Tema 1: Algoritmos de ordenamiento

Objetivo: El alumno diseñará los métodos más importantes de algoritmos para efectuar ordenamientos en la computadora

1.1 Generalidades

- Ordenar datos es una operación muy común en muchas aplicaciones.
- Ordenar significa reagrupar o reorganizar un conjunto de datos u objetos en una secuencia específica.

1.1 Generalidades

- ¿Para qué realizar ordenamientos?

El ordenamiento es fundamental en diversas aplicaciones computacionales y de la vida cotidiana

Permite manejar grandes volúmenes de información de manera más eficiente.

En muchos contextos de datos facilita la operación de búsqueda.

1.1 Generalidades

- El objetivo de un algoritmo de ordenamiento es reacomodar colecciones de elementos de tal forma que todos sus elementos cumplan con la lógica de una secuencia de acuerdo a una regla predefinida.
- Siempre se sigue alguna de las dos opciones para establecer el criterio de ordenamiento

  Ascendente

  Descendente

1.1 Generalidades

- Saber cual es el mejor algoritmo para alguna situación o conjunto de datos, puede depender de distintos factores, como los datos a ordenar, la memoria, el entorno de SW, etc.
- El principal criterio para medir la eficiencia de un algoritmo es aislar la operación clave
- Las técnicas utilizadas en el diseño de algoritmos se utilizan en algoritmos de ordenamiento.

1.1 Generalidades

- Para cada algoritmo de ordenamiento que se analiza se deben considerar las siguientes cuestiones:

1.- Verificación. ¿El algoritmo siempre devuelve los elementos en orden correcto?

2.- Tiempo de ejecución. Para calcular el tiempo es importante contabilizar comparaciones, intercambios o accesos al arreglo /lista.

3.- Memoria. Se debe considerar la cantidad de memoria extra necesaria para realizar el ordenamiento

1.1 Generalidades

- Existen diversas clasificaciones para los métodos de ordenamiento. La más utilizada es aquella que los jerarquiza de acuerdo a la forma en la que utiliza la memoria en la computadora.

  ✓Ordenamientos internos
  ✓Ordenamientos externos

1.2 Ordenamientos internos

- Existen ordenamientos internos que requieren estructuras de datos auxiliares, los cuales son menos eficientes cuando se busca optimizar memoria.
- Los algoritmos de ordenamiento interno se pueden estudiar de acuerdo a la forma en que realizan operaciones.

1.2.1 Algoritmos por intercambio

1.- Ordenamiento por burbuja

Funciona revisando cada elemento de la lista que va a ser ordenada, junto con el elemento inmediato siguiente, intercambiándose de posición si están en orden inverso

Es necesario revisar varias veces la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

1.2.1 Algoritmos por intercambio

```
ordenamientoBurbuja(lista)
    n = longitud(lista)
    para(i=n hasta i=1, i--)
        para (j=1 hasta j=i-1, j++)
            si(lista [j] > lista[j+1])
              intercambiar(lista[j], lista[j+1])
            fin
        fin
    fin
```

Solución

```
ordenamientoBurbuja(lista)
    n = longitud(lista)
    para(i=n hasta i=1, i--) //ciclo 1
        cambios=falso
        para (j=1 hasta j=i-1, j++)   //ciclo 2
            si(lista [j] > lista[j+1]) //cond 1
              intercambiar(lista[j], lista[j+1])
              cambios=verdadero
            fin //cond 1
        fin  //ciclo2
        if(!cambios)
             break;  //sale de c1
    fin
```

1.2.1 Algoritmos por intercambio

2.- Quicksort

Es el algoritmo más eficiente de los métodos de ordenamiento interno (si se analiza el tiempo).

El algoritmo fue propuesto por Charles Hoare.

Está basado en el paradigma “divide y vencerás” debido a que la idea del algoritmo es dividir el problema en 2 (o más sub problemas) y después combinar la solución de ambos.

1.2.1 Algoritmos por intercambio

La idea del algoritmo consiste en lo siguiente.

1.- Tomar un elemento X del arreglo que se vaya a ordenar.

