Instituto Politécnico Nacional

Escuela Superior de Cómputo

Análisis de algoritmos

Practica 01 : Pruebas a posteriori (Algoritmos de Ordenamiento)

M. en C. Edgardo Adrián Franco Martínez
http://www.eafranco.com
edfrancom@ipn.mx
@edfrancom edgardoadrianfrancom

Contenido

- Definición del problema
- Actividades
- Observaciones
- Reporte de práctica
- Rubrica de evaluación del reporte
- Entrega vía Web
- Fechas de Entrega

Definición del problema

- Con base en el archivo de entrada proporcionado que tiene **10,000,000 números diferentes**; ordenarlo bajo **los siguientes métodos de ordenamiento** y **comparar experimentalmente las complejidades** de estos.
  - Burbuja *(Bubble Sort)*
    - Burbuja Simple
    - Burbuja Optimizada
  - Inserción *(Insertion Sort)*
  - Selección *(Selection Sort)*
  - Shell *(Shell Sort)*
  - Ordenamiento con árbol binario de búsqueda *(Tree Sort)*

Ordenamiento Burbuja

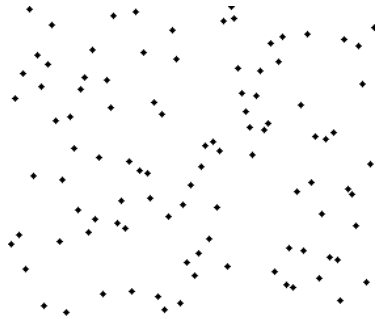
6 5 3 1 8 7 2 4

Burbuja Simple

- Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.
  - El método de la burbuja es uno de los mas simples, es tan fácil como comparar todos los elementos de una lista contra todos, si se cumple que uno es mayor o menor a otro, entonces los intercambia de posición.
  - Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el método del intercambio directo.

Burbuja Simple

6 5 3 1 8 7 2 4



```
Procedimiento BurbujaSimple(A,n)
        para i=0 hasta n-2 hacer
                para j=0 hasta (n-2)-i hacer
                        si (A[j]>A[j+1]) entonces
                                aux = A[j]
                                A[j] = A[j+1]
                                A[j+1] = aux
                        fin si
                fin para
        fin para
fin Procedimiento
```

El arreglo A indexa desde 0 hasta n-1 -- A[0,1,…, n-1]

Una versión de Burbuja Optimizada

- Como al final de cada iteración el elemento mayor queda situado en su posición, ya no es necesario volverlo a comparar con ningún otro número, reduciendo así el número de comparaciones por iteración, además pude existir la posibilidad que realizar iteraciones de más si el arreglo ya fue ordenado totalmente.

```
Procedimiento BurbujaOptimizada(A,n)
        cambios = "No"
        i=0
        Mientras i< n-1 && cambios != "No" hacer
                cambios = "No"
                Para j=0 hasta (n-2)-i hacer
                        Si(A[i] < A[j]) hacer
                                aux = A[j]
                                A[j] = A[i]
                                A[i] = aux
                                cambios = "Si"
                        FinSi
                FinPara
                 i= i+1
        FinMientras
fin Procedimiento
```

El arreglo A indexa desde 0 hasta n-1 -- A[0,1,…, n-1]

Ordenamiento por inserción

6 5 3 1 8 7 2 4

- Es una manera muy natural de ordenar para un ser humano, y puede usarse fácilmente para ordenar un mazo de cartas numeradas en forma arbitraria.

- Inicialmente se tiene un solo elemento, que obviamente es un conjunto ordenado. Después, cuando hay *k* elementos ordenados de menor a mayor, se toma el elemento *k+1* y se compara con todos los elementos ya ordenados, deteniéndose cuando se encuentra un elemento menor (todos los elementos mayores han sido desplazados una posición a la derecha) o cuando ya no se encuentran elementos (todos los elementos fueron desplazados y este es el más pequeño). En este punto se ***inserta*** el elemento ***k+1*** debiendo desplazarse los demás elementos.


```
Procedimiento Insercion(A,n)
{
        para i=0 hasta n-1 hacer
                j=i
                temp=A[i]
                mientras(j>0) && (temp<A[j-1]) hacer
                        A[j]=A[j-1]
                        j--
                fin mientras
                A[j]=temp
        fin para
fin Procedimiento
```

6 5 3 1 8 7 2 4

El arreglo A indexa desde 0 hasta n-1 -- A[0,1,…, n-1]

Ordenamiento por selección

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

- Se basa en buscar el mínimo elemento de la lista e intercambiarlo con el primero, después busca el siguiente mínimo en el resto de la lista y lo intercambia con el segundo, y así sucesivamente.

