Comenzado el	jueves, 6 de mayo de 2021, 20:31
Estado	Finalizado
Finalizado en	jueves, 6 de mayo de 2021, 21:00
Tiempo empleado	29 minutos 34 segundos
Calificación	**8** de 10 (**77**%)

Pregunta **1**

Correcta

Puntúa 1 sobre 1

Suponga que se quiere plantear un programa que dado un arreglo v de n componentes, proceda a aplicar un proceso recursivo que lo vaya dividiendo en tres en forma sucesiva y que en cada subarreglo calcule el valor de la suma de la subsecuencia de suma máxima, guardando esos resultados en una lista para fines estadísticos. Suponga que alguien sugirió la siguiente clase, conteniendo el método "*sumas_maximas()*" para cumplir esa tarea:

```
import java.util.ArrayList;
public class Prueba
{
    private int v[];
    private ArrayList<Long> p;

    public Prueba(int n)
    {
        v = new int[n];
        p = new ArrayList<>();
    }

    public void generar ()
    {
        // suponga que al invocar este metodo
        // el arreglo se llena en forma aleatoria...
    }

    public ArrayList<Long> sumas_maximas()
    {
        proceso(0, v.length);
        return p;
    }

    private void proceso(int iz, int de)
    {
        int n = de - iz + 1;
        if(n >= 3)
        {
            calcular(iz, de);

            int ts = n / 3;

            int s1 = iz + ts - 1;
            proceso(iz, s1);

            int s2 = s1 + ts;
            proceso(s1 + 1, s2);

            int s3 = s2 + ts;
            proceso(s2 + 1, s3);
        }
    }

    private void calcular(int iz, int de)
```

```
        {
            long mss = 0;
            for(int i = iz; i <= de; i++)
            {
                  long sp = 0;
                  for(int j = i; j <= de; j++)
                  {
                      sp += v[j];
                      if(sp > mss)
                      {
                         mss = sp;
                      }
                  }
            }
            p.add(mss);
        }
    }
```

Sin importar la utilidad real del proceso mostrado, aplique el Teorema Maestro y determine cuál es (en notación O) el rendimiento asintótico del proceso que se dispara al invocar al método *sumas_maximas()*.

Seleccione una:

a. El tiempo de ejecucíon del algoritmo analizado tiene un peor caso $O(n^2 * log(n))$

b. El tiempo de ejecución del algoritmo analizado tiene un peor caso $O(n^2)$. ✔

c. El tiempo de ejecucíon del algoritmo analizado tiene un peor caso $O(n)$.

d. El *Teorema Maestro* no es aplicable para este algoritmo.

Pregunta **2**

Correcta

Puntúa 1 sobre 1

Suponga que se quiere plantear un programa que dado un arreglo v de n componentes, proceda a aplicar un proceso recursivo que lo vaya dividiendo en dos en forma sucesiva y que en cada subarreglo simplemente calcule el promedio de sus elementos, guardando esos elementos en una lista para fines estadísticos. Suponga que alguien sugirió la siguiente clase, conteniendo el método "*promedios()*" para cumplir esa tarea:

```
import java.util.ArrayList;
public class Prueba
{
    private int v[];
    private ArrayList<Float> p;

    public Prueba(int n)
    {
        v = new int[n];
        p = new ArrayList<>();
    }
    public void generar ()
    {
        // suponga que al invocar este metodo
        // el arreglo se llena en forma aleatoria...
    }

    public ArrayList<Float> promedios()
    {
        proceso(0, v.length);
        return p;
    }

    private void proceso(int iz, int de)
    {
        if(iz <= de)
        {
            int c = (iz + de) / 2;
            calcular(iz, de);
            proceso(iz, c);
            proceso(c+1, de);
        }
    }

    private void calcular(int iz, int de)
    {
        int t = de - iz + 1;
        float ac = 0;
        for(int i = iz; i <= de; i++)
        {
            ac += v[i];
        }
        p.add(ac / t);
    }
}
```

Sin importar la utilidad real del proceso mostrado, aplique el Teorema Maestro y determine cuál es (en notación O) el rendimiento asintótico del proceso que se dispara al invocar al método *promedios()*.