2.- Ubicar el elemento de tal manera que los elementos que se encuentren a su izquierda sean menores o iguales a el y todos los elementos que se encuentren a su derecha sean mayores.

3.- Se repiten esos pasos en los conjuntos de datos que se encuentran a la izquierda y a la derecha.

1.2.1 Algoritmos por intercambio

```
Quicksort (lista)
     si longitud(lista) > 1
          pivote= lista[x]
          para cada elemento en lista
               si elemento > pivote
                         agregarSubLista1 (elemento)
               en caso contrario
                         agregarSubLista2 (elemento)
          Quicksort (SubLista1)
          Quicksort (SubLista2)
```

1.2.1 Algoritmos por intercambio

Consideraciones para Quicksort

- En el algoritmo no especifica el elemento a elegir como pivote y en qué orden realizar los intercambios, sin embargo no importando el elemento que se elija se llegará al mismo resultado (arreglo ordenado)

1.2.1 Algoritmos por intercambio

- Existen diversas implementaciones para el algoritmo(iterativas, recursivas)
- El algoritmo ofrece una complejidad de n log n para el caso promedio
- El algoritmo ofrece una complejidad de n^2 para el peor caso.
- Sin embargo esto se pude resolver con diferentes técnicas.

1.2.2 Algoritmos por selección

1. **Ordenamiento por selección**

- El ordenamiento por selección es una de las formas más sencillas de ordenar valores.
- Consiste en seleccionar el valor más pequeño del arreglo e intercambiarlo con el primero, de esta forma, el menor de los datos se encontrará al inicio.
- Posteriormente se debe repetir el procedimiento con los siguientes elementos

1.2.2 Algoritmos por selección

```
ordenamientoSeleccion (lista)
    n ← longitud (lista)
    para i ← 1 hasta n − 1
        minimo ← i;
        para j ← i + 1 hasta n
            si lista [j] < lista [minimo]
                minimo ← j
        intercambiar (lista [i] , lista [minimo])
    regresar lista
```

1.2.2 Algoritmos por selección

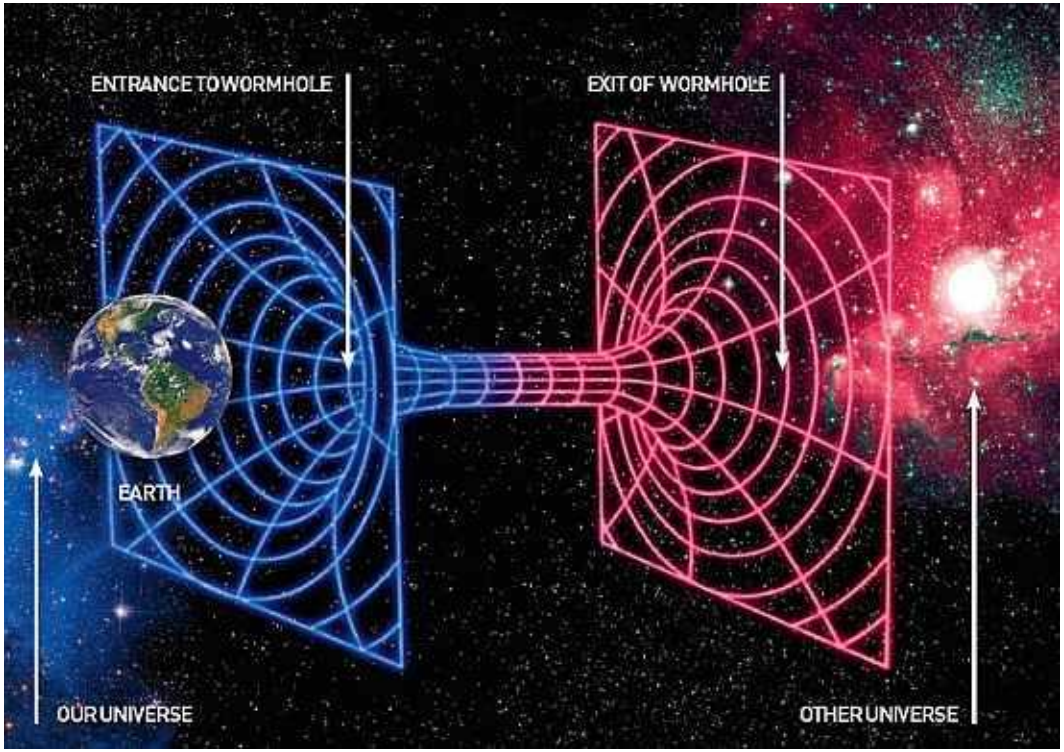
- El algoritmo presenta el inconveniente de no tomar en cuenta el orden de la entrada.
- El proceso de determinar el menor elemento en cada pasada no entrega información alguna sobre la posición del menor elemento para la siguiente pasada.

