Estructuras de Datos 1 - ST0245
Primer Parcial Grupo 032 (Martes)

Nombre .....
Departamento de Informática y Sistemas
Universidad EAFIT

Marzo 12 de 2019

En las preguntas de selección múltiple, una respuesta incorrecta tendrá una deducción de 0.2 puntos en la nota final. Si dejas la pregunta sin responder, la nota será de 0.0. Si no conoces la respuesta, no adivines.

1 Recursión 20%

Lika y Kefo están estudiando para el examen de matemáticas en su colegio. Hoy, ellos encontraron muchas secuencias de números interesantes, una de ellas los números Fibonacci. Para Lika –que ya conocía estos números– se le hace aburrido escribirlos todos; sin embargo, ellos encontraron otra secuencia de números llamados los números de Lucas. En el libro donde ellos encontraron ésta secuencia de números, sólo hay 7 números pertenecientes a la secuencia y al final de la hoja dice: "¿Podrás definir una relación de recurrencia que nos permita deducir el n-ésimo número de Lucas?" "Por supuesto" –dijeron ambos. Ahora, ellos quieren escribir un algoritmo que nos permita retornar el n-ésimo número de Lucas. ¿Puedes ayudarlos a escribir el algoritmo? La secuencia encontrada fue la siguiente: 2, 1, 3, 4, 7, 11, 18, ... Se sabe que el número 2 y el número 1 son el primer y segundo término, respectivamente, de la secuencia de los números de Lucas.

```
1 int lucas(int n){
2   if(n == 0) return 2;
3   if(n == 1) return 1;
4   return lucas(.........) + .........;
5 }
```

(a) (10%) Completa la línea 4 ....., ...

(b) (10%) La complejidad asintótica del algoritmo anterior, para el peor de los casos, en términos de n, es:

(i) $T(n) = T(n-1) + c$, que es $O(n)$

(ii) $T(n) = 4T(n/2) + c$, que es $O(n^2)$

(iii) $T(n) = T(n-1) + T(n-2) + c$, que es $O(2^n)$

(iv) $T(n) = c$, que es $O(1)$

2 Listas Enlazadas 20%

Sean L_1 y L_2 dos listas simplemente enlazadas, y sea x un número entero aleatorio tomado del rango $[1, n]$. Vamos a realizar las siguientes dos operaciones en las dos listas:

1. Si x NO se encuentra en la lista L_1, entonces lo insertamos al inicio de L_1 y, además, si x se encuentra en L_2, tomamos x números aleatorios del rango $[1, n]$ y los insertamos al inicio de L_2.
2. Si el número x se encuentra x veces en L_2, lo insertamos al inicio de L_1.

Puedes asumir que encontrar un número aleatorio en el rango $[1, n]$ se hace en $O(1)$. También, puedes asumir que buscar un número en una lista simplemente enlazada se hace en $O(n)$.

(a) (10%) La primera operación se ejecuta n veces. ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo que ejecuta la primera operación n veces?

(i) $O(n)$
(ii) $O(2^n)$
(iii) $O(n^2)$
(iv) $O(n^3)$

(b) (10%) La segunda operación se ejecuta n veces. ¿Cuál es la complejidad asintótica, en el peor de los casos, del algoritmo que ejecuta la segunda operación n veces?

(i) $O(n)$
(ii) $O(2^n)$
(iii) $O(n^2)$
(iv) $O(n^3)$

3 Complejidad 30%

a (10%) Sea $f(n, m) = n^2 + n \times \log(\log(m))$ y $g(n, m) = n^3 + m \times \sqrt{m}$. Calcule $O(f(n, m) + g(n, m))$. Ten en cuenta que no se sabe cuál es más grande entre n y m.

(i) $O(n^3 + n(\log(\log(m)) + m \times \sqrt{m})$
(ii) $O(n^3)$
(iii) $O(m \times \sqrt(m) + n^3)$
(iv) $O(m \times \sqrt{m})$

b (10%) Sea $f(n,m) = n \times \log(n) + m^2$ y $g(n,m) = n + m$. Calcule $O(f(n,m) \times g(n,m))$. Ten en cuenta que si la regla del producto fuera válida para el producto de dos variables, entonces $O(n \times n)$ sería $O(n)$, lo cual no es cierto. Ten en cuenta que no se sabe cuál es más grande entre n y m.

(i) $O(n \times \log(n) + m^2)$
(ii) $O(m \times n \times \log(n) + n \times m^2 + n^2 \times \log(n) + m^3)$
(iii) $O(m^3)$
(iv) $O(n^3 + m^3)$

c (10%) ¿Cuál de las siguientes afirmaciones es correcta con respecto a $func3(n)$?

```
1  void func3(int n){
2    if(n < 1) return;
3    else {
4     System.out.println(n);
5     func3(n - 1);
6    }
7  }
```

(i) Esta ejecuta $T(n) = c + T(n-1)$ pasos, que es $O(n)$.
(ii) Esta ejecuta $T(n) = n + T(n-1)$ pasos, que es $O(n!)$.
(iii) Esta ejecuta $T(n) = cn + T(n-1)$ pasos, que es $O(n!)$.
(iv) Esta ejecuta $T(n) = c + 2.T(n-1)$ pasos, , que es $O(2^n)$.

4 Recursión 30%

Un conejo está ubicado en la celda $(0,0)$ de un tablero A de $n \times m$. Si el conejo está en la casilla (i,j), el conejo puede avanzar –únicamente– horizontalmente a la casilla $(i+1,j)$ o verticalmente a la casilla $(i,j+1)$. En algunas celdas (ik,jk), hay manzanas que le darán un grado de satisfacción d al conejo y, en otras celdas (ih,jh), hay zanahorias que le darán satisfacción k al conejo. Donde hay manzanas estará el carácter '*' y donde hay zanahoras estará el carácter '#'. Donde no hay ni zanahorias ni manzanas, está el carácter '.' ¿Cuál es la mayor satisfacción que puede tener el conejo, si el conejo recorre el tablero de la manera anteriormente mencionada y tiene que llegar a la posición $(n-1,m-1)$ del tablero? Hemos escrito el siguiente algoritmo para ayudar al conejo, pero faltan algunas líneas, por favor complétalas. Puedes usar `Math.max(a, b)`, de la librería de Java.

```
1   int conejo(char[][] A, int n, int m,
        int i, int j, int d, int k){
2     if(i >= n || j >= m){
3       return 0;
4     }
5     int sat = 0;
6     if(A[i][j] == '*'){
7       sat = d;
8     }
9       if(A[i][j] == '#'){
10      sat = k;
11    }
12    if(i == n - 1 && j == m - 1){
13      return .............;
14    }
15    int fi = conejo(A, n, m, i + 1, j,
          d, k);
16    int fj = conejo(A, n, m, i, j + 1,
          d, k);
17    sat += ................;
18    return ................;
19  }
```

a) (10%) Completa la línea 13 .....

b) (10%) Completa la línea 17 .....

c) (10%) Completa la línea 18 .....