Seleccione una:

- a. El tiempo de ejecución del algoritmo analizado tiene un peor caso $O(n * \log(n))$. ✔
- b. El tiempo de ejecucíon del algoritmo analizado tiene un peor caso $O(n^2)$.
- c. El tiempo de ejecucíon del algoritmo analizado tiene un peor caso $O(n^{1.5})$.
- d. El *Teorema Maestro* no es aplicable para este algoritmo.

Pregunta **3**

Correcta

Puntúa 1 sobre 1

¿Qué relación existe entre los conceptos de *recursividad* y *relación de recurrencia*?

Seleccione una:

- a. Las relaciones de recurrencia no tendrian sentido alguno si no existiera la posibilidad de escribir procesos recursivos mediante lenguajes de programación.
- b. Todos los compiladores implementan un proceso recursivo en forma de relaciones de recurrencia.
- c. No hay relación: la recursividad es un concepto de programación y las relaciones de recurrencia son conceptos matemáticos.
- d. Si se puede plantear un problema mediante relaciones de recurrencia, entonces se puede escribir un proceso recursivo que lo resuelva (y viceversa). ✔

Pregunta **4**

Incorrecta

Puntúa 0 sobre 1

Sabemos que la recursión es una de las técnicas o estrategias básicas para el planteo de algoritmos y que una de sus ventajas es que permite el diseño de algoritmos compactos y muy claros, aunque al costo de usar algo de memoria extra en el stack segment y por consiguiente también un algo de tiempo extra por la gestión del stack. Algunos problemas pueden plantearse en forma directa procesando recursivamente diversos subproblemas menores. Tal es el caso de la *sucesión de Fibonacci*, en la cual cada término se calcula como la suma de los dos inmediatamente anteriores [esto se expresa con la siguiente relación de recurrencia: *F(n) = F(n-1) + F(n-2)* con *F(0) = 0* y *F(1) = 1*]. La clase sencilla que mostramos a continuación contiene el método estático *Fibo()* que calcula el término n-ésimo de la sucesión en forma recursiva:

```
import java.util.ArrayList;
public class Prueba
{
    public static int fibo(int n)
    {
        if(n == 0) return 0;
        if(n == 1) return 1;
        return fibo(n-1) + fibo(n-2);
    }
}
```

Sin embargo, un inconveniente adicional es que la aplicación directa de *dos o más invocaciones recursivas* en el planteo de un algoritmo podría hacer que un mismo subproblema se resuelva más de una vez, incrementando el tiempo total de ejecución. Suponga que se quiere calcular el valor del sexto término de la sucesión. Analice el árbol de llamadas recursivas que se genera al hacer la invocación *t = Fibo(6)* **¿Cuántas veces en total el método *Fibo()* se invoca para resolver un mismo problema, SIN incluir en ese conteo a las invocaciones para obtener *Fibo(0)* y *Fibo(1)*?**

Seleccione una:

- a. 0
- b. 10
- c. 12 ✖
- d. 11

Pregunta **5**

Correcta

Puntúa 1 sobre 1

Suponga que para resolver un problema dado, se ha planteado un algoritmo recursivo cuyo análisis de tiempo de ejecución responde a la siguiente relación de recurrencia:

$$T(n) = 3 * T(n/3) + O(n^2)$$

¿Cuál de las siguientes expresiones en notación *Big O* describe el peor caso en tiempo de ejecución del algoritmo, según el *Método Maestro*?

