Estructuras de Datos y Algoritmos. Curso 2012-13. Grupo 81 (Leganés).
Fecha: 04-04-2013. **Calificación 1.[ ] 2.[ ] 3.[ ] 4.[ ] 5.[ ] Total: [ ]**

Apellidos _____ Nombre _____ NIA _____

Recuerda que puedes (y debes) apoyarte en determinados dibujos y tablas, y que se valorarán. Las preguntas en las que esto es importante están marcadas con *.
Puedes usar el espacio previsto para la respuesta. Si necesitaras usar el reverso de la hoja, o las hojas cuadriculadas, indícalo.
Se entregarán todas las hojas, aunque sólo se corregirán las cuadriculadas si se indica.

---

1. (4 puntos) Test multirrespuesta – Señala la o las respuestas que consideres correctas:

1.1) En una lista simplemente enlazada (SList):
- a) La inserción al comienzo es inmediata (O(1)).
- b) La inserción al final es inmediata (O(1)).
- c) Tenemos una referencia al primer nodo y otra al último.

1.2) En una lista doblemente enlazada (DList):
- a) La inserción al comienzo es inmediata (O(1)).
- b) Borrar el último elemento es inmediato (O(1)).
- c) Hay tantos nodos como elementos hayamos insertado.

1.3) En una pila (SStack):
- a) ~~Obtener~~ *(Desapilar)* el primer elemento que fue añadido es inmediato (O(1)).
- b) ~~Obtener~~ *(Desapilar)* el último elemento que fue añadido es inmediato (O(1)).
- c) Usamos una referencia para el elemento en la cima y otra para el elemento en el fondo.

1.4) En una cola (SQueue):
- a) ~~Obtener~~ *(Desencolar)* el primer elemento que fue añadido es inmediato (O(1)).
- b) ~~Obtener~~ *(Desencolar)* el último elemento que fue añadido es inmediato (O(1)).
- c) Usamos una referencia para el elemento delantero y otra para el elemento en la cola.

2a. (2 puntos) De los siguientes métodos recursivos, que aceptan *(inicialmente)* números enteros positivos *(>=0)*, sólo uno funciona. Indica cuál es, y describe brevemente el problema en los demás. El comportamiento de los mismos no es relevante.

```
int rec (int a, int b) {
  if (b==0) {
    return rec(a+1,0);
  } else {
    return 1+rec(a+1,b-1);
  }
}
```
_____
_____
_No hay caso base_____
_____

```
int rec (int a, int b) {
  if (b==0) {
    return a;
  } else {
    return 1+rec(a-1, b);
  }
}
```
_____
__El caso base no aproxima a la solución__
_____

```
int rec (int a, int b) {
  if (a==0) {
    return b;
  } else {
    return 1+rec(a-1, b-1);
  }
}
```
_____
_____
_____
_____

```
int rec (int a, int b) {
  if (a==0) {
    return a;
  } else {
    return 1+rec(a+1, b);
  }
}
```
_____
__El caso base no aproxima a la solución__
_____

2b. (1 punto) Calcula la salida del método correcto del apartado anterior para los siguientes valores de *a* y *b*:

	b=0	b=1	b=2	b=3
a=0	0	1	2	3
a=1	0	1	2	3
a=2	0	1	2	3

3a. (1 punto*) Indica y justifica brevemente si las siguientes funciones pertenecen a O(3N +2N·logN):

a) 5N+10 __Sí, N <= N log N_____

b) N^2+5N __ No, N <= N log N _____

c) 5N·logN+3N __ Sí, N log N <= N log N _____

1 log N sqrt N N N log N N^2 …..

3b. (1 punto) De los siguientes algoritmos de búsqueda en un array de tamaño N, indica cuál es la complejidad en los casos mejor y peor, indicando cuál sería dicho caso (puedes poner un ejemplo de array si esto te ayuda):

	Mejor caso	Peor caso
Búsqueda lineal, el array no está ordenado ___1 3 5 2 4 6 _____	O(1) _____ _____Buscar 1_____ _____	O(N) _____ _____Buscar 6_____ _____
Búsqueda binaria o dicotómica, el array está ordenado _____1 2 4 6 7 9 11_____	O(1) _____ _____Buscar 6_____ _____	O(log N) _____ _____Buscar 1 ó 11____ _____

4a. (1 punto) Crea una clase que represente una lista doblemente enlazada de números enteros.

```
public class ListaEnteros _____extends DList<Integer>_____ {
}
```

4b. (~~2~~ 3 puntos *) Crea un constructor para dicha clase que lo rellene con los números contenidos en una pila, de forma que los números pares queden al comienzo de la lista y los impares al final. El orden de los elementos en la pila, lógicamente, no se respeta.

```
public ListaEnteros  (SStack<Integer> pila) {
  while (!pila.isEmpty()) {
    int a = pila.pop();
    if (a%2==0) {
      addFirst(a);
    } else {
      addLast(a);
    }
  }
}
```

Un dibujo con una pila, y otro dibujo con cómo quedaría la lista tras insertar los elementos

Continúa en la siguiente página →

4c. (2 puntos) Crea un método que devuelva en una cola de enteros aquellos números (de la lista) que sean múltiplos de *a*.

```
public SQueue<Integer> obtenerMúltiplos (int a) {
   SQueue<Integer> cola = new SQueue<Integer>();
   DNode<Integer> nodo = headre.getNextNode();
   while (nodo != trailer) {
      if (nodo.getElement() % a == 0) {
      }
      nodo = nodo.getNextNode();
   }


}
```

5a. (~~2~~ 2.5 puntos *) Crea un método que reciba un nodo de una lista simplemente enlazada, y que lo duplique, insertando a continuación ~~de él un nodo que apunte~~ *(del nodo un nuevo nodo que contenga)* al mismo elemento.

```
public static void duplicar (SNode<E> nodo) {




}
```

5b. (~~2~~ 2.5 puntos *) Crea un método que reciba un nodo de una lista doblemente enlazada, y que lo duplique, insertando antes ~~de él un nodo que apunte~~ *(del nodo un nuevo nodo que contenga)* al mismo elemento.

```
public static void duplicar (DNode<E> nodo) {




}
```