Estructuras de datos y algoritmos
Laboratorio 1-2: Algoritmos de ordenación y Búsqueda

Objetivos:
- Experimentar con los algoritmos de ordenación y búsqueda vistas en clase
- Utilizar el debugger para un mejor entendimiento de los pasos principales de los algoritmos
- Modificar los algoritmos para obtener unos algoritmos genéricos o independientes del tipo de información y criterio de ordenación

Proyecto
- *Mergesort*

1. Analizar la ejecución del algoritmo MergeSort vista en clase. Utiliza el debugger para entender el funcionamiento del algoritmo (ver anexo Ayuda 1). Fijaros en:
- Cómo se van produciendo las llamadas recursivas (ver Stack)
- Cuales son los valores de los parámetros de las llamadas recursivas (ver Data)
- Cómo va ordenando el array, utilizando el array temporal. (ver Data)

2. Utilizando el algoritmo de ordenación *Quicksort*(pivote elemento central), trazar las etapas necesarias para ordenar el array que contiene los elementos indicados a continuación: {8, 43, 17, 6, 40, 16, 18, 97, 11, 7}

3. Dado un array que contiene los elementos {8, 13, 17, 26, 44, 56, 88, 97} y utilizando el algoritmo de *búsqueda binaria*, trazar las etapas necesarias para encontrar el número 88.

4. Realizar un programa que compare el tiempo de cálculo de las búsquedas lineal y binaria para arrays de 10, 100, 1000, 5000 y 10000. Rellena los arrays con números enteros entre 0..a.length-1 de forma aleatoria y a continuación ordenarlos. Después invoca los algoritmos de búsqueda para un elemento que exista y para otro que no exista. (ver anexo Ayuda 2)

5. Dado un string y un array de Strings, implementar un método de búsqueda que devuelva la posición del array dónde se encuentra el string dado. En caso de no existir devolver -1.

6. Ordenar la secuencia (8, 1, 4, 1, 5, 9, 2, 6, 5) utilizando los siguientes métodos y describe el estado posterior a la ejecución del paso principal de cada algoritmo.
a. InsertionSort: ordenación por inserción
b. SelectionSort: ordenación por selección
c. MergeSort: ordenación por fusión
d. QuickSort: ordenación rápida. Utilizando el elemento central como pivote.
e. QuickSort: ordenación rápida. Utilizando la mediana de tres como pivote.

7. Cuando todos los elementos son iguales, ¿cuál es el tiempo de ejecución de los siguientes métodos?: insertionSort, selectionSort, mergeSort, quickSort

8. Cuando la entrada está ya ordenada, ¿cuál es el tiempo de ejecución de los siguientes métodos?: insertionSort, selectionSort, mergeSort, quickSort

9. Cuando la entrada está originalmente ordenada, pero en orden inverso, ¿cuál es el tiempo de ejecución de los siguientes métodos?: insertionSort, selectionSort, mergeSort, quickSort

Ayuda1 : Funcionamiento del *debugger*

1. Introducir un *breakpoint* para que el *debugger* pare en esa línea de código
2. Ejecutar el proyecto con el *debugger*
3. Step Into (F7): Para entrar dentro de la ejecución de un método o un bloque de código
4. Step over (F8): Para ejecutar sentencia por sentencia, pero sin entrar en la ejecución de los métodos
5. Step out (Shfit+F7): Para salir de la ejecución de un método
6. Ventana Stack: se ve la pila de llamadas producidas a métodos
7. Ventana Data: se ve el contenido de las variables locales del método seleccionado en la ventana Stack
8. Ventana Smart Data: se ve el contenido de las variables utilizadas en la sentencia que se va a ejecutar
9. Ventana Watch: se pueden añadir las expresiones que queremos analizar

Ayuda2: Generación de números aleatorios

```
int n = 1000;
int range = Integer.MAX_VALUE / 3 * 2;
int a[]=new int[n];
Random rand = new Random();
rand.setSeed(new Date().getTime());
for (int i=0; i<n; i++) {
    a[i]= rand.nextInt(range);
}
```

Para más información consultar el documento “Aclaración generar números.pdf” en moodle.