Seleccione una:

a. $O(n)$

b. $O(n^2 * log(n))$

c. $O(n^2)$ ✔

d. $O(1)$

Pregunta **6**

Correcta

Puntúa 1 sobre 1

Considere la siguiente clase simple con un método Fibo() que aplica una versión recursiva para obtener el término n-ésimo de la suceción de Fibonacci:

```
import java.util.ArrayList;
public class Prueba
{
    public static int fibo(int n)
    {
        if(n == 0) return 0;
        if(n == 1) return 1;
        return fibo(n-1) + fibo(n-2);
    }
}
```

¿Puede aplicarse para el algoritmo mostrado en *Fibo()* el Teorema Maestro del Análisis de algoritmos?

Seleccione una:

a. Sí, y su tiempo de ejecución de $O(n^2)$.

b. No, ya que el algoritmo no está basado en la estrategia Divide y Vencerás. ✔

c. Sí, y su tiempo de ejecución de O(n*log(n)).

d. Sí, y su tiempo de ejecución de O(log(n)).

Pregunta **7**

Correcta

Puntúa 1 sobre 1

¿Qué tiempo de ejecución para el peor caso tendrá el algoritmo *Mergesort* para ordenar un arreglo de *n* componentes si en lugar de particionar el arreglo en 2 subarreglos, se lo parte en 3 en cada llamada recursiva? Obviamente, suponga el mismo proceso general para *merge*, pero ahora considerando 3 vias de entrada y no solo 2.

Seleccione una:

- a. O(n*log(n)) ✔
- b. $O(n^3 * log(n))$
- c. $O(n * log^3(n))$
- d. $O(n * log(n^3))$

Pregunta **8**

Incorrecta

Puntúa 0 sobre 1

¿Cuál de las siguientes NO ES una *relación de recurrencia*?

Seleccione una:

- a. T(n) = T(n/2) + 1 [con T(1) = 1]
- b. T(n) = T(n-1) + T(n-2) ✖
- c. T(n) = P(n/3) + k [con n >= 3, k != 0 y T(1) = T(2) = 1]
- d. T(n) = n * T(n - 1) [con n >= 0, T(0) = 1]

Pregunta **9**

Correcta

Puntúa 1 sobre 1

Suponga que para resolver un problema dado, se ha planteado un algoritmo recursivo cuyo análisis de tiempo de ejecución responde a la siguiente relación de recurrencia:

```
T(n) = 2n * T(n/2) + O(n)
```

¿Cuál de las siguientes es correcta en relación a esta recurrencia y la aplicación del *Método Maestro*?

Seleccione una:

- a. El tiempo de ejecución del algoritmo analizado tiene un peor caso O(n * log(n))
- b. El tiempo de ejecucíon del algoritmo analizado tiene un peor caso O(n)
- c. El *Método Maestro* no es aplicable para esta recurrencia. ✔
- d. El tiempo de ejecucíon del algoritmo analizado tiene un peor caso O(1)

Pregunta **10**

Parcialmente correcta

Puntúa 1 sobre 1

Para problema general nombrado en la columna de la izquierda, seleccione la estrategia de planteo de algoritmos que se sabe haya resultado más útil para resolver ese problema, o bien la que sea que haya podido aplicarse para resolverlo aún sin llegar a una solución eficiente (considere a cada problema en su situación más general, y no casos particulares de cada uno):

Problema de las Ocho Reinas	Backtracking	✔
Ordenamiento rápido.	Divide y vencerás	✔
Problema del Viajante.	Fuerza Bruta [O(n!)] / Programación Dinámica [O(n^2 * 2^n)]	✔
Problema del árbol de expansión mínimo de un grafo.	Recursión	✖
Generación de gráficos fractales.	Fuerza Bruta [O(n!)] / Programación Dinámica [O(n^2 * 2^n)]	✖
Problema de la alineación de secuencias.	Programación dinámica	✔

◄ Materiales Adicionales para la Semana 04

Ir a...

Desafío 02 [Conteo de inversiones - Divide y Vencerás] ►