- **Algoritmo**
  - Buscar el mínimo elemento entre una posición i y el final de la lista Intercambiar el mínimo con el elemento de la posición i.

```
Procedimiento Seleccion(A,n)
        para k=0 hasta n-2 hacer
                p=k
                para i=k+1 hasta n-1 hacer
                        si A[i]<A[p] entonces
                                p=i
                        fin si
                fin para
                temp = A[p]
                A[p] = A[k]
                A[k] = temp
        fin para
fin Procedimiento
```

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

El arreglo A indexa desde 0 hasta n-1 -- A[0,1,…, n-1]

Ordenamiento Shell

- El Shell es una generalización del ordenamiento por inserción, teniendo en cuenta dos observaciones:
  1. El ordenamiento por inserción es eficiente si la entrada está "casi ordenada".
  2. El ordenamiento por inserción es ineficiente, en general, porque mueve los valores sólo una posición cada vez.

- El algoritmo Shell mejora el ordenamiento por inserción comparando elementos separados por un espacio de varias posiciones. Esto permite que un elemento haga "pasos más grandes" hacia su posición esperada. Los pasos múltiples sobre los datos se hacen con tamaños de espacio cada vez más pequeños. El último paso del ordenamiento Shell es un simple ordenamiento por inserción, pero para entonces, ya está garantizado que los datos del vector están casi ordenados.

- Shell propone que se haga sobre el arreglo una serie de ordenaciones basadas en la inserción directa, pero dividiendo el arreglo original en varios sub-arreglos tales que cada elemento esté separado k elementos del anterior (a esta separación a menudo se le llama ***salto*** o ***gap***)

- Se debe empezar con k=n/2, siendo n el número de elementos del arreglo, y utilizando siempre la división entera (**TRUNC**)

- Después iremos variando k haciéndolo más pequeño mediante sucesivas divisiones por 2, hasta llegar a k=1.

```
Procedimiento Shell(A,n)
        k = TRUNC(n/2)
        mientras k >= 1 hacer
                b= 1
                mientras b!=0 hacer
                        b=0
                        para i=k hasta i>=n-1 hacer
                                si A[i-k]>A[i]
                                        temp=A[i]
                                        A[i]=A[i-k]
                                        A[i-k]=temp
                                        b=b+1
                                fin si
                        fin para
                fin mientras
                k=TRUNC(k/2)
        fin mientras
fin Procedimiento
```

El arreglo A indexa desde 0 hasta n-1 -- A[0,1,…, n-1]

Ordenamiento con un Árbol binario de búsqueda

- El ordenamiento con la ayuda de un árbol binario de búsqueda es muy simple debido a que solo requiere de dos pasos simples.
  1. Insertar cada uno de los números del vector a ordenar en el árbol binario de búsqueda.
  2. Remplazar el vector en desorden por el vector resultante de un recorrido InOrden del Árbol Binario, el cual entregara los números ordenados.

- La eficiencia de este algoritmo esta dada según la eficiencia en la implementación del árbol binario de búsqueda, lo que puede resultar mejor que otros algoritmos de ordenamiento.


```
Procedimiento OrdenaConArbolBinario(A,n)

        para i=0 hasta i>=n hacer
                Insertar(ArbolBinBusqueda,A[i]);
        fin para

        GuardarRecorridoInOrden(ArbolBinBusqueda,A);

fin Procedimiento
```


Actividades

1. Programar en ANSI C, cada uno de los algoritmos de ordenamiento mencionados.

```
inserción(t)
for i=1 to n
    x= t[i]
    j=i-1
    while j > 0 and x<t[j]
       t[j+1]=t[j]
       j=j-1
    t[j+1]=x
```

2. Adaptar el programa para que sea capaz de recibir un parámetro “n” que indica el numero de enteros a ordenar a partir de un archivo con máximo 10,000,000 de números en desorden.

