Mentorías de Computación Tarea 1: Arreglos

El objetivo de la presente tarea es ejercitar la manipulación de arreglos. Como primer acercamiento, consideraremos únicamente arreglos de enteros no negativos.

Para los puntos a resolver de la presente tarea, considera como base la siguiente plantilla de código en Java:

```java
import java.util.Arrays;

public class ArrayExercises {

  public static int[] numbers = {1, 3, 3, 6, 8, 9, 9, 10, -1};

  public static void main(String[] args) {
    System.out.println(Arrays.toString(numbers));
  }
}
```

Listing 1: ArrayExercises.java

Prueba a ejecutarla corriendo *javac ArrayExercises.java* seguido de *java ArrayExercises*.

Con el objetivo de operar sobre arreglos sin basura, buscaremos que nuestras operaciones sobre los arreglos no dejen *huecos*. Definimos a continuación un atributo sobre nuestros arreglos que tendremos presente a lo largo de la tarea.

Definición. Decimos que un arreglo A es *compacto* si para cualesquiera dos de sus elementos no existen localidades vacías entre ellos. Las localidades vacías, en el caso de arreglos de arreglos de enteros, se denotarán por la constante -1.

Ejercicios

1. En Java, los arreglos son inicializados en todas sus localidades con el valor 0. Sin embargo hemos definido nuestra constante de localidad vacía como -1. Programa un método initArray que *inicialice* todas las localidades del arreglo a -1.

```java

    public static void initArray(int[] numArray) {}

```

   De acuerdo con tu código, ¿cuál es la complejidad de inicializar los valores de un arreglo?

2. Programa un método que determine si un elemento pertenece al arreglo dado. El método debe recibir un arreglo y un valor y devolverá un booleano indicando la pertenencia del valor a dicho arreglo. La firma del método debe ser la siguiente:

```java

    public static boolean containsElement(int[] numArray, int value) {}

```

   De acuerdo con tu código, ¿cuál es la complejidad de determinar si un valor está presente en el arreglo?

3. De acuerdo a lo trabajado en la sesión, diseñamos el siguiente método que agrega un entero a un arreglo ordenado, de manera que el arreglo resultante conserve su orden:

```java

    public static void insertOrdered(int[] numArray, int value) {
      int insertIndex = 0, shift = 0, tmp = 0;
      for (int i = 0; i < numArray.length; i++) {
        if (numArray[i] > value || numArray[i] == -1) {
          insertIndex = i;
          break;
        }
      }
      shift = value;
      for (int i = insertIndex; i < numArray.length; i++) {
```

```
12          tmp = numArray[i];
13          numArray[i] = shift;
14          shift = tmp;
15        }
16      }
17
```

A pesar de ser una buena aproximación a la solución, el algoritmo presenta dos fallas importantes: no evalúa si el arreglo aún cuenta con espacio para insertar elementos y, al recorrer los elementos para insertar el nuevo valor, se recorren potencialmente todos los elementos, inclusive las posiciones de localidad vacía (-1). En la mentoría conversamos que podemos crecer nuestro arreglo si éste ya no cuenta con espacio y que, además, podemos poner atención a no copiar localidades vacías.

(a) Mejora el código anterior para que contemple las acciones recién descritas.

(b) ¿Por qué en la línea 6 del método tenemos que observar si $numArray[i] == -1$? Hint: Simula, con lápiz y papel, la inserción de un elemento mayor que cualquiera de los que se encuentran en el arreglo y observa qué pasa con la variable $insertIndex$.

(c) De acuerdo con nuestro código, ¿cuál es la complejidad de insertar un elemento en un arreglo ordenado?

4. Implementa un método dentro de nuestra clase tal que, dado un arreglo y un valor, elimine la primera ocurrencia encontrada del valor dentro de dicho arreglo. Adicionalmente, el método debe regresar un valor de verdad que indique si el elemento en efecto se encontró y fue eliminado dentro del arreglo. La firma del método será la siguiente:

```
1
2     public static void deleteElementFromArray(int[] numArray, int value) {}
3
```

No olvides que el resultado de eliminar el elemento debe garantizar que el arreglo siga siendo compacto, por lo que, si se realizó un borrado, deberás recorrer las posiciones y garantizar así que no existan *huecos* en el arreglo resultante. Hazlo de manera óptima y no recorras posiciones vacías.

