SEGUNDO EXAMEN PARCIAL ESTRUCTURA DE DATOS Y ALGORITMOS

Colmenarejo, 8ABRIL 2013

Nombre y Apellidos: **Grupo:**

Algunas reglas:

- Antes de comenzar el examen, escribe tu nombre y grupo.
- Lee atentamente el enunciado de cada ejercicio.

1. ¿Qué salida se muestra por consola? (1 punto)

Solución:

D

D

C

```
import colme.edalib.list.singlelink.SStack;

public class ProbarPilas {

   public static void main(String args[]) {
      SStack<Character> pila=new SStack<Character>();
      pila.push('A');
      pila.push('B');
      pila.push('C');
      pila.push('D');

      System.out.println(pila.top());
      System.out.println(pila.pop());
      System.out.println(pila.pop());
   }

}
```

2. ¿Qué salida se muestra por consola? (1 punto)

Solución:

3

2

1

1

* Dependerá de la implementación de SQueue. En la implementación actual, si la cola está vacía el método devuelve null. Otras implementaciones podrían lanzar una excepción al intentar desencolar de una cola vacía.

```
import colme.edalib.list.singlelink.SQueue;;

public class ProbarColas {
   public static void main(String args[]) {
      SQueue<Integer> cola=new SQueue<Integer>();
      cola.enqueue(3);
      cola.enqueue(2);
      cola.enqueue(1);

      System.out.println(cola.dequeue());
      System.out.println(cola.dequeue());
      System.out.println(cola.front());
      System.out.println(cola.dequeue());
      System.out.println(cola.dequeue());

   }
}
```

3. ¿Qué salida se muestra por consola? (1 punto)

Solución:

B

B

A

* Dependerá de la implementación de SStack. En la implementación actual, si la pila está vacía el método devuelve null. Otras implementaciones podrían lanzar una excepción al intentar desapilar de una pila vacía.

```
import colme.edalib.list.singlelink.SStack;

public class ProbarPilas {

   public static void main(String args[]) {
      SStack<Character> pila=new SStack<Character>();
      pila.push('A');
      pila.push('B');

      System.out.println(pila.top());
      System.out.println(pila.pop());
      System.out.println(pila.pop());
      System.out.println(pila.pop());

   }

}
```

4. Completa el código del método invertirPila que recibe una pila y devuelve una pila con los elementos en orden inverso. En la solución, sólo se permite el uso de pilas (SStack).

(2 punto)

Solución:

```
public static SStack<Integer> invertirPila(SStack<Integer> pila) {
    SStack<Integer> pilaIn=new SStack<Integer>();
    while (!pila.isEmpty()) {
        pilaIn.push(pila.pop());
    }
    return pilaIn;
}
```

5. Completa el código del método borrarEto que recibe una cola y un carácter y devuelve una nueva cola compuesta por los mismos elementos excepto los que sean iguales al carácter pasado por argumento. En la solución sólo se permite el uso de colas (SSQueue)**(2 pto)**

Solución:

```
public static SQueue<Character> borrarEto(SQueue<Character> cola, char c) {
    SQueue<Character> rCola=new SQueue<Character>();

    while (!cola.isEmpty()) {
        char peek=cola.front();
        if (peek!=c) rCola.enqueue(peek);
        cola.dequeue();
    }

    return rCola;
}
```

6. Implementa el método getIndexOf de la clase SList (lista simplemente enlazada). Dicho método recibe como argumento un elemento y devuelve posición en la lista del primer nodo que contiene dicho elemento. Si el elemento no existe, el método debe devolver -1.**(2 pto)**

Solución:

```
public int getIndexOf(E elem) {
int index = -1; //1
SNode<E> nodeIt = firstNode;//1
while (nodeIt != null) {//  n+1
        ++index; //n
        if (nodeIt.getElement().equals(elem)) {//n
                return index;
        }
        nodeIt = nodeIt.nextNode;//n
}
return -1;//1
}
```

7. Estima el tiempo de ejecución y orden del método anterior. Razona tu respuesta. (**1 pto**)

Solución:

T(n)=1+1+n+1+n+n+n+1=4n+4

En el peor de los caso e es el último elemento de la lista o no existe en la lista, y por tanto, hay que recorrer todo la lista completa.

Orden Lineal (O(n))

8. Implementa el método insertAt de la clase DList (lista doblemente enlazada) que debe insertar el elemento e en la posición index dentro de la lista. (**2 pto**s)

Solución:

```
public void insertAt(int index, E elem) {
        DNode<E> newNode = new DNode<E>(elem);//1
        int i = 0;//1
        DNode<E> nodeIt = header;//1

        while ( nodeIt != tailer ) {//n+1
                if (i == index) {//n
```

```
                                newNode.nextNode = nodeIt.nextNode; //1
                                newNode.previousNode = nodeIt; //1
                                nodeIt.nextNode.previousNode = newNode;
        //1
                                nodeIt.nextNode = newNode;                                  //1
                                return;//1
                        }
                        ++i;//n
                        nodeIt = nodeIt.nextNode;//
                }

                System.out.println("DList: Insertion out of bounds");//1
        }
```

9. Estima el tiempo de ejecución y orden del método anterior. Razona tu respuesta (**1 puntos**)

Solución:

El peor de los casos es cuando se inserta al final de la lista en el que se tiene que insertar en la ú

T(n)=1+1+1+n+1 + n + 5 =2n+9

O(n)

10. Escribe un método que reciba dos colas ordenadas y devuelva una cola ordenada formada por los elementos de ambas colas

    (**3 puntos**)

```
/**
 * Método que mezcla dos colas ordenadas
 */
public static SQueue<Integer> mezclarColas(SQueue<Integer> c1, SQueue<Integer> c2) {
   SQueue<Integer> c3=new SQueue<Integer>();

   while (!c1.isEmpty() && !c2.isEmpty()){
      if (c1.front()<c2.front()) c3.enqueue(c1.dequeue());
      else c3.enqueue(c2.dequeue());
   }
   //puede que en la c1 todavía queden elementos
   while (!c1.isEmpty()){
      c3.enqueue(c1.dequeue());
   }

   //puede que en la c2 todavía queden elementos
   while (!c2.isEmpty()){
      c3.enqueue(c2.dequeue());
   }
   return c3;
}
```

11. Añade un método **recursivo** a la clase SList que permita invertir la lista (**3 puntos**)

    Solución:

```
public void reverse() {
   if (isEmpty()) return;
   SNode<E> first=this.firstNode;
   this.removeFirst();
   reverse();
   addLast(first.getElement());
}
```

12. Ordena las siguientes complejidades de menor a mayor: O(n), O(logn), $O(n^2)$, $O(n^3)$, O(nlogn). (1 punto)
    Solución:
    O(logn)< O(n) <O(nlogn) < $O(n^2)$ < $O(n^3)$,