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.
- La clave para analizar la eficiencia de un algoritmo es identificar 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.

  2.- Tiempo de ejecución.

  3.- Memoria.

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.
- Se clasifican de acuerdo a la forma en que realizan operaciones.

1.2.1 Algoritmos por intercambio

1.- Ordenamiento de la 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

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.

https://en.wikipedia.org/wiki/Tony_Hoare

1.2.1 Algoritmos por intercambio

La idea del algoritmo consiste en lo siguiente.

1.- Seleccionar un elemento.

2.- Ubicarlo de tal manera que, todo lo que se encuentre a su izquierda sea menor o igual, y todo lo que se encuentre a su derecha sea mayor.

3.- Se repiten esos pasos en los conjuntos de datos que se generan 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

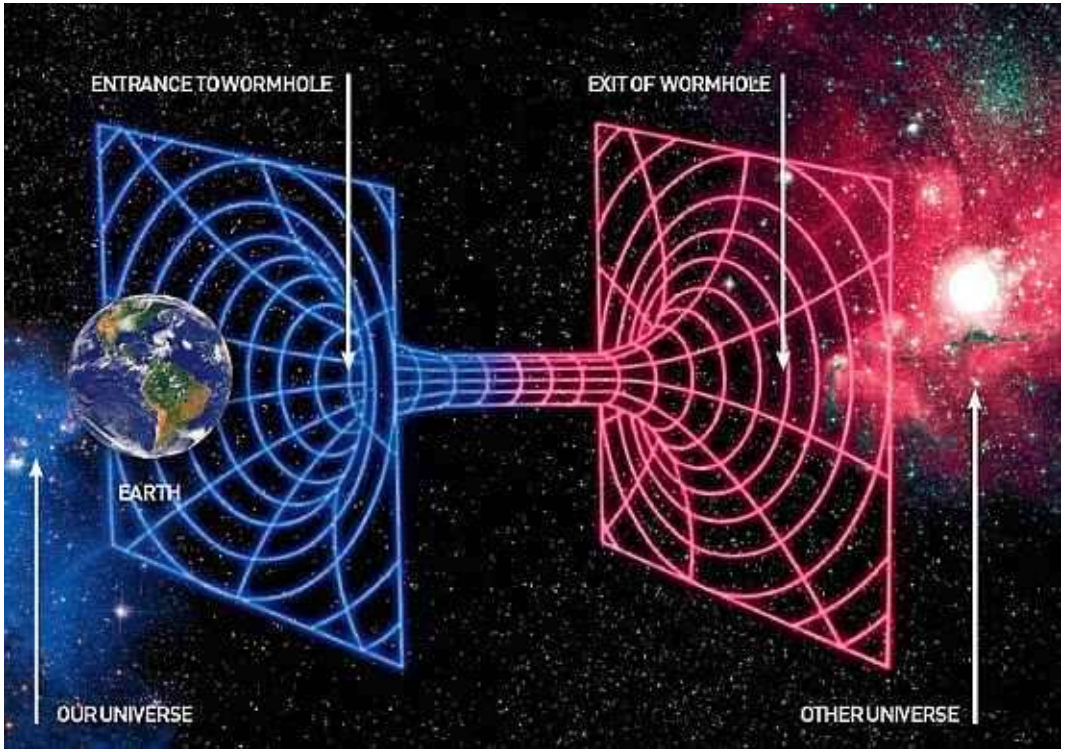
- El algoritmo presenta el inconveniente de no tomar en cuenta el orden de la entrada.
- El proceso de determinar el menor elemento en cada pasada no entrega información alguna sobre la posición del menor elemento para la siguiente pasada.

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 número de intercambios en el arreglo es una función lineal de la longitud del arreglo

1.2.2 Algoritmos por selección

- **2.- HeapSort**



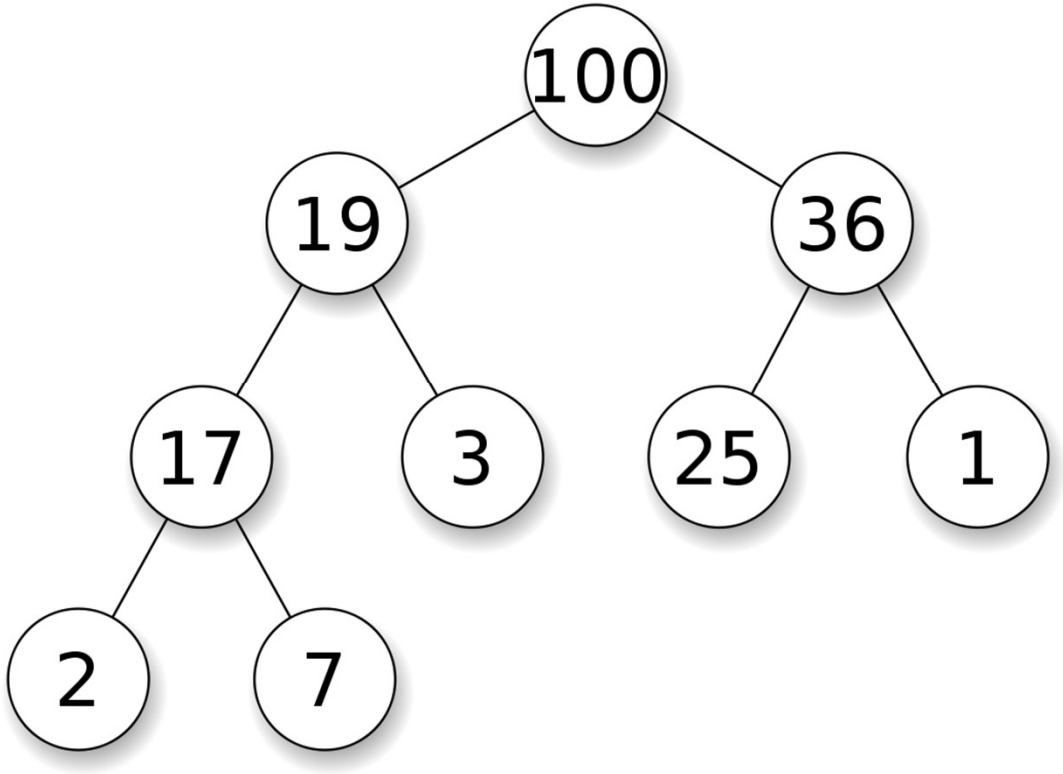
1.2.2 Algoritmos por selección

- 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.
- En cada iteración se elimina la raíz y se verifica que el resto de la estructura conserve la integridad de Heap
- Ofrece una complejidad de O(n log n) para todos los casos.