ORDENAMIENTO

- Introducción.
- Métodos simples de ordenamiento.
- Introducción a métodos mas complejos.
- Comparativas.

INTRODUCCIÓN

- Ordenar: reorganizar un conjunto de elementos considerando ciertas condiciones o relación de orden entre ellos.
- Podemos ordenar los elementos o los punteros que los apuntan (por ejemplo array de punteros)
- Los mejores algoritmos requieren de n Log_2 n operaciones para ordenar n elementos.
- Dos de los mejores algoritmos son: Quicksort y Heapsort.

Tener en cuenta (criterios)

- EFICACIA
  - Número de pasos
  - Número de Comparaciones (saltos)
  - Número de movimientos de elemento (y tamaño)
- SENCILLEZ
  - Para editarlo
  - Para modificarlo

También tener en cuenta otros aspectos como tamaño, numero de procesadores, etc.

SIEMPRE DEPENDERA DEL USO !!!

Comentarios

- Para N pequeñas, usar pequeños algoritmos de simple implementación no trae perjuicios importante.
- Los algoritmos de ordenamiento complementaran a otros como ser de búsqueda.
- Como en otros algoritmos, hay toda una TEORÍA MATEMÁTICA que analiza las ventajas y desventajas de cada método

CONCEPTOS TEÓRICOS

- Cota superior asintótica (o “Notación O Grande”)
  - Es una función que sirve de cota superior de otra función cuando el argumento tiende a infinito.
  - Suele usarse la notación de Landau O(g(x))
  - Representado como: $f(x) \in O(g(x))$ (comúnmente aparece como f(x)=O(g(x)) pero esto es un error porque se trata de un conjunto).

Clasificación

Según

- Lugar de ordenamiento (internos y externos)
- Complejidad computacional (comparaciones)
- Complejidad computacional (intercambios)
- Uso de memoria (soporte temporal)
- Recursividad (recursivo o no recursivo)
- Estabilidad (mantiene relación anterior)
- Adaptabilidad (respuesta según un orden previo)
- Método (distintas variantes)

Tipos

- Ordenamiento interno

  Se lleva a cabo completamente en memoria principal

- Ordenamiento externo

  No cabe toda la información en memoria principal y es necesario ocupar memoria secundaria.

Complejidad (comparaciones)

- Se usa la nomenclatura “O GRANDE”
  - O(1): complejidad constante (siempre lleva el mismo tiempo).
  - O(n^2): complejidad cuadrática (el tiempo depende del cuadrado del número de elementos).
  - También se lo suele analizar con tres valores:mejor caso, peor caso y promedio.
- El número de comparaciones suele ser un valor critico ya que vienen acompañadas de la posibilidad de salto lo cual incrementa mucho el tiempo de proceso.

Complejidad (intercambios)

- El intercambio suele venir acompañado con acceso a distintos medios de almacenamiento lo que agrega tiempos “muertos” al procedimiento.

Uso de memoria

- Algunos algoritmos solo usan la memoria para almacenar los datos a ser ordenados. Esto es O(1)
- Otros necesitan almacenamiento temporal usando memoria adicional para ello.

OTROS

- Recursividad
  - Si utilizan o no algoritmos recursivos
- Estabilidad
  - Si mantienen el orden preestablecido de otras columnas.
- Adaptabilidad
  - Si cambia o no su rendimiento según que tan ordenados están inicialmente.
- Método
  - Todo método puede tener variantes

Métodos simples de ordenamiento

- Inicialmente veremos tres métodos básicos que por su sencillez y practicidad son perfectamente usados para ordenar pequeños conjuntos.
  - Ordenamiento de BURBUJA (Bubble sort)
  - Ordenamiento por SELECCIÓN (Selection sort)
  - Ordenamiento por INSERCIÓN (Insertion sort)

BURBUJA

- Compara cada elemento con el contiguo y los conmuta o no de posición según sea necesario o no para obtener el orden deseado.
- En cada pasada se tiene mínimo un nuevo elemento en su posición definitiva.
- Complejidad: $O(n^2)$
- Mejor/Peor/Promedio: $O(n)/O(n^2)/O(n^2)$
- Memoria:O(1)

