ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL
FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN
ESTRUCTURAS DE DATOS
EVALUACIÓN PARCIAL - I TÉRMINO 2019

Nombre: _____ **Paralelo:** _____

TEMA 1: PREGUNTAS DE OPCIÓN MÚLTIPLE (10 PUNTOS)

Cada una de las siguientes preguntas tiene una sola respuesta correcta. Para obtener los puntos correspondientes, seleccione únicamente el casillero de dicha respuesta. En ningún otro caso se asignarán puntos.

1) ¿Cuál de las siguientes operaciones **no** tiene tiempo constante de ejecución en una lista doblemente enlazada?:

a. addFirst
b. addLast
c. removeFirst
d. removeLast
e. get x index (donde x es un número entero)

Respuesta:

a	b	c	d	e

2) ¿Cuál de las siguientes operaciones requiere compactación en una lista implementada con arreglos estáticos?:

a. addFirst
b. addLast
c. removeFirst
d. removeLast
e. get x index (donde x es un número entero)

Respuesta:

a	b	c	d	e

3) En una lista circular simplemente enlazada, ¿cuántas referencias se necesitan y a qué nodos?:

a. 1 referencia al first.
b. 1 referencia al last.
c. 1 referencia al siguiente del last.
d. 2 referencias, una al first y otra al last.
e. 2 referencias, una al last y otra al siguiente del last.

Respuesta:

a	b	c	d	e

Los métodos **h1** y **h2** mostrados a continuación retornan códigos de hash para claves de tipo **String** que serán asociadas a valores en un mapa de **n** cubetas (para cualquier **n** > 50).

```
public int h1 (String key) {
  return (31 + key.length()) % n;
}
```

```
public int h2 (String key) {
  return 43 + (key.length() % key.length());
}
```

¿Cuál de los siguientes enunciados es verdadero respecto a cada una de estas funciones?

a. Solo sirve para almacenar en el mapa valores de tipo **String**
b. No produce colisiones
c. Podría generar índices fuera del rango [0, n-1]
d. Siempre produce colisiones
e. Podría usarse para resolver colisiones producidas por claves de tipo **Integer**

4) h1:

a	b	c	d	e

5) h2:

a	b	c	d	e

TEMA 2: PILAS (20 PUNTOS)

IMPORTANTE: El siguiente ejercicio debe ser resuelto utilizando pilas. No es permitido utilizar otros tipos de colecciones en su solución.

Implemente un método que elimine los **k** primeros elementos de un arreglo de enteros que:

- sean menores que el siguiente elemento, o
- que se conviertan en menor que el siguiente, producto de la eliminación de otro elemento

Considere el siguiente ejemplo:

Entrada: `arr = [20, 10, 25, 30, 40]` `k = 2`

Salida: `[25, 30, 40]`

Explicación: Primero se elimina el `10` porque `10 < 25`. Sin embargo, luego de esta eliminación, el `20` también debe ser eliminado porque `20 < 25`. El `25` ya se había desplazado a la izquierda producto de la eliminación del `10`. Finalmente, ya no se aplican más eliminaciones porque ya se aplicaron **k** eliminaciones.

Su método debe retornar una pila con los elementos restantes del arreglo original y debe tener el siguiente prototipo:

```
public static Stack<Integer> eliminarPrimerosK (int[] array, int k)
```

TEMA 3: COLAS Y MAPAS (20 PUNTOS)

Un servidor de correo electrónico recibe un conjunto de mensajes en una cola y debe distribuirlos a las bandejas de entrada (*inbox*) de los destinatarios correspondientes. Para cada usuario, los mensajes con mayor importancia deben aparecer primero en su inbox.

El servidor cuenta además con un filtro de correo no deseado (spam). Dependiendo del dominio del correo del remitente, el filtro envía ciertos correos a la carpeta *spam* de cada usuario. Allí, los correos de menor riesgo aparecen primero. Los correos filtrados (es decir, los que son enviados a la carpeta spam) no aparecen en el inbox.

Para validar si un dominio en particular (por ejemplo, `publicidad.com`, `ads.org`, `ads.com`, etc.) es spam, el servidor cuenta con un mapa cuya clave es el nombre del dominio y cuyo valor asociado es un número entero que representa el nivel de riesgo de dicho dominio.

Usted debe escribir el método `distribuirEmails` que, dada la cola de emails y el mapa del filtro que indica el riesgo de cada dominio, organice las bandejas de entrada y de spam de cada usuario. El método debe retornar un mapa cuyas claves son direcciones de correo de destinatarios (por ejemplo `admin@espol.edu.ec`). El valor asociado a cada clave de este mapa es un `ArrayList` que, a su vez, contiene dos `LinkedLists` de emails. La primera de estas listas es la bandeja de entrada del usuario y la segunda es su bandeja de spam.

El siguiente código muestra la clase `Email` que llega a la cola y prototipo del método `distribuirEmails` que usted debe implementar:

```
public class Email {
	private String remitente; // dirección de correo del remitente
	private String destinatario; // dirección de correo del destinario
	private int importancia;
	private String mensaje;
}
```

```
// el siguiente método retorna un mapa con claves que representan direcciones de correo de destinatarios. Cada destinatario está asociado a sus bandejas de entrada y spam.
```

```
public static Map<String, ArrayList<LinkedList<Mail>>> distribuirEmails
(Queue<Mail> emails, Map<String, Integer> filtroSpam)
```

TEMA 4: IMPLEMENTACIÓN DE TDAs (20 PUNTOS):

El TDA **Duo** permite añadir enteros a (y removerlos de) dos colecciones ordenadas en las que los duplicados son permitidos. Las operaciones de una implementación en Java de este TDA se muestran a continuación:

Operación	Descripción
`Duo()`	Crea una nueva instancia de tipo **Duo**
`void addTo (int n, int c)`	Agrega el entero **n** a la colección **c**
`void deleteFrom (int n, int c)`	Remueve el entero **n** de la colección **c**
`List<Integer> getDuplicates ()`	Retorna una lista con los enteros que se repiten en ambas colecciones.

