CC30A - Pauta Control 2 - viernes 28 de octubre de 2005

1.Un algoritmo de ordenamiento es estable si los elementos iguales permanecen en su orden original.Al respecto, reescriba quicksort para que realice un ordenamiento estable de acuerdo al siguiente encabezamiento:
quicksort(String[]x, int ip, int iu, int i, int j)
i y j representan los índices de los caracteres de los strings que contienen el valor por el cual deben ser ordenados. Por ejemplo, si i=10 y j=19 significan que los strings deben ordenarse de acuerdo a los valores de sus segundos 10 caracteres

Por ejemplo, si A={1B,2C,3E,4A,5C,6B,7C,8B,9A} entonces quicksort(A,0,8,1,1) entrega A={4A,9A,1B,6B,8B,2C,5C,7C,3E}

Indicación. Use un arreglo auxiliar con los strings y las posiciones originales, ordénelo y desempate los elementos de igual valor de acuerdo al valor de la posición. Finalmente, deje el resultado en el arreglo original

```
static public void  quicksort(String[]x,int ip,int iu,int i,int j){
  if(ip>=iu)return;
  //copiar en arreglo auxiliar: 2.5 puntos
  int n=iu-ip+1;              //0.2
  String[]aux=new String[n];  //0.3
  for(int k=0; k<n; ++k)                  //0.5
    aux[k] = x[ip+k].substring(i,j+1)+     //0.5
             rellenarPosicion(""+(ip+k),5)+//0.5
             x[ip+k];                      //0.5
  //ordenar arreglo auxiliar: 0.5
  quicksort(aux,0,n-1);
  //copiar en arreglo original: 1.5
  for(int k=0; k<n; ++k)                  //0.5
    x[ip+k]=aux[k].substring(j-i+1+4);    //1.0
}
//convertir indice en String de n digitos: 0.5 pto
static public String rellenarPosicion(String x,int n){
  return x.length()==n ? x : rellenarPosicion("0"+x,n);
}
//quicksort: 0.4
static public void  quicksort(String[]x,int ip,int iu){
  if(ip>iu) return;
  int i=particionar(x,ip,iu);
  quicksort(x,ip,i-1);
  quicksort(x,i+1,iu);
}
//particionar: 0.4
static public int particionar(String[]x,int ip,int iu){
  String P=x[ip];
  int i=ip;
  for(int j=ip+1; j<=iu; ++j)
    if(x[j].compareTo(P)<0) intercambiar(x,++i,j);
  intercambiar(x,i,ip);
  return i;
}
//intercambiar: 0.2
static public void intercambiar(String[]x,int i,int j){
  String a=x[i]; x[i]=x[j]; x[j]=a;
```

}
2. El TDA Diccionario puede ser representado usando una tabla de hashing con rehashing lineal con las siguientes declaraciones:

```
class Elemento{
public Object palabra, significado;
public boolean borrado;//true si eliminado
public Elemento(Object x,Object y){
  palabra=x; significado=y; borrado=false;
}
}
class Diccionario{
static protected final int N=100;
protected Elemento[] tabla;
public Diccionario(){tabla=new Elemento[N];}
//2 puntos
public int palabras(){
  int n=0,   //0.1
  for(int i=0; i<N, ++i)        //0.5
    if(tabla[i]!=null          //0.5
    && tabla[i].borrado==false) //0.5
      ++n;   //0.2
  return n;  //0.2
}
//4 puntos
public void reorganizar(){
  Elemento[]t=tabla;         //0.5
  tabla=new Elemento[n];     //1.0
  for(int i=0; i<N, ++i)     //0.5
    if(t[i]!=null            //0.5
    && t[i].borrado==false) //0.5
      agregar(t[i].palabra,t[i].significado); //1.0
}
```