ALGORITMO

```
void orden_burbuja(int datos[],int elementos)
{
  int superior = elementos, cambios, i, aux;
  do {
    cambios = FALSE;    //#define FALSE 0
    superior--;
    for(i=0; i<superior; i++)
    {
      if(datos[i]>datos[i+1])
      {
        aux = datos[i];
        datos[i] = datos[i+1];
        datos[i+1] = aux;
        cambios = TRUE;       //#define TRUE 1
      }
    }
  }while(cambios);
return;
}
```

SELECCIÓN

- Realiza una búsqueda por toda la lista del mínimo valor (o máximo) y lo intercambia con el primer elemento, repite pero empezando desde el próximo elemento.
- En cada pasada se tiene mínimamente un nuevo valor en su posición definitiva.
- Complejidad: $O(n^2)$
- Mejor/Peor/Promedio: $O(n^2)/O(n^2)/O(n^2)$
- Memoria:O(1)

ALGORITMO

```
void orden_seleccion(int datos[],int elementos)
{
  int i,j,menor,aux;

  for(i=0;i<elementos;i++)
  {
    menor=i;
    for(j=i+1;j<elementos;j++)
    {
      if(datos[j]<datos[menor])
    menor=j;
    }
    aux=datos[i];
    datos[i]=datos[menor];
    datos[menor]=aux;
  }
  return;
}
```

INSERCIÓN

- Se toma el primer elemento como un conjunto ordenado de un elemento y se van “insertando” de a uno los otros elementos en su posición ordenada relativa al actual conjunto.
- El conjunto siempre esta ordenado y se le insertan nuevos elementos.
- Complejidad: $O(n^2)$
- Mejor/Peor/Promedio: $O(n)/O(n^2)/O(n^2)$
- Memoria:O(1)

ALGORITMO

```
void orden_insercion(int datos[],int elementos)
{
   int i,j,menor,aux;
   for(i=1;i<elementos;i++)
  {
     aux = datos[i];
     for(j=i-1;(datos[j]>aux)&&(j>=0);j--)
     {
       datos[j+1]=datos[j];        // subo elemento un nivel
     }
     datos[j+1]=aux;    // inserto elemento
  }
  return;
}
```

Introducción a métodos mas avanzados

- Comentaremos un par de métodos de ordenamiento mas avanzados.
- Siempre esta la posibilidad de que se creen nuevos métodos.
- Veremos:
  - Shell sort
  - Quick sort (ordenamiento rápido)

SHELL SORT

- Es una generalización del ordenamiento por inserción,con dos observaciones:

  1) Inserción es eficiente si la entrada está "casi ordenada".

  2) Inserción es ineficiente, en general, porque mueve los valores sólo una posición cada vez.

El algoritmo Shell sort mejora el ordenamiento por inserción comparando elementos separados por un espacio de varias posiciones, permitiendo "pasos más grandes" hacia su posición esperada. Estos saltos se hacen de tamaño cada vez más pequeños. El último paso es simplemente un ordenamiento por inserción básico, pero ya se garantizo que los datos del vector están casi ordenados.

Características

- Complejidad:
  - $O(n^2)$ u $O(n \log^2 n)$ (modificación de V. Pratt)
- Mejor/Peor/Promedio[1]:
  - $O(n)/O(n \log^2 n)/O(n \log^2 n)$
- Memoria:
  - O(1)

1) Tambien puede ser $n^{3/2}$

QUICK SORT

- Basado en la técnica de divide y vencerás.
- Pasos:
  - Elegir un elemento (pivote)
  - Reubicar los otros elementos a partir del pivote según orden.
  - Repetir este proceso recursivamente en los dos grupos que quedaron.

Características

- Complejidad:
  - O(n log n) (peor caso $O(n^2)$)
- Mejor/Peor/Promedio:
  - $O(n \log n)/O(n^2)/O(n \log n)$

El mejor caso es cuando el pivote cae en la mitad y el peor cuando cae en un extremo.

- Memoria:
  - O(1)