El siguiente código ilustra el uso de esta estructura. Se muestran, además, el estado de las colecciones internas y la colección retornada por el método `getDuplicates`:

Código	Colección 1	Colección 2
`Duo duo = new Duo();`	`[]`	`[]`
`duo.addTo (23, 1);`	`[23]`	`[]`
`duo.addTo (45, 1);`	`[23, 45]`	`[]`
`duo.addTo (56, 2);`	`[23, 45]`	`[56]`
`duo.addTo (56, 2);`	`[23, 45]`	`[56, 56]`
`duo.getDuplicates ();`	`[]`	
`duo.addTo (23, 2);`	`[23, 45]`	`[23, 56, 56]`
`duo.addTo (45, 2);`	`[23, 45]`	`[23, 45, 56, 56]`
`duo.getDuplicates ();`	`[23,45]`	
`duo.deleteFrom (56, 2);`	`[23, 45]`	`[23,45]`
`duo.deleteFrom (23, 1);`	`[45]`	`[23,45]`
`duo.getDuplicates ();`	`[45]`	

a. Defina, en código Java, la clase **Duo** especificando únicamente las variables de instancia y la implementación del constructor vacío utilizado arriba. No incluya métodos *getters* o *setters*.

b. Implemente el método **`addTo`**.

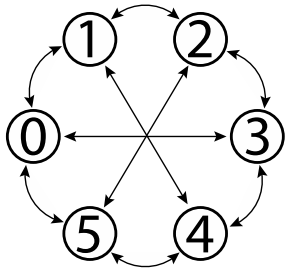
c. Implemente el método **`deleteFrom`**.

d. Implemente el médoto **`getDuplicates`** con tiempo de ejecución **`O(n)`**, donde **n** es el tamaño de una de las dos colecciones de la clase **Duo**. Para la implementación de este método no se puede hacer uso de conjuntos.

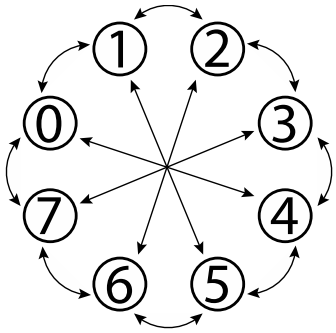
e. Explique brevemente, en español, qué modificaciones se necesitarían en la clase **Duo** (tanto en su definición como en sus métodos) para que <u>todas</u> las operaciones del TDA soporten otros tipos de datos (no solo números enteros).

TEMA 5: TDA LINKEDRING (30 PUNTOS)

El TDA **LinkedRing** utiliza un arreglo estático para producir configuraciones de nodos como las que se muestran a continuación:



LinkedRing de 6 nodos



LinkedRing de 8 nodos

Cada nodo de la estructura tiene un contenido de cualquier tipo. Cada nodo está conectado a los nodos siguiente, al previo; y almacena, además, una referencia al nodo opuesto (el que se encuentra al frente). Por ejemplo, en un `LinkedRing` de tamaño 6, el nodo en la posición 1 está conectado a los nodos: 0 (previo), 2 (siguiente) y 4 (opuesto). En un `LinkedRing` de tamaño 8, el nodo 1 tiene acceso directo a los nodos: 0 (previo), 2 (siguiente) y 5 (opuesto).

- **a.** Defina, en Java, la clase **`NodeLinkedRing`** para representar los nodos de la estructura **`LinkedRing.`** No incluya métodos *getters* o *setters*.
- **b.** Defina, en Java, la clase **`LinkedRing`** para representar el TDA **`LinkedRing.`** No incluya métodos *getters* o *setters*.
- **c.** Implemente el constructor de la clase **`LinkedRing`** que permita crear instancias de este tipoa partir de una coleción de objetos. Por ejemplo, la llamada **`new LinkedRing(List<T> objects)`** instancia un nuevo **`LinkedRing`** de **n** nodos, donde **n** es el número de elementos en la lista que recibe el constructor. Si lalista recibida contiene 6 elementos, la **LinkedRing** resultante tendrá 6 nodos, cada uno almacenando un objeto.

TEMA BONO: NOTACIÓN O GRANDE (5 PUNTOS)

Un algoritmo ordena los **n** elementos de una estructura de datos en tres pasos. La tabla mostrada a continuación detalla el tiempo de ejecución de cada uno de estos pasos en cinco implementaciones distintas del algoritmo. Usando la notación O grande, escriba en la fila inferior de la tabla la complejidad total de cada una estas implementaciones.

	Implementación 1	**Implementación 2**	**Implementación 3**	**Implementación 4**	**Implementación 5**
Paso 1	log (n)	n log(n)	37n	$1000n^2$	2^{10}
Paso 2	1000	15n	n log (n^2)	16n	3^5
Paso 3	27 log (n)	$0.002n^2$	5000 log(n)	2^n	1000000
O					