UNLP. Facultad de Informática.

Algoritmos y Estructuras de Datos 2006. **Trabajo Práctico 5**

Tiempos de Ejecución - Recurrencias

Objetivos

- Expresar la función recurrente de un algoritmo
- Resolver la recurrencia

Ejercicio 1.

i)

```
int rec2 (int n) {
   if (n <= 1)
      rec2 = 1;
   else  {
     rec2 =  2 * rec2 (n-1);
   }
   return rec2;
}
```

ii)

```
int rec1 (int n) {
   if (n <= 1)
      rec1 = 1;
   else  {
     rec 1 =  rec1 (n-1) + rec1 (n-1);
   }
   return rec1;
}
```

iii)

```
int  f (int n) {
  if  ( n = 0 )
     f = 0;
  else { if ( n = 1 )
            f = 1;
         else  f = f (n-2) x f (n-2);
       }
  return f;
}
```

iv)

```
int  potencia_iter (int x, n) {
   int potencia
    if  ( n = 0 )
        potencia = 1;
    else { if  ( n = 1)
             potencia  =  x:
         else  {
              potencia  = x;
              for  (int  i  = 2 ; i ≤  n ; i++) {
                 potencia = potencia * x ;
               }
         }
    }
  return  potencia;
}
```

v)

```
int  potencia_rec (int x, n) {
  if  ( n = 0 )
     potencia_rec = 1;
  else  {if  ( n = 1)
          potencia_rec  =x;
          else {
            if ( n  mod  2 ) = 0
               potencia_rec =  potencia_rec (x * x, n div 2 );
          else  potencia_rec =  potencia_rec (x * x, n div 2) * x;
        }
   }
  return potencia_rec;
}
```

Para cada uno de los algoritmos presentados:

a) **Expresar** en función de n el tiempo de ejecución.

b) **Analizar** y **resolver** la correspondiente recurrencia.

c) **Determinar** el orden de las funciones obtenidas usando notación *big-Oh*.

d) **Comparar** el tiempo de ejecución de la función 'rec2' con el de la función 'rec1'.

e) **Implementar** un algoritmo más eficiente que la función f (es decir que el T(n) sea menor).

f) **Implemente** una clase llamada **Algoritmos** en el paquete **estructurasdedatos**, que contenga los métodos dados (declarados como métodos de clase). Incluya la implementación del inciso e)

Ejercicio 2.

Resolver las siguientes recurrencias :

a)

$$T(n) = \begin{cases} 1 & \text{si } n = 1 \\ 8\,T(n/2) + n^3 & \text{si } n \geq 2 \end{cases}$$

b)

$$T(n) = \begin{cases} c & \text{si } n = 0 \\ d & \text{si } n = 1 \\ [T(n-2)]^2 & \text{si } n \geq 2 \end{cases}$$

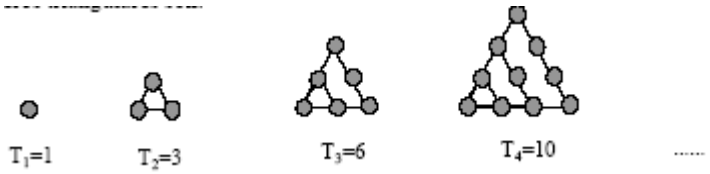
c)

$$T(n) = \begin{cases} 2 & \text{si } n = 1 \\ T(n-1) + n & \text{si } n \geq 2 \end{cases}$$

d)