1.2.2 Algoritmos por selección

- Incluso si recibe un arreglo que ya está ordenado, tomará el mismo tiempo que con un arreglo que no lo está.
- Por otro lado el numero de intercambios que realiza el algoritmo, el número de accesos al arreglo es una función lineal de la longitud del arreglo

1.2.2 Algoritmos por selección

- **2.- HeapSort**



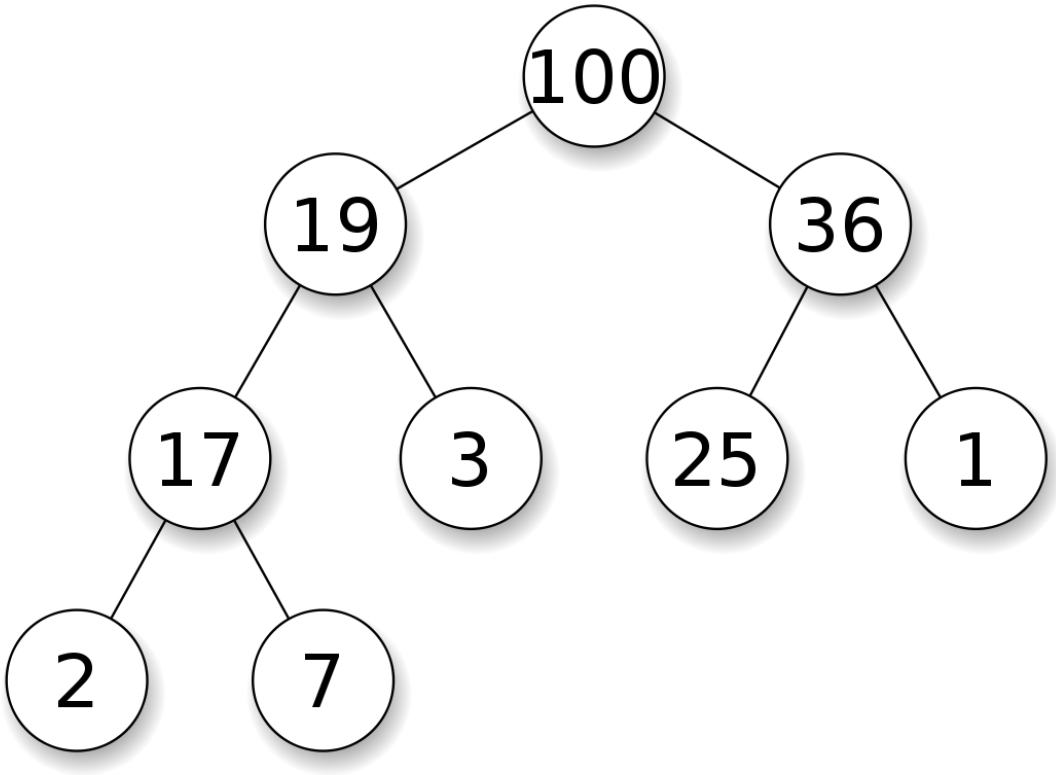
1.2.2 Algoritmos por selección

- Se encuentra entre los algoritmos de ordenamiento más eficientes; en este caso trabaja con estructuras de tipo árbol.
- La idea central de éste algoritmo se basa en dos operaciones:
  1. Construir un “heap”
  2. Eliminar la raíz en forma repetida.

