CC30A – Pauta Control 2- 2006
Tiempo: 1 hr 45 minutos – CON Apuntes – SIN consultas – Contestar en hojas separadas

1.
a) implemente el método suponiendo que el Diccionario se representa con un árbol 2-3:

```
//1 punto
public Queue buscarIdentificadores(Object x){
  Queue aux=new Queue(); //0.3
  buscar(x,raiz,q);      //0.5
  return q;              //0.2
}
//2 puntos
public void buscar(Object x,Nodo r, Queue q){
  if( r==null ) return;  //0.1
  buscar(x,r.izq,q);    //0.3

  if(r.info1.equals(x)) //0.3
    q.enque(id1);       //0.2

  buscar(x,r.medio,q);  //0.3

  if(r.info2.equals(x)) //0.3
    q.enque(id2);       //0.2

  buscar(x,r.der,q);    //0.3
}
```

b) implemente el método suponiendo que el Diccionario se representa con una tabla de hashing con rehashing lineal:

```
protected Elemento[] tabla = new Elemento[N];
class Elemento{ Comparable id; Object info; bolean borrado;}

static public Queue buscarIdentificadores(Object x){
  //crear arreglo con identificadores: 1.5
  Comparable[]id=new Comparable[N];//0.2
  int n=0;                        //0.1
  for(int i=0; i<N; ++i)          //0.3
    if(!tabla[i].borrado          //0.3
    && tabla[i].info.equals(x)){ //0.3
      id[n]=tabla[i].id;          //0.2
      ++n;                        //0.1
    }
  //ordenar arreglo: 0.5
  ordenar(id,n);

  //cola con identificadores ordenados: 1.0
  Queue q=new Queue();   //0.2
  for(int i=0;i<n; ++i) //0.3
    q.enque(id[i]);     //0.3
  return q;             //0.3

}
```

2. Implemente las siguientes mejoras adicionales al quicksort

a) Reescriba el método particionar de modo que separe el archivo en 3 partes. (menores, iguales y mayores).

```
static public Par particionar(RAF x,int ip,int iu){
     //indices de primero y ultimo iguales a pivote: 0.5
     Comparable pivote=leerReg(x,ip);//0.1
     int i1=ip, i2=ip;               //0.1
     for(int j=ip+1; j<=iu; ++j){    //0.2
       Comparable reg=leerReg(x,j);  //0.1

       //procesar iguales al pivote: 1
       int c =reg.compareTo(pivote);      //0.3
       if( c==0 ){                        //0.1
         escribirReg(leerReg(x,++i2),x,j);//0.4
         escribirReg(pivote,x,i2);         //0.2
       }
       //menores que el pivote: 1
       if( c < 0 ){      //0.1
          escribirReg(leerReg(x,j),x,++i1);//0.4
          escribirReg(leerReg(x,++i2),x,j);//0.3
          escribirReg(pivote,x,i2);  //0.2
       }
     }
    //devolver resultado: 0.5
     return new Par(i1,i2);   //0.2
}
class Par{int a,b; public(int x,int y){a=x;b=y;}} //0.3
```

b) reescriba el método quicksort de modo que utilice el método anterior y posteriormente ordene recursivamente los menores y los mayores en dos threads independientes.

```
//método quicksort: 1.5
static public void quicksort(RAF x, int ip, int iu){
     if(ip >= iu) return;          //0.1
     Par p = particionar(x,ip,iu); //0.3
     T t = new T(x,ip,p.izq-1);    //0.5
     t.start();                    //0.1
     quicksort(x,p.der+1,iu);      //0.3
     t.join();                     //0.2
}
//clase thread: 1.5
class T extends Thread{            //0.2
     private RAF a;                //0.1
     private int ip, iu;           //0.2
     public T(RAF x,int y,int z){  //0.3
          a=x; ip=y; iu=z;         //0.3
     }
     public void run(){            //0.1
          quicksort(a,ip,iu);      //0.3
     }
}
```