave primaria), Nombre, Apellido, Email, Teléfono

##### 2. Habitación:

- a. Atributos: HabitaciónID (clave primaria), Tipo (individual, doble, suite), Precio, Estado (ocupada, disponible)

##### 3. Reserva:

- a. Atributos: ReservaID (clave primaria), Fecha de Reserva, Fecha de Check-in, Fecha de Check-out, ClienteID (clave foránea), HabitaciónID (clave foránea)

##### 4. Servicio del Hotel:

- a. Atributos: ServicioID (clave primaria), Nombre del Servicio, Precio, Descripción

##### 5. Facturación:

- a. Atributos: FacturaID (clave primaria), ReservaID (clave foránea), Total, Fecha de Facturación

## **Relaciones:**

### **6. Relación Cliente-Reserva:**

- a. Un cliente puede realizar varias reservas, pero cada reserva pertenece a un único cliente.
- b. Tipo de Relación: 1 a muchos (1:N)
- c. Claves Extranjeras: ClientelD en Reserva (clave foránea).

### **7. Relación Habitación-Reserva:**

- a. Una habitación puede estar asociada con varias reservas, pero cada reserva está relacionada con una única habitación.
- b. Tipo de Relación: 1 a muchos (1:N)
- c. Claves Extranjeras: HabitaciónID en Reserva (clave foránea).

### **8. Relación Reserva-Facturación:**

- a. Una reserva puede tener varias facturas asociadas, pero cada factura está relacionada con una única reserva.
- b. Tipo de Relación: 1 a muchos (1:N)
- c. Claves Extranjeras: ReservaID en Facturación (clave foránea).

### **9. Relación Reserva-Servicio del Hotel:**

- a. Una reserva puede estar asociada con varios servicios del hotel, y un servicio del hotel puede estar asociado con varias reservas (relación muchos a muchos).
- b. Tipo de Relación: Muchos a Muchos (M:N)
- c. Se introduce una tabla intermedia para representar esta relación:
- d. Tabla Intermedia Servicio x Reserva:
- e. Atributos: ReservaID (clave foránea), ServicioID (clave foránea).

### **10. Servicio x Reserva:**

- a. Atributos: Servicio x ReservaID (clave primaria), ReservaID (clave foránea), ServicioID (clave foránea)
- b. Relaciones:
  - i. Relación con Reserva
  - ii. Relación con Servicio
- c. Justificación:
  - i. Una reserva además de ser de una habitación, puede incluir extra uno o varios servicios, y éstos servicios pueden tener varias reservas asociados a ellos, esto asumiendo que un cliente desee realizar dicha acción y el Hotel quiera que su sistema de reservas pueda permitirlo.

**Justificación Modelo:**

Ya que se nos pide un modelo ER sólo del sistema de reservas de Hotel, se asume que es un único hotel, ya que no se presenta información acerca de una cadena o marca del último, en tal caso, cada hotel tendría su propio sistema de reservas con el cual el administrador rendiría cuentas ante el dueño o superior, además de obviar proveedores, empleados o demás entidades pertenecientes al Hotel en sí, ya que no tienen influencia significativa sobre los elementos pertenecientes al sistema de reservas.