4.?.? Heaps

- ¿Qué es un heap?
- ¿Cómo se construye?
- ¿Cómo se elimina su raíz?
- Heaps vs EDA 1 y EDA2

¿Qué es un heap?



¿Cómo se construye un heap?

- Dos simples pasos:

1. Insertar el nuevo elemento en la primera posición disponible
2. Verificar si el valor es mayor que el “padre”, en tal caso, intercambiar los elementos, en caso contrario finaliza la inserción.

¿Cómo se elimina la raíz de un heap?

- Dos simples pasos:

1. Se reemplaza la raíz con el elemento que ocupa la última posición en el heap
2. Se verifica si el valor de la “nueva” raíz es menor que el valor más grande entre sus hijos. Si eso ocurre se realiza un intercambio, en caso contrario finaliza la ejecución.

1.2.2 Algoritmos de selección

Heapsort

- El algoritmo funciona convirtiendo la estructura a ordenar en un heap.
- Una vez convertido en Heap, en cada iteración se elimina la raíz y se verifica que el resto de la estructura conserve la integridad de Heap
- Se estima que el algoritmo puede ser más lento en su ejecución que Quicksort, sin embargo ofrece una complejidad de O(n log n) para todos los casos

1.2.3 Métodos por inserción

1.- Inserción directa.

En éste tipo de ordenamiento, se considera un elemento en cada caso, insertándolo en el lugar apropiado entre aquellos que ya se encuentran ordenados.

Los elementos que se encuentran a la izquierda del índice actual se encuentran ordenados pero no en su posición final.

1.2.3 Métodos por inserción

```
ordenamientoInsercion(lista)
    n ← longitud(lista)
    para i ← 1 hasta n
        indice ← lista[i]
        j ← i − 1
        mientras j > 0 y lista[j] > indice
            lista[j + 1] ← lista[j]
            j ← j − 1
        lista[j + 1] ← indice
    regresar lista
```

1.2.3 Métodos por inserción

- En el mejor caso, este algoritmo ofrece una complejidad de n ya que el número de intercambios se reduce y solo se realizan comparaciones.
- En caso promedio y peor caso el algoritmo ofrece una complejidad de n^2

1.2.3 Métodos por inserción

2. Inserción Binaria

Consiste en una mejora del algoritmo anterior, dicha mejora radica en que en lugar de comparar con todos los elementos a la izquierda del dato a evaluar, se realiza una búsqueda binaria para encontrar el lugar de inserción adecuado

1.2.5 Método por intercalación

1.- MergeSort

El ordenamiento por intercalación inventado en 1945 por J. von Neumann es otro método basado en el paradigma divide y vencerás.

