Estructuras de Datos 1 - ST0245
Examen Parcial 1 - Martes (032)

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

Septiembre 03 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.

Recursión 20%

1.1 En la vida real, la teoría de juegos ha demostrado ser muy útil en la inteligencia artificial moderna; por ejemplo, para hacer transferencia de estilo que permite generar obras de arte con el estilo de un pintor determinado. Para este parcial, considera un juego en el que un jugador puede ganar 3, 5 o 7 puntos en un solo turno. ¿De cuántas maneras puede el jugador obtener un total de T puntos? A continuación algunos ejemplos:

- Para $T = 10$. Respuesta: 3 que son $5+5$, $7+3$, $3+7$.
- Para $T = 2$. Respuesta: 0.
- Para $T = 15$. Respuesta: 8.

El algoritmo `ways` soluciona el problema, pero le faltan unas líneas. Complétalas, por favor.

```
1  int ways(int T){
2      //Caso(s) base(s).
3      ............
4      ............
5      int f1 = ways(T - 3);
6      int f2 = ways(T - 5);
7      int f3 = ways(T - 7);
8      return ...........;
9  }
```

a (10%) Completa las líneas 3, 4 ....., ...

b (10%) Completa la línea 8 .....

Complejidad 20%

2.1 (10%) Juanito implementó un algoritmo para sumar 2 matrices cuadradas de dimensión N. Su algoritmo tiene complejidad $T(n) = c \times n^2$ y toma $T(n)$ segundos para procesar n datos. ¿Cuánto tiempo tardará este algoritmo para para procesar 10000 datos, si sabemos que, para $n = 100$, $T(n) = T(100) = 1\,\text{ms}$? Recuerda que $1\,\text{s} = 1000\,\text{ms}$. Así como en los parciales de Física 1, NO olvides indicar la unidad de medida del tiempo que calcules.

_____

2.2 (10%) Un estudiante afirma que $O(\log n^2)$ es $O(\log n)$. Recuerda que $\log n^a = a \times \log n$. Determina si el estudiante tiene o no la razón y en ambos casos justifica tu respuesta.

_____
_____
_____

Complejidad 20%

3.1 Considera el siguiente algoritmo:

```
1  int doSomething(int n){
2    if(n < 0) return 0;
3    int res = 1;
4    for(int i = 1; i <= n; ++i){
5      res += doSomething(n - i);
6    }
7    return res;
8  }
```

a (10%) ¿Cuál es la ecuación de recurrencia que mejor define la complejidad, para el peor caso, del algoritmo anterior? Asume que C es la suma de todas las operaciones que toman un tiempo constante en el algoritmo.

i $T(n) = \sum_{1}^{n} T(n-i) + C$

ii $T(n) = \sum_{1}^{n-i} T(n-i) + C$

iii $T(n) = \sum_{n}^{n} T(n+i) + C$

iv $T(n) = \sum_{n}^{i} T(n-i) + C$

b (10%) ¿El algoritmo anterior siempre termina? Si estás viendo Estructuras Discretas y quieres estar seguro de tu respuesta, realiza una demostración por inducción matemática (opcional).

i Sí

ii No

Notación O 20%

4.1 (10%) Considera el siguiente algoritmo:

```
1  void f(int n){
2    for(int i=1; i*i <= n; i++) {
3      for(int j=1; j*j<=n; j++) {
4        for(int k=0; k<n; k++) {
5          for(int h=0; h<=n; h++) {
6            System.out.println("hola");
7          }
8        }
9      }
10   }
11 }
```

¿Cuál es la complejidad asintótica, para el peor de los casos, del algoritmo $f(n)$ para $n > 1$?

i $O(n^3)$

ii $O(n^2)$

iii $O(n^3 \times \sqrt{n})$

iv $O(n^4 \times \sqrt{n})$

4.2 (10%) Considera las siguientes proposiciones:

A $O(f + g) = O(max(f, g))$

B $O(f \times g) = O(f) \times O(g)$

C Si $h = O(g)$ y $g = O(f)$, entonces $h = O(f)$

D $O(c.f) = O(f)$, donde c es una constante

¿Cuál(es) de las anteriores proposiciones son verdaderas?

No es necesario justificar tu respuesta, pero, si estás viendo Estructuras Discretas, puedes realizar una demostración directa o por reducción al absurdo (opcional)

ArrayList 20%

En este punto utilizaremos una lista implementada con arreglos. Esta lista se conoce en Java como ArrayList. Ten en cuenta los siguientes métodos:

- La función `A.add(a)` agrega el elemento a al final de la lista A
- La función `Arrays.asList(a)` convierte el arreglo a en una lista implementada con arreglos (ArrayList).
- La funcion `A.contains(value)` determina si en la lista A existe el valor *value*. En caso de éxito devuelve *true* y en caso contrario devuelve $false$.
- La funcion `A.toString()` convierte la lista A en una cadena de caracteres. Como un ejemplo, convierte la lista $\{1, 2, 3\}$ en "[1, 2, 3]"

5.1 (10%) ¿Cuál es la salida de `mistery()`?

```
1  void mistery(){
2      Integer[] a = {7,8,3,1,2,1,3,8,9};
3    ArrayList<Integer> s =
4    new ArrayList<>(Arrays.asList(a));
5    System.out.println(mistery1(s, a));
6  }
7  String mistery1(ArrayList<Integer> s,
8                  Integer[] a){
9    ArrayList<Integer> sol = new
         ArrayList<>();
10   mistery2(sol, s, a);
11   return sol.toString();
12 }
13 void mistery2(ArrayList<Integer> sol,
14       ArrayList<Integer> s, Integer[] a)
             {
15   for(int ele: s){
16     if(!sol.contains(ele)){
17       sol.add(ele);
18     }
19   }
20 }
```

i $[7, 8, 3, 1, 2, 9]$

ii $[7, 8, 3, 1, 2, 1, 3, 8, 9]$

iii $[7, 8, 3, 1, 2, 1, 3]$

iv $[3, 1, 2, 1, 3]$

5.2 (10%) Imagina que se le entegra a `mistery2` un ArrayList s de n elementos y un ArrayList *sol* de tamaño 0, ¿cuál es su complejidad asintótica, en el peor de los casos, de `mistery2`?

i $O(n)$

ii $O(n^2)$

iii $O(n^3)$

iv $O(1)$