Tarea # 1

Análisis y Diseño de Algoritmos / Ingeniería Civil Informática
Departamento Ciencias de la Computación y
Tecnologías de la Información

Universidad del Bío-Bío

Prof.: Gilberto Gutiérrez Ayud. Israel Gajardo

Primavera 2018

Problema # 1 Suponga tres bloques de programa, $B1$, $B2$ y $B3$. El número de comparaciones (peor caso) para cada bloque está definido por las siguientes funciones:

a) bloque `B1` $t1(n) = 10n^4 + n^2 + 3n$

b) bloque `B2` $t2(n) = \frac{n^3}{2} - \frac{n}{2}$

c) bloque `B3` $t3(n) = n^2 \log_2 n + 3n^2$

Para cada uno de los siguientes algoritmos determine su complejidad (número de comparaciones en el peor caso):

a) El algoritmo A ejecuta secuencialmente los bloques $B1$ y $B3$.

b) El algoritmo B ejecuta los bloques de la siguiente manera: primero ejecuta $B1$. Luego n veces el bloque $B3$ y finalmente el bloque $B2$.

c) El algoritmo C ejecuta primero $t2(n)$ veces el bloque $B3$, luego el bloque $B2$ y finalmente el bloque $B1$.

Problema 2 Asuma un arreglo S de enteros de tamaño n y el lgoritmo (*MinMax()*) (ver Algorithm 1) para calcular el valor mínimo y el valor máximo almacenados en S.

a) Asumiendo como operación relevante la "comparación" entre enteros, defina la ecuación de recurrencia en el peor de los casos para el algoritmo *MinMax()*.

b) Resuelva la ecuación de recurrencia del punto anterior

c) Asumiendo un algoritmo tradicional para calcular el m
inimo y el máximo, es decir un algoritmo que revisa todos los elementos de S mejorando en cada paso el mínimo y el máximo; ¿ Cuál de los dos algoritmos es mejor, el tradicional o $MinMax()$?

d) Notar que en la línea 5 de Algorithm 1, cuando existen dos elementos en S se realizan dos comparaciones (una en $min(a, b)$ y otra en $max(a, b)$. Es claro que en este caso $\|S\| = 2$ es posible determinar el mínimo y el máximo con una sola comparación. ¿ Determine cuanto afecta este cambio al algortimo $MinMax()$.

Algorithm 1 Algoritmo recursivo para cacular MinMax

```
1: MinMax(S)
2: if ‖S‖ = 1 then
3:    return (a, a) { S = {a} }
4: else if ‖S‖ = 2 then
5:    return (min(a, b), max(a, b)) {S = {a, b}}
6: else
7:    Dividir S en dos subconjuntos S1, S2 de aproximadamente el mismo tamaño
8:    (n1, m1) =MinMax(S1)
9:    (n2, m2) =MinMax(S2)
10:   return (min(n1, n2), max(m1, m2))
11: end if
```

Problema # 3 (Problema de la suma de la subsecuencia máxima) El problema de la Suma de la Subsecuencia Máxima (SSM) consiste en, dado un conjunto de enteros (posiblemente negativos) $a_1, a_2, \ldots, a_n$, encontrar el valor máximo de $\sum_{k=i}^{j} a_k$. Por conveniencia, la suma de la subsecuencia máxima es 0 si todos los enteros son negativos. Suponiendo que los elementos se almacenan en un arreglo A de tamaño n, un algoritmo para resolver el problema SSM se presenta en Algorithm 2 (SSM1). La idea detrás del algoritmo SSM1 es muy sencilla y consiste en generar todas las posibles subsecuencias (definidas por $mejor_i$ y $mejor_j$) y seleccionar aquella en la que la suma es máxima. Todas las posibles subsecuencias se generan con los ciclos **for** de las líneas 3 y 4. El ciclo **for** que empieza en la línea 6 y termina en la línea 8 permite obtener la suma de cada subsecuencia generada. En las líneas 9 a 13 se mejora la solución candidata. Es decir, si la suma de una nueva subsecuencia es mejor que la candidata (registrada hasta el momento), entonces se actualiza dicha solución, esto es los valores de las variables $sumaMaxima, mejor_i$ y $mejor_j$ se modifican a los nuevos valores de la nueva subsecuencia.

Algorithm 2 Algoritmo $O(n^3)$ para resolver SSM1.

```
1: SSM1(A, n)
2: Sea sumaMaxima = 0; mejor_i = 0, mejor_j = 0
3: for i = 1 to n do
4:    for j = 1 to n do
5:       sumaParcial = 0;
6:       for k = i to j do
7:          sumaParcial = sumaParcial + A[k]
8:       end for
9:       if sumaParcial > sumaMaxima then
10:         sumaMaxima = sumaParcial
11:         mejor_i = i
12:         mejor_j = j
13:      end if
14:   end for
15: end for
16: return (mejor_i, mejor_j, sumaMaxima)
```

Se pide:

a) Implemente en JAVA el algoritmo SSM1.

b) Calcular la complejidad del algoritmo SSM1.

c) Diseñe e implemente en JAVA un algoritmo $O(n^2)$ SSM2 para resolver el problema SSM. Es decir usando sólo dos ciclos **for**, uno incluido en el otro. Para ello vea como puede eliminar el ciclo **for** que comienza en la línea 6 del algoritmo Algorithm 2.

d) Demuestre que su algoritmo SSM2 es de complejidad cuadrática. Es decir, usando solamente un ciclo **for**.

e) Diseñe e implemente en JAVAun algoritmo $O(n)$ SSM3 para resolver el problema SSM.

f) Demuestre que su algoritmo SSM3 es de complejidad lineal.

g) Para valores de n (tamaño del conjunto de enteros) de 10, 50,100, 500, 1000, ejecute y obtenga el tiempo promedio de cada algoritmo considerando 10 instancia del problema (10 arreglos de enteros del mismo tamaño). Genere al azar los elementos de los arreglos. Con los resultados complete la siguiente tabla:

n	SSM1(seg)	SSM2(seg)	SSM3(seg)
10			
50			
100			
500			
1000			

h) A partir de los datos de la tabla anterior, obtenga dos conclusiones respecto del rendimiento de los algoritmos.

Fecha de Entrega: 03 de Octubre de 2018. **Grupos:** 3 estudiantes máximo