UT8. Algoritmos de búsqueda y ordenación

Sumario

NOTA: Los algoritmos aquí reflejados representan un caso concreto. Para cada problema pueden variar las cabeceras en función de si estamos utilizando métodos estáticos, si tenemos los arrays como atributos dentro de la clase, etc.

1 Búsqueda Lineal (Secuencial) de un elemento en un array desordenado.

Consiste en recorrer el array secuencialmente de izquierda a derecha hasta encontrar el elemento buscado o hasta alcanzar el final del array en cuyo caso finalizará la operación de búsqueda sin haber localizado el elemento.

https://www.youtube.com/watch?v=W3ClRnYb0KM

```
int busquedaSecuencial(int[] datos, int claveBusqueda) {
     //Método que recibe un array con datos y el dato que
     //queremos buscar.
     //El método devuelve en qué posición se sitúa el primero
     //que encontremos o -1 si no está el valor buscado
     for (int i = 0; i < datos.length; i++) {
          if (datos[i] == claveBusqueda) {
               return i; // devuelve el índice del entero
          }
     }
     return -1; // no se encontró el entero
}
```

2 Busqueda Lineal (Secuencial) de un elemento en un array ordenado.

Consiste en recorrer el vector secuencialmente de izquierda a derecha hasta encontrar el elemento buscado o hasta sobrepasar el valor del elemento a buscar lo que indica que no se encuentra entre los valores almacenados en el array no siendo necesario llegar hasta el final para determinar que el elemento buscado no existe.

```
int busquedaSecuencialOrdenada(int[] datos, int claveBusqueda) {
     //Método que recibe un array con datos y el dato que
     //queremos buscar.
     //El método devuelve en qué posición se sitúa el primero
     //  que encontremos o -1 si no está el valor buscado
     for (int i = 0; i < datos.length; i++) {
          if (datos[i] == claveBusqueda) {
               return i; // devuelve el índice del entero
          }
          if (datos[i]>claveBusqueda) {
               return -1;
          }
     }
     return -1; // no se encontró el entero
}
```

3 Busqueda Dicotómica (Binaria) de un elemento en un array ordenado.

Consiste en acotar el tramo de busqueda entre el extremo_izquierdo y el extremo_derecho, calculando la posición del elemento central de ese tramo.

- posición_central =(extremo_izquierdo + extremo_derecho) / 2.

Seguidamente se compara el elemento central con el valor a buscar.

En el caso de no ser iguales se acota un nuