Tarea
Análisis de algoritmos IC3002.40 2014
Prof. Mauricio Rojas

1. **Asuma que cada una de las expresiones de abajo da el tiempo de procesamiento T(n) que toma un algoritmo para resolver un problema de tamaño n.**
   Seleccione el término dominante que tenga el mayor crecimiento en n y especifique la complejidad O más baja para cada algoritmo.

Expresión	Termino dominante	O(...)
$5 + 0.001n^3 + 0.025n$	$0.001n^3$	$O(n^3)$
$500n + 100n^{1.5} + 50 \log \log_{10} n$	$100n^{1.5}$	$O(n^{1.5})$
$0.3n + 5n^{1.5} + 2.5\, n^{1.75}$	$2.5\, n^{1.75}$	$O(n^{1.75})$
$n^2 \log_2 n + n\,(\log_2 n)^2$	$n^2 \log_2 n$	$O(n^2 \log_2 n)$
$n\ \log_3 n +\ n\ \log_2 n$	$n\ \log_3 n\,, n\ \log_2 n$	$O(n \log n)$
$3 \log_8 n +\ \log_2 \log_2 \log_2 n$	$3 \log_8 n$	$O(\log n)$
$100n + 0.01n^2$	$0.01n^2$	$O(n^2)$
$0.01n + 100n^2$	$100n^2$	$O(n^2)$
$2n + n^{0.5} + 0.5n^{1.25}$	$0.5n^{1.25}$	$O(n^{1.25})$
$0.01n\ \log_2 n + n(\log_2 n)^2$	$n(\log_2 n)^2$	$O(n(\log_2 n)^2)$
$100n\ \log_3 n + n^3 + 100n$	n^3	$O(n^3)$
$0.003\ \log_4 n +\ \log_2 \log_2 n$	$0.003\ \log_4 n$	$O(\log n)$

2. **Un algoritmo cuadrático tiene un tiempo de procesamiento T(n) = c * n ^2 y tarda T (Z) segundos para procesar Z elementos de datos. Cuanto tiempo tardaría en procesar n = 5000 elementos de datos, asumiendo que Z = 100 y que T (Z) = 1ms?**

Se reemplaza T(n) con el resultado de T (Z) y se reemplaza n con Z:

$$1 = c * 100^2$$

Se despeja c:

$$\frac{1}{100^2} = c$$

Con el valor de c, se procede a resolver el problema original

$$T(5000) = \frac{1}{100^2} * 5000^2$$

La respuesta de la operación es 2500.

3. **Un algoritmo con una complejidad de tiempo O (f(n)) y un tiempo de procesamiento T(n) = c f(n) donde f(n) es una función conocida de n, tarda 10 segundos para procesar 1000 elementos de datos. Cuanto tiempo tomara procesar 100000 elementos de datos si f(n) = n y si f(n) = n^3?**

El primer paso a seguir es despejar C utilizando los datos que se brinda:

$$T(n) = c * F(n) => c = \frac{T(n)}{F(n)}$$

Reemplazando n

$$c = \frac{T(1000)}{F(1000)}$$

Sabiendo que T (1000) es igual a 10, se reemplaza el valor

$$c = \frac{10}{F(1000)}$$

Ahora se procede a evaluar el tiempo utilizando las funciones

- $F(n) = n$

$$T(n) = c * F(n)$$

$$T(100000) = \frac{10}{1000} * 100000$$

Dando como respuesta 1000 s

- $F(n) = n^3$

$$T(n) = c * F(n)$$

$$T(100000) = \frac{10}{1000^3} * 100000^3$$

Dando como respuesta 10000000s

4. Se tienen dos paquetes de software A y B de complejidad O(n log n) y O(n) respectivamente. Y se tiene que

$$\boldsymbol{T_A(n) = C_a n \log_{10} n} \text{ y}$$
$$\boldsymbol{T_B(n) = C_b n}$$

expresan el tiempo en milisegundos de cada paquete.

Durante una prueba, el tiempo promedio de procesamiento de $\boldsymbol{n = 10^4}$ elementos de datos con los paquetes A y B es de 100ms y 500ms respectivamente. Establezca las condiciones en las cuales uno de los paquetes empieza a desempeñarse mejor que el otro y recomiendo la mejor opción si se van a procesar $\boldsymbol{n = 10^9}$

Se despeja el valor de C en cada una de las ecuaciones.

$$T_A(n) = C_A n \log_{10} n => 100 = C_A 10^4 \log_{10} 10^4 => C_A = \frac{100}{C_A 10^4 \log_{10} 10^4} = \frac{1}{400}$$

$$T_b(n) = C_b n => 500 = C_b 10^4 => C_b = \frac{500}{10^4} = \frac{1}{20}$$

Para saber cuándo un paquete se comienza a desempeñar mejor que el otro planteamos la siguiente inecuación

$$T_A > T_B$$

$$\frac{n \log_{10} n}{400} > \frac{n}{20} => \frac{n \log_{10} n}{n} > \frac{400}{20} => \log_{10} n > 20 => n = 10^{20}$$

Al resolverla nos damos cuenta que el paquete B se empieza a desempeñar mejor cuando n=10^{20}.
En base al resultado anterior, podemos concluir que para $n = 10^9$ la opción más recomendable es el A.

5. Asuma que el arreglo a contiene n valores, que el método randomValue toma un numero constante c de pasos computacionales para producir cada valor de salida, y que el método goodSort toma n log n pasos computacionales para ordenar un arreglo. Determine la complejidad O para el siguiente fragmento de código:

```
for (i = 0; i < n; i++) {   2(n-1)
   for (j = 0; j < n; j++) {6(n-1)
     a [j] = randomValue (i); 6(n-1) *C
   }
   goodSort(a); (2n-1) * n log n
}
```

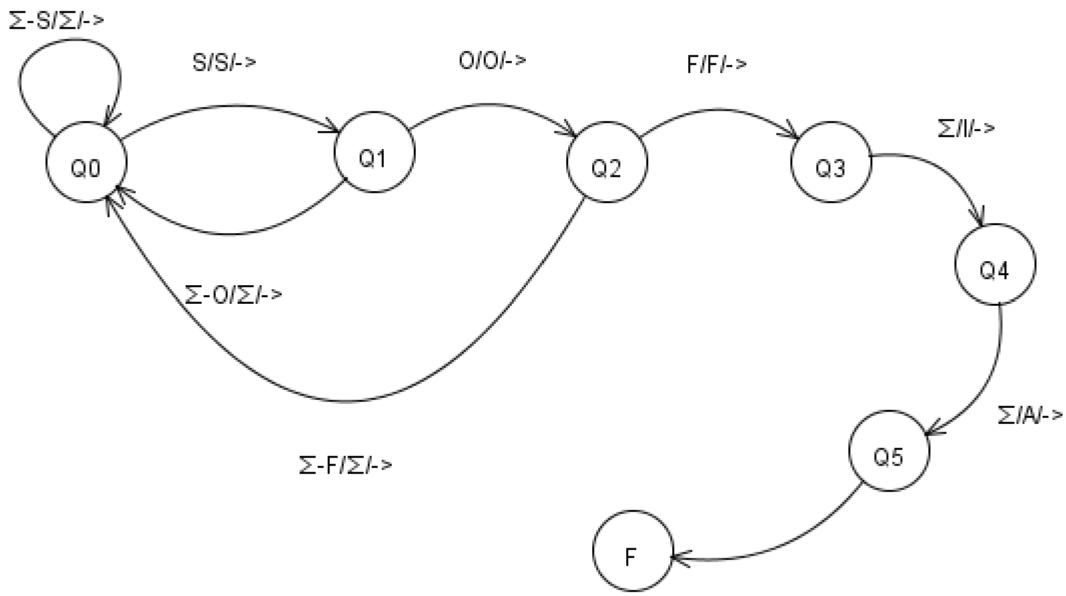
La complejidad O del fragmento de código es de $n^2 \log n$

6. **Se le pide clasificar un archivo que contiene enteros entre 0 y 999999. No puede utilizar un millón de casillas, así que en su lugar decide utilizar mil casillas numeradas desde 0 a 999 (recordar el ordenamiento por casillas o buckets comentado en clase). Comienza la clasificación colocando cada entero en la casilla correspondiente a sus tres primeras cifras. A continuación utiliza mil veces la ordenación por inserción para ordenar el contenido de cada casilla por separado.**
Y por último se vacían las casillas por orden para obtener una secuencia completamente ordenada.
Haga el pseudo código del algoritmo y realice el análisis del tiempo de ejecución. Compare con los tiempos esperados para ejecutar el ordenamiento solo usando el algoritmo de ordenamiento por inserción.

```
Función bucket-sort (elementos)
  Casilleros ← lista de 999

Para i = 0 hasta longitud (elementos)
    Si elementos[i]>999 entonces
        Elementos[i] = obtenerdigitos (elemento[i])
    C ←  elemento[i]
    Insertar elementos[i] en casillero[c]
  Fin para
  Para i = 1 hasta 999 hacer
    Inserción (casilleros[i])
  Fin para

Función obtener dígitos (numero)
    Si numero>100000 entonces
        Numero= la parte entera de numero / 1000
    Si numero>10000 entonces
        Numero= la parte entera de numero / 100
    Si numero>1000 entonces
        Numero= la parte entera de numero / 10

     Regrese número
```

7. Cree una máquina de Turing que reemplace su Nick por su nombre. (Puede utilizar solo las primeras 3 letras del Nick y las primeras 5 del nombre).

Σ = Todas las letras del alfabeto y el blanco
R= Todas las letras del alfabeto y el blanco
Q= [Q,F]
F= [F]



Tira Original

H	O	L	A	#	S	O	F	#	#	#	#	#	#	#	#	#

Tira después de ejecutar la máquina

H	O	L	A	#	S	O	F	I	A	#	#	#	#	#	#	#