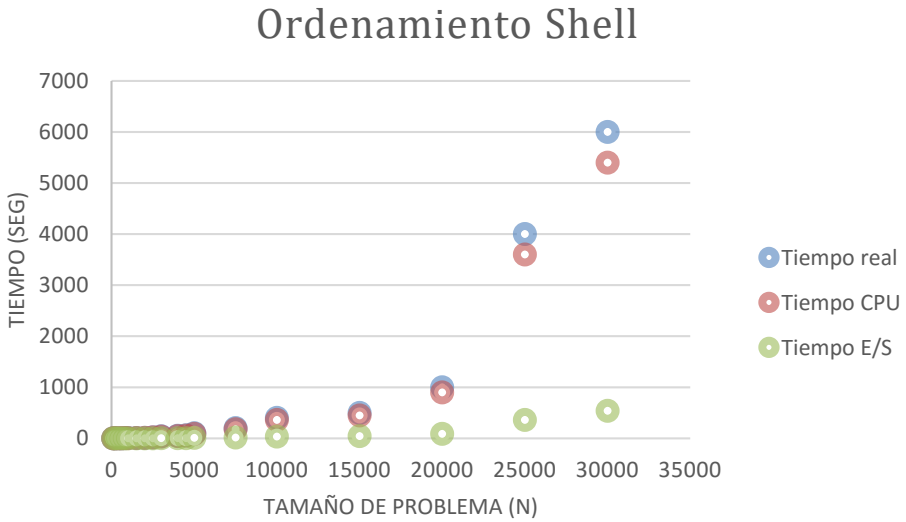
3. Medir el tiempo que tarda cada algoritmo en ordenar el archivo completo *(n=10,000,000)* y compare los tiempos *(Real y de CPU)* de cada algoritmo gráficamente en una grafica de barras *(2 graficas de barras).*

- Auxiliarse de la librería de C proporcionada para medir tiempos de ejecución bajo Linux.

Algoritmo	Tiempo Real	Tiempo CPU	Tiempo E/S	% CPU/Wall

4. Realizar un análisis temporal **para cada algoritmo, ordenando**:

- Los primeros 100, 1000, 5000, 10000, 50000, 100000, 200000, 400000, 600000, 800000, 1000000, 2000000, 3000000, 4000000, 5000000, 6000000, 7000000, 8000000, 9000000 y 10000000.
- Graficar el comportamiento temporal de cada algoritmo



5. Graficar una comparativa de los 5 algoritmos de ordenamiento (Tiempo real).

6. Realizar una **aproximación polinomial** del comportamiento temporal (tiempo real), de cada uno de los algoritmos probados según el punto 4.
   - Aproximar cada algoritmo con un polinomio de grado 1, 2, 3, 4 y 8.

7. Mostrar gráficamente la comparativa de las aproximaciones para cada algoritmo *(5 graficas)* y determinar de manera justificada cuál es la mejor aproximación para cada algoritmo.

8. Determine con base en las aproximaciones obtenidas cual será el tiempo real de cada algoritmo para 50000000, 100000000, 500000000, 1000000000 y 500000000 de números a ordenar.

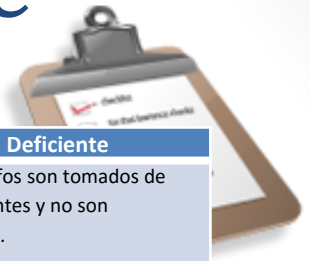
9. Finalmente responda a las siguientes preguntas:

i. ¿Cuál de los 5 algoritmos es más fácil de implementar?
ii. ¿Cuál de los 5 algoritmos es el más difícil de implementar?
iii. ¿Cuál algoritmo tiene menor complejidad temporal?
iv. ¿Cuál algoritmo tiene mayor complejidad temporal?
v. ¿Cuál algoritmo tiene menor complejidad espacial? ¿Por qué?
vi. ¿Cuál algoritmo tiene mayor complejidad espacial? ¿Por qué?
vii. ¿El comportamiento experimental de los algoritmos era el esperado? ¿Por que?
viii. ¿Sus resultados experimentales difieren mucho de los del resto de los equipos? ¿A que se debe?
ix. ¿Existió un entorno controlado para realizar las pruebas experimentales? ¿Cuál fue?
x. ¿Qué recomendaciones darían a nuevos equipos para realizar esta practica?