(a) De acuerdo con tu código, ¿cuál es la complejidad de eliminar un elemento dentro de un arreglo ordenado?

5. Es una mala práctica de programación, en cualquier lenguaje, el uso de *números mágicos*, esto es, constantes numéricas en tu código que no son denotadas por algún identificador. Para remover nuestro número mágico, -1, define al inicio del código de tu clase una constante $EMPTYLOCATION$ y reemplaza dentro de tu código todas las ocurrencias del valor -1 por dicha constante. ¿Cuál es la ventaja de hacerlo de esta manera?

6. Programa un método que, dada una cadena, indique si ésta es un palíndromo. Recuerda que un palíndromo es una palabra que se lee igual de izquierda a derecha que de derecha a izquierda, ignorando signos de puntuación y espacios. La firma de tu método deberá ser la siguiente:

```
1
2     public static boolean isPalindrome(String sentence) {}
3
```

Para una cadena s ten en cuenta que puedes obtener su tamaño invocando $s.length()$, y que puedes obtener el i-ésimo carácter invocando $s.charAt(i)$. A pesar de que una cadena es un arreglo de *char*, tu labor es crear un arreglo auxiliar de caracteres que te ayude con tu cómputo. Hint: Crea un arreglo de *char* con la misma longitud que la cadena y copia sus caracteres al arreglo. Tú decides que tipo de caracteres ignorar al crear el arreglo. ¿Cuál es la complejidad de verificar si una palabra es un palíndromo?

Preguntas

1. Explica con tus palabras qué es una estructura de datos.
2. ¿Qué ventajas encuentras en usar arreglos? ¿Qué desventajas?
3. ¿Por qué decimos que los arreglos son de acceso aleatorio?
4. ¿A qué se refiere el que un arreglo sea una estructura de datos estática?
5. ¿Cómo se reservan las localidades de memoria en la computadora al inicializar un arreglo?
6. Supongamos que declaramos un arreglo de elementos de tipo *long* de la siguiente manera:

```
1
2    public static long[] numbers = new long[57];
3
```

   ¿Cuántos bytes de memoria se reservan para dicho arreglo?

Entrega

Las tarea se calificará como revisión de pull request sobre GitHub. Asegúrate de seguir los pasos a continuación descritos:

1. Crea un repositorio en tu cuenta de GitHub que lleve por nombre **data-structures**. En el archivo README.md indica el propósito de tu repositorio. Considera que cualquier persona puede leerlo.
2. Crea una nueva rama *tarea-1* y sube a la rama el contenido de tu archivo ArrayExercises.java dentro de las carpetas **assignments** $->$ **hw1**.
3. Crea un archivo markdown con las preguntas y respuestas, tanto de los ejercicios como de las preguntas abiertas, y súbelo como *README.md* dentro de **hw1**.
4. Crea un pull request con tus cambios y etiqueta al usuario **j-vargas-munoz** como revisor. Yo revisaré y dejaré comentarios sobre el PR en caso de que se deban hacer cambios sobre el código.
5. Cuando aprobemos el pull request y se mezcle con *master*, la tarea se considerará como finalizada. No olvides eliminar después la rama *tarea-1* para mantener limpieza en tu repositorio.

La estructura de directorios dentro de tu repositorio deberá verse como sigue:

```
README.md
assignments
    hw1
        ArrayExercises.java
        README.md
```

Consideraciones

1. Se hace uso del lenguaje de programación Java por ser lo suficientemente verboso en cuanto a tipos y sentencias para favorecer el aprendizaje.
2. No es necesario que subas tu código en un sólo commit. Lo que sí debes considerar es que cada commit que subas no impida que el código compile y se ejecute.