1er Trabajo de Control Parcial
Programación 2 Facultad 4
Curso 2014/2015
Batería A

Nombre: _____ **Grupo:** _____ **Fecha:** ___/___/___

Pregunta # 1 Dadas las siguientes funcionalidades:

void Algoritmo1(**int** n){
  int[] aux = **new int**[n];
  int p = 0;
  for(**int** i = 0; i < n; ++i){
  if(Algoritmo2(i))
  aux[p] = i * i;
  else
  aux[p] = Algoritmo3(i * i);
  ++p;
  }
}
	O(n log n) //suma de tiempos constantes y n * log n
	O(1)
	O(1)
	O(n log n) //n iteraciones * condicional
	O(log n) //siempre se cumple
	O(1)
	
	$O(n^2)$ //es mayor pero nunca se cumple
	O(1)

bool Algoritmo2(**int** n){
  bool r = **false**;
  double pot = Math.pow(2, n);
  for(**int** k = 1; !r && k <= pot); k++)
  if(n % k == 0)
  r = **true**;
  return r;
}
	O(log n) //suma de tiempos constantes y log n
	O(1)
	O(log n) //esta operación se hace en log n operaciones
	O(1) //como máximo solo realiza una iteración
	O(1) //cualquier número mayor que 1 es div por 1
	O(1) //ahora no se cumple la condición del for (!r)
	O(1)
	//**siempre retorna true para n>=0**

int Algoritmo3(**int** n){
  int aux = 1;
  for(**int** i = n; i >= 0; i-=5)
  aux *= 2;
  return aux;
}
	O(n) //1/5 es una constante que multiplica a n.
	O(1)
	O(n) //hace n/5 iteraciones
	O(1) //asignación a *aux*, tiempo constante
	O(1)

a) Determine la complejidad temporal de cada método. Justifique su respuesta empleando las fórmulas y realizando los cálculos correspondientes.

*****Cálculos para justificar el análisis de la complejidad de los algoritmos 1, 2 y 3**

T(for) = T(I1) + T(C) + iter * (T(I3) + T(I2) + T(C))

T(If) = T(C) + max(T(I1), T(I2))

Análisis del Algoritmo 3

T(Algoritmo3) = T(aux=1) + T(for) + T(return)

= O(1) + O(n) + O(1) = O(n)

Análisis del Algoritmo 2

T(Algorimo2) = T(r = false) + T(pow) + T(for) + T(return)

= O(1) + O(log n) + O(1) + O(1) = O(log n)

Nota: El porqué de la complejidad O(log n) se estudiará más adelante. Una variante más ineficiente sería efectuar n multiplicaciones de 2 (2*2*2…*2) siendo O(n) pero se puede mejorar el cáculo mediante un algoritmo Divide y Vencerás para obtener el resultado en O(log n).

Análisis del Algoritmo 1

T(Algorimo2) = T(a = int[]) + T(p = 0) + T(for) + T(return)

= O(1) + O(1) + O(n log n) + O(1) = O(n log n)

T(for Alg1) = T(i=0) + T(i<n) + n * (T(if Alg1) + T(p++) + T(++i) + T(i<n))

= O(1) + O(1) + n* (O(log n) + O(1) + O(1) + O(1)) = O(n log n)

T(if Alg1) = T(Algoritmo2(i)) + Max(T(aux[p]= i*i), T(aux[p]=Algoritmo3(i*i)))

= O(logn) + O(1) = O(log n) ***//*** **aux[p]=Algoritmo3(i*i) nunca se ejecuta**

1er Trabajo de Control Parcial
Programación 2 Facultad 4
Curso 2014/2015
Batería A

Nombre: _____ **Grupo:** _____ **Fecha:** ___/___/___

Pregunta # 2 Dado el siguiente diagrama de clases en UML, implemente la funcionalidad **RotarDerecha** en la clase *ListaDE* que desplace todos los elementos una posición hacia la derecha, pasando el último elemento a ser el primero:

NodoDE<T>
info: T # siguiente: NodoDE<T> # anterior: NodoDE<T>
+ getInfo(): T + setInfo(info: T) + getSiguiente(): NodoDE<T> + setSiguiente(siguiente: NodoDE<T>) + getAnterior(): NodoDE<T> + setAnterior(anterior: NodoDE<T>)

ListaDE<T>
primero: NodoDE<T>
+ **RotarDerecha()**

Por ejemplo: (12, 56, 43, 87, **32**) ⇨ RotarDerecha (**32**, 12, 56, 43, 87)

R/ Este problema se reduce a mover el último elemento a la primera posición.

```
public void rotarDerecha(){
    NodoDE<T> cursor = primero;                    //declaro un nodo que referencia el primer nodo de la lista

    if(cursor != null){                            //si la lista no está vacía
        while(cursor.getSiguiente() != null)       //recorrer toda la lista hasta llegar al último nodo
            cursor = cursor.getSiguiente();

        cursor.setSiguiente(primero);              //como el último pasará a la primera posición tiene que tener como siguiente el que está actualmente al principio.

        if(cursor.getAnterior() != null)           //si tiene un anterior (para el validar el caso de que tenga un solo elemento), la referencia al siguiente
            cursor.getAnterior().setSiguiente(null); //de ese anterior se hace nula.

        cursor.setAnterior(null);                  //como ahora cursor pasará a ser el primer elemento
                                                   //no tendrá un anterior

        primero.setAnterior(cursor);               //el anterior de primero pasa a ser cursor
        primero = cursor;                          //(se actualiza primero)
    }
}
```

1er Trabajo de Control Parcial
Programación 2 Facultad 4
Curso 2014/2015
Batería A

Nombre: _____ **Grupo:** _____ **Fecha:** ___/___/___

Pregunta # 3 Para simular el funcionamiento de una central telefónica se han definido las siguientes clases:

CentralTelefonica
llamadas: Queue<Llamada>
+ LlamadasEfectuadas(hasta: Fecha): List<Llamada>
…

Llamada
telf_origen: String # telf_destino: String # fecha_inicio: Fecha # duracion: **int**
+ getFechaInicio(): Fecha
…

Fecha
a: **int** # m: **int** # d: **int** # hh: **int** # mm: **int** # ss: **int**
+ getAnno(): **int** + getMes(): **int** + getDia(): **int** + getHora(): **int** + getMinuto(): **int** + getSegundo(): **int** + compareTo(d: Fecha): **int**

La central almacena en una cola las llamadas según van entrando al sistema. Es necesario implementar el servicio ***LlamadasEfectuadas*** que recibe como parámetro una fecha y debe devolver un listado de las llamadas almacenadas en el sistema hasta esa fecha. La cola de llamadas NO debe sufrir cambios.

```
public List<Llamada> LlamadasEfectuadas(Fecha hasta){

    List<Llamada> efectuadas = new LinkedList();

    Queue<Llamada> aux = new LinkedList();



    while(!llamadas.isEmpty())
    {

        Llamada actual = llamadas.poll();

        if(actual.getFechaInicio().compareTo(hasta) <= 0)
            efectuadas.add(actual);


        aux.offer(actual);

    }




    llamadas = aux;


    return efectuadas;
}
```

```
//se crea la lista de llamadas a devolver

//se crea una cola auxiliar para almacenar las llamadas que serán eliminadas de la cola de la central

//mientras la cola de llamadas de la
//central no esté vacía

//saco la primera llamada (actual)

//comparo la fecha de inicio con la fecha
//que recibo como parámetro. Si es menor o igual la adiciono a la lista.

//y encolo actual en la cola auxiliar


//una vez terminado, todas las llamadas
//están en aux y en la lista solo aquellas
//que cumplieron la condición

//asigno el valor de aux a llamadas para conservar el estado inicial.

//retorno la lista de llamadas efectuadas.
```