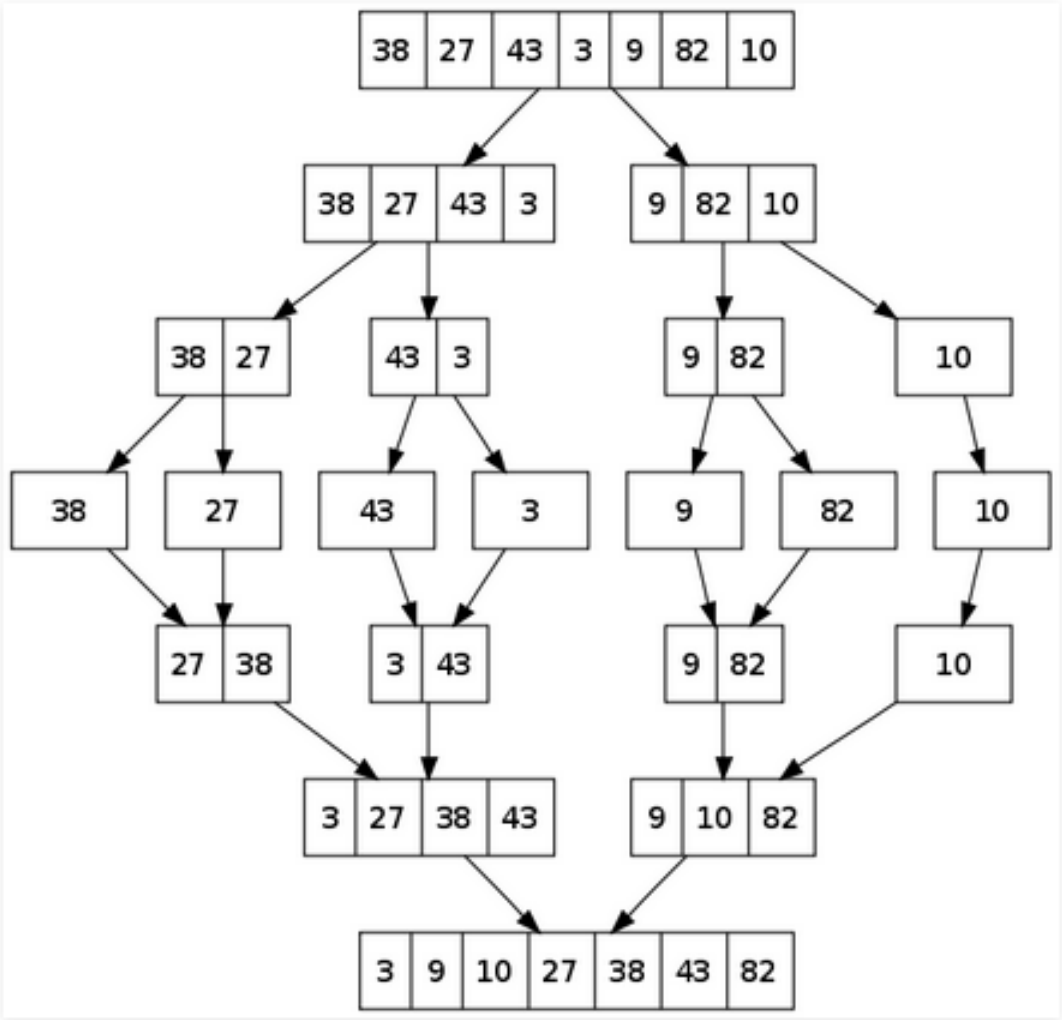
La idea básica es combinar dos listas que ya han sido ordenadas para obtener una lista ordenada mas grande.

1.2.5 Método por intercalación

El proceso consiste en lo siguiente

- ✓ Dividir la lista de n elementos en dos sub-listas de n/2 elementos cada una
- ✓ Ordenar las dos sub-listas utilizando mergesort
- ✓ Combinar (intercalar) las dos sub-listas ordenadas para obtener la lista ordenada final.

Ejemplo


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

1.2.5 Método por intercalación

- El mayor atractivo del ordenamiento por mergesort es que garantiza ordenar cualquier lista de tamaño n en tiempo proporcional a n log n.
- Su desventaja es que necesita espacio extra, que será proporcional al tamaño de la lista a ordenar

1.2.4 Métodos por distribución

1.- Ordenamiento por conteo

En este algoritmo se asume que cada uno de los n elementos de la entrada se encuentra en un rango de [0,k] para algún entero k.

La idea básica del ordenamiento por conteo es determinar, para cada elemento x, el número de elementos menores a x.

Ésta información puede ser utilizada para colocar a x en su posición en la lista de salida

1.2.4 Métodos por distribución

- El algoritmo requiere de 2 arreglos adicionales al arreglo que se está ordenando, uno para hacer “la cuenta” de los elementos y otro para entregar la salida ordenada.
- La complejidad del algoritmo es (n + k)
- El compromiso “espacio-tiempo” se ve afectado ya que se resuelve en una menor cantidad de tiempo pero se utiliza una mayor cantidad de memoria la cual puede ser considerable para arreglos muy grandes.

1.2.4 Métodos por distribución

2.- Ordenamiento Radix

Este algoritmo está basado en los valores absolutos de los dígitos de los números que son ordenados.