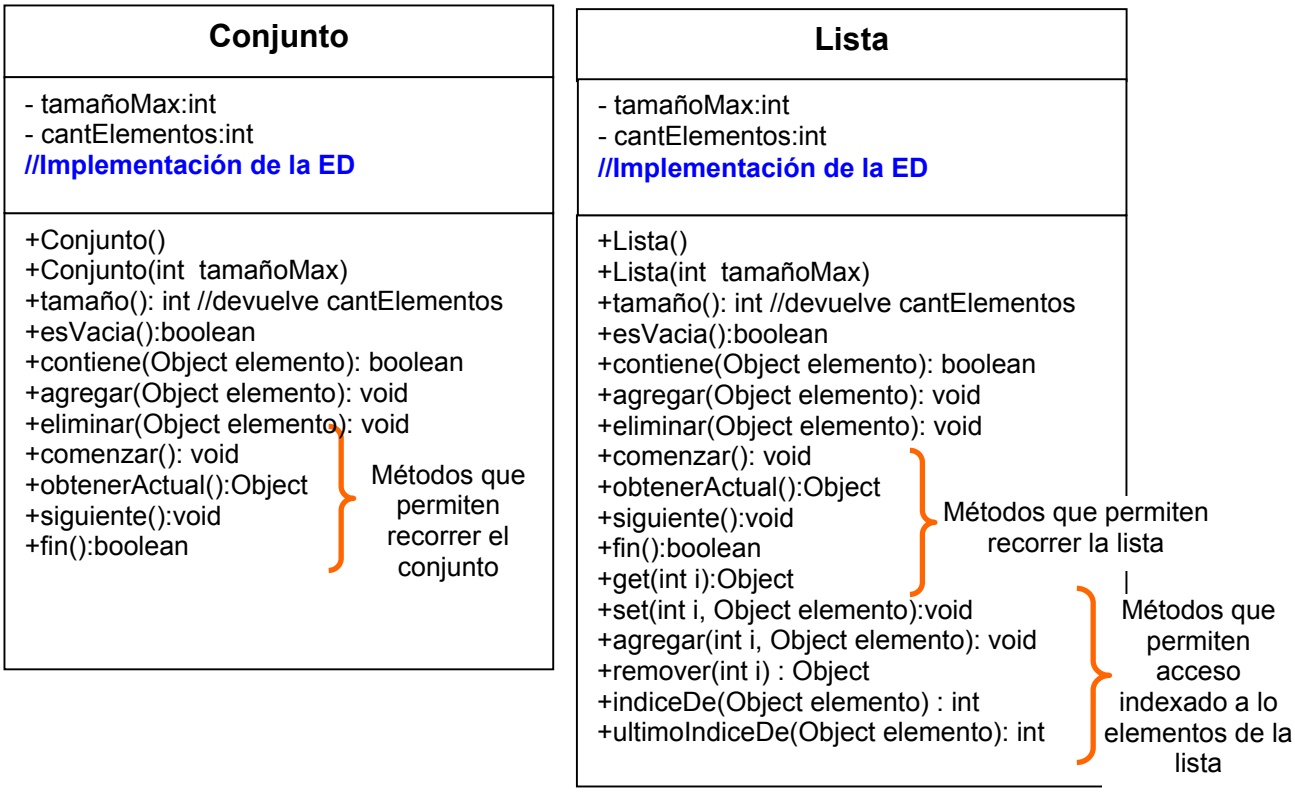
$$T(n) = \begin{cases} 2 & \text{si } n = 1 \\ T(n-1) + n/2 & \text{si } n \geq 2 \end{cases}$$

Ejecicio 3.
Los números triangulares son:



Encontrar la función de recurrencia para T(n) y resolverla.

Ejecicio 4.
Implementar las clases Lista y Conjunto en el paquete estructurasdedatos.colecciones, de acuerdo a la siguiente especificación:
a) Un objeto Lista es un grupo de objetos llamados elementos, sin orden y posiblemente duplicados. Los elementos de una lista soportan acceso posicional.
b) Un objeto Conjunto es un grupo de objetos llamados elementos, sin orden y sin elementos duplicados.



Nota: esta implementación de Lista y Conjunto genérica permite almacenar cualquier tipo de objeto. Para la realización de las mismas utilice los algoritmos de la practica 3 adaptándolos para lograr esta nueva funcionalidad.

Ejecicio 5.
Implemente los siguientes ejercicios en un paquete llamado **estructurasdedatos.aplicaciones.practica5**
a) Escribir una clase llamada **Cursada** que tenga el nombre de la materia, el año y una lista grupos de alumnos.
b) Escribir una clase **Grupo** que posea un número que lo identifica y una lista de alumnos.
c) Escribir una clase **TestCursada** que cree una cursada con algunos grupos de alumnos inscriptos e imprima el listado de todos los grupos de alumnos. Para ello redefina el metodo **toString()** de la clase Alumno y Grupo y opcionalmente Cursada.

Ejecicio 6.
Escribir una clase **TestConjunto** que cree 2 conjuntos, uno con enteros múltiplo de 7 y otro con múltiplos de 6 (Sugerencia: utilizar un bucle de 0 a 600). Luego imprima la intersección de los mismos.

Ejercicio 7.
Considerar la estrategia *mergesort*, que permite ordenar un arreglo recursivamente. Se divide el vector en dos partes, se ordena con *mergesort* cada una de ellas, y luego se combinan ambas partes ordenadas.
a) Define una clase **OrdenArreglo** en el paquete **estructurasdedatos.utiles** e implemente el método mergesort como un método de clase.
b) **Hallar** la función del tiempo de ejecución del algoritmo planteado.