Observaciones

- Utilizar solo ANSI C.
- **La programación de cada uno de los métodos de ordenamiento no deberá de incluir recursividad (Algoritmos secuenciales). Por lo tanto buscar una implementación no recursiva para la inserción en un árbol binario y para realizar el recorrido inOrden.**
- Indique cual fue su plataforma experimental (Características del hardware, compilador, sistema operativo y entorno controlado)
- Se sugiere crear scripts que faciliten la experimentación.
- En el laboratorio mostrar el funcionamiento de los programas, estos ya deberá de contar con la documentación necesaria.
  - Auto-documentación
  - Documentación de funciones y algoritmos

Rubrica de evaluación del reporte



Indicador	Excelente	Muy bien	Bien	Deficiente
Construcción de párrafos	Todos los párrafos incluyen una introducción, explicaciones o detalles y una conclusión	Los párrafos incluyen información relacionada pero no fueron generalmente bien organizados	La estructura del párrafo no estaba clara y las oraciones no estaban generalmente relacionadas	Los párrafos son tomados de otras fuentes y no son originales.
Redacción	No hay errores de gramática, ortografía y puntuación y la redacción es coherentemente	No hay errores de gramática, ortografía y puntuación, pero la redacción presenta incoherencias	Pocos errores de gramática, ortografía y puntuación	Muchos errores de gramática, ortografía y puntuación
Cantidad de información Portada, Introducción, Planteamiento del problema, algoritmos e implementación, actividades y pruebas, errores detectados, posibles mejoras, conclusiones y anexos	Todos los temas son tratados de manera clara y precisa, según lo solicitado.	La mayoría de los temas son tratados de manera clara y precisa	Dos temas no están tratados o están imprecisos y no cumplen lo solicitado.	Tres o más temas no están tratados o están imprecisos y no cumplen lo solicitado.
Calidad de la información	La información está claramente relacionada con el tema principal y proporciona varias ideas secundarias y/o ejemplos	La información da respuestas a las preguntas principales, y solo da algunos detalles y/o ejemplos	La información da respuestas a las preguntas principales, pero no da detalles y/o ejemplos	La información tiene poco o nada que ver con las preguntas planteadas.
Algoritmos	Los algoritmos dan solución apoyándose de pseudocódigo, diagramas y/o figuras en un lenguaje claro.	La mayoría de los algoritmos dan solución apoyándose de pseudocódigo, pero diagramas y/o figuras.	Los algoritmos son mencionados textualmente pero no se describen	Los algoritmos no son expresados en el reporte.
Organización	La información está muy bien organizada con párrafos bien redactados y con subtítulos con estilos adecuados	La información está organizada, pero no se distingue en estilos adecuados	La información está organizada, pero los párrafos no están bien redactados	La información proporcionada no parece estar organizada o es copiada de referencias externas de manera literal

Reporte de practica

- Portada
- Introducción
- Planteamiento del problema
- Algoritmos (Descripción de la abstracción del problema y los algoritmos de ordenamiento que dan solución, apoyándose de pseudocódigo, diagramas y figuras en un lenguaje claro)
- Implementación de los algoritmos (Dados los algoritmos de ordenamiento como se implementaron en el código)
- Actividades y Pruebas (Verificación de la solución, pruebas y resultados de la práctica según lo solicitado)
- Errores detectados (Si existe algún error detectado, el cuál no fue posible resolver o se desconoce el motivo y solo ocurre con ciertas condiciones es necesario describirlo)
- Posibles mejoras (Describir posibles disminuciones de código en la implementación o otras posibles soluciones)
- Conclusiones (Por cada integrante del equipo)
- Anexo (Códigos fuente *con colores e instrucciones de compilación)
- Bibliografía (En formato IEEE)

Entrega vía Web

Grupo	Contraseña
3CM1	analisis3cm1
3CM3	analisis3cm3

- **En un solo archivo comprimido** (ZIP, RAR, TAR, JAR o GZIP)
  - Reporte (DOC, DOCX o PDF)
  - Códigos fuente (.C, .H, etc.)
    - **Código documentado:** Titulo, descripción, fecha, versión, autor.
    - *(Funciones y Algoritmos: ¿Qué hace?, ¿Cómo lo hace?, ¿Qué recibe?, ¿Qué devuelve?, ¿Causa de errores?).*
    - OBSERVACIONES
    - *NO enviar ejecutables o archivos innecesarios, las instrucciones de compilación van en el anexo del reporte. (Yo compilare los fuente).
    - *NO enviar archivo de números en desorden ni archivo de números ordenados.

Fechas de entrega

- **Demostración** *Laboratorio de Programación 2 (2107)*
  - 3CM3 “Lunes 26 de febrero o lunes 05 de marzo de 2018”.
  - 3CM1 “Miércoles 28 de febrero o miércoles 07 de marzo de 2018”.

- **Entrega de reporte y código**
  - En un solo archivo comprimido.

- **Fecha y hora limite de entrega vía Web**
  - Miércoles 14 de Marzo de 2018 a las 23:59:59 hrs.