Para cada digito de las cifras a ordenar se efectúan los siguientes pasos

a) Se ordenan los números de acuerdo al dígito de en la posición menos significativa
   - Para ese ordenamiento se utilizan colas auxiliares dependiendo de cada uno de esos dígitos

1.2.4 Métodos por distribución

b) Se repite el proceso para la segunda posición
c) Se continúa hasta llegar a la posición más significativa

Complejidad Algoritmos de Ordenamiento

Algorithm	Time Complexity		
	Best	Average	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))
Bubble Sort	O(n)	O(n^2)	O(n^2)
Insertion Sort	O(n)	O(n^2)	O(n^2)
Select Sort	O(n^2)	O(n^2)	O(n^2)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)
Radix Sort	O(nk)	O(nk)	O(nk)

1.3 Ordenamientos externos

- En muchas aplicaciones de ordenamiento es necesario procesar archivos grandes para almacenarlos en memoria principal.
- Los métodos que resuelven este tipo de problemas son métodos de ordenamiento externo.

1.3 Ordenamientos externos

- La mayoría de los métodos externos utiliza la siguiente estrategia general.
  - ✓Hacer una primera pasada sobre el archivo a ordenar, dividiéndolo en bloques de tamaño manipulable en memoria principal
  - ✓Ordenar cada bloque con algún método interno
  - ✓Mezclar los bloques ordenados realizando varias pasadas sobre el archivo.

1.3 Ordenamientos externos

- Dado que el costo principal de los ordenamientos externos está dado por el tiempo de acceso al dispositivo, los algoritmos buscan reducir el número de pasadas sobre el archivo.
- Para estos algoritmos se define
  - número máximo de valores que se pueden llevar a memoria principal
  - número total de llaves a ordenar
  - la cantidad de archivos utilizados en los métodos.

1.3.1 Polifase

- Consiste en aplicar una estrategia de mezclar hasta vaciar el archivo , utilizando archivos auxiliares para almacenar el resultado parcial. Durante la ejecución, el archivo de entrada y alguno de salida intercambian los datos para construir el archivo ordenado.

1.3.1 Polifase

- Fase 1. Mientras existan datos en la entrada
  1. Leer m llaves
  2. Ordenarlas por algún método interno
  3. Colocar las llaves en un archivo F auxiliar (por bloques)
  4. Colocar las siguientes m llaves en otro archivo auxiliar
  5. Para las siguientes m llaves se vuelve a utilizar el primer archivo en un segundo bloque

1.3.1 Polifase

- Fase 2. Mientras los archivos auxiliares tengan datos
  1. Intercalar el primer bloque del archivo auxiliar 1 con el primer bloque del archivo auxiliar 2 y dejar el resultado en el archivo original.
  2. Intercalar el siguiente bloque de cada archivo y dejar el resultado en un tercer archivo auxiliar.
  3. Repetir los pasos 1 y 2 hasta que no haya mas claves por procesar.

1.3.2 Mezcla directa

- Es probablemente el método de ordenamiento externo más utilizado por su facilidad en la implementación.
- La idea central consiste en la realización sucesiva de una partición y una mezcla que produce secuencias ordenadas de las claves del archivo.

1.3.2 Método por mezcla directa

- En la primera pasada la partición es de longitud 1 y la mezcla produce secuencias ordenadas de longitud 2.
- En la segunda pasada, la partición es de longitud 2 y la mezcla produce secuencias de longitud 4.
- Éste proceso se repite hasta ordenar el archivo original

1.3.3 Método por mezcla equilibrada

- Es una optimización del método de mezcla directa
- Consiste en realizar las particiones tomando secuencias ordenadas de máxima longitud en lugar de secuencias de tamaño fijo.
- Se realiza la mezcla entre las secuencias en forma alternada sobre dos archivos.

1.3.4 Método por distribución

- Este método externo está basado en el método radix.
- La diferencia fundamental radica que en el método radix interno se utilizan colas para cada elemento de la cadena {0,1,2,3…} y extrapolando lo anterior, en este método se utiliza un archivo para cada uno de los símbolos analizados en las cadenas.