Ejercicios Tema 5

Ejercicios Adaptados de Apuntes y Exámenes de EDA
Germán Moltó
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Mostrar Números Ascendentemente

- Desarrollar una rutina recursiva que muestre por pantalla de manera ascendente los números del i al n (ámbos incluidos).

```
public static void numAscendentes(int i, int n)
```

- Realizar una traza de ejecución para averiguar la secuencia de llamadas que se produce al invocar numAscendentes(7,9)
- ¿Qué ocurre si intercambiamos las dos líneas de código correspondientes al caso recursivo?

toString Recursivo de LEG

- Implementar el método toString de LEG de manera recursiva.
- Para ello, definir dos métodos en LEG:
  1. Un método vía o lanzadera:

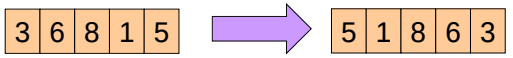
```
public String toString(){
        return toString(primero);
}
```

  2. El método recursivo:

```
private String toString(NodoLEG<E> aux){
…
}
```


Inversión Recursiva de un Vector

- Dado un array v, se pide diseñar un método recursivo genérico que invierta in-situ el orden en el que están situadas inicialmente sus componentes.
  - Tras la ejecución del método, v contendrá en su posición 0 la componente que inicialmente ocupaba la posición v.length-1, en su posición 1 la componente que inicialmente ocupaba la posición v.length-2 y así sucesivamente.



Diferentes Diseños de Sumar Array

- Implementa dos métodos recursivos diferentes para calcular la suma de las componentes de un array mediante descomposición recursiva ascendente.
  - Cada uno de ellos debe tener un caso base (finalización de la recursión) diferente.
- Calcula la complejidad temporal asintótica de cada algoritmo:
  - Talla del problema
  - Instancias significativas
  - Ecuaciones de recurrencia
  - Cotas de complejidad asintótica

Máximo Elemento de un Vector

- Diseña un método recursivo genérico que devuelva el máximo elemento de un array.
- Realizar un análisis de la complejidad temporal asintótica del algoritmo recursivo.
  - Talla del problema expresada en función de los argumentos.
  - Instancias significativas (para una talla dada)
  - Ecuaciones de Recurrencia
  - Cotas de complejidad temporal asintótica

De Lista a Vector (Externo)

- Diseñar un método recursivo dentro de una clase **ManipuladorDeListas** que dada una LEG<T> copie todos sus elementos a un array.
- Se asume que ese array ha sido convenientemente inicializado desde el código que invoca al método.

```
public static <T> void toArray(LEG<T> l,T v[])
```

- Hacer un análisis del coste del método diseñado.
- NOTA: Asumid que la clase ManipuladorDeListas está ubicada en otro paquete distinto a lineales y, por lo tanto, no se tiene acceso a la implementación de la lista enlazada.

De Lista a Vector (Interno)

- Diseñar un método recursivo dentro de la clase **LEG<E>** que copie todos sus elementos al vector que se recibe como argumento.

```
public void toArray(E[] v);
```

- Se asume que el vector que se recibe como argumento ha sido convenientemente inicializado desde el método que invoca a toArray.
- Hacer un análisis del coste del método diseñado.
- Recuerda que es posible definir métodos auxiliares.

Mismo Valor que Posicion

- Dado un array v de componentes Integer, ordenado de forma creciente y sin elementos repetidos, se quiere determinar si existe alguna componente de v que represente el mismo valor que el de su posición en v (y obtener su posición). En el caso de que no haya ninguna, se devolverá un -1.

0	1	2	3	4	5	6
-5	-4	-2	1	4	7	8

Es Capicúa

- Diseñar un método genérico que determine si un array dado es capicúa.

```
public static <T> boolean esCapicua(T[] v)
```

- Es posible construir métodos adicionales.

Sumar Vector 2 Versiones

- Razónese cual de los dos métodos recursivos que figuran a continuación resuelve con mayor eficiencia el problema de sumar todas las componentes de un vector de Integer:

```
static int sumarV1(Integer v[], int inicio) {
    if ( inicio == v.length ) return 0;
    else return sumarV1(v, inicio + 1) + v[inicio].intValue();
}

static int sumarV2(Integer v[], int inicio, int fin) {
   if ( inicio == fin ) return v[inicio].intValue();
    else { int mitad = (inicio + fin) / 2;
    return sumarV2(v, inicio, mitad) + sumarV2(v, mitad + 1, fin);
    }}
```


Inserción Directa Recursivo

- Diseña un método recursivo genérico de ordenación de un array por Inserción Directa.
- Calcula la Complejidad Temporal del mismo, indicando si existen instancias significativas.

El método buscaPar (I)

```
// v ordenado ascendentemente sin elementos repetidos. x < y
public static boolean buscaPar(Integer[] v,Integer x,Integer y,int
   izq,int der) {
   if (izq>=der) return false;
   else {
      int mitad= (izq+der)/2;
      int comp = v[mitad].compareTo(x);
      if (comp==0) // si v[mitad] es igual a x
        return (v[mitad+1].compareTo(y)==0);
      else if (comp<0) //si v[mitad] es menor que x
        return buscaPar(v,x,y,mitad+1,der);
      else // si v[mitad] es mayor que x
           return buscaPar(v,x,y,izq,mitad);
   }}
```


El método buscaPar (II)

1. Describir qué problema resuelve buscaPar, detallando el significado de cada uno de sus parámetros.
2. Describir la estrategia seguida en el diseño de buscaPar.
3. Calcular la complejidad temporal del método buscaPar.

Coste del Método Comparar

```
private static <T> boolean comparar(T a[], T b[], int inicio, int fin){
   if (inicio > fin) return true;
   int mitad = (fin + inicio) / 2;
   boolean res = a[mitad].equals(b[mitad]);
   if (res) {
      res = comparar(a, b, inicio, mitad – 1);
      if (res) res = comparar(a, b, mitad + 1, fin);
   }
   return res;
}
```

- Calcula la complejidad temporal asintótica (talla, instancias significativas, ecuaciones de recurrencia y cotas)

Complejidad Temporal de Selección Rápida

- Calcula la complejidad temporal asintótica del método de selección rápida:

```
private static <T extends Comparable<T>> void seleccionRapida(T v[], int
   k, int izq, int der) {
   if ( izq <= der){
      int indiceP = particion(v, izq, der);
      if ( k - 1 < indiceP ) seleccionRapida(v, k, izq, indiceP-1);
      else if ( k - 1 > indiceP ) seleccionRapida(v, k, indiceP+1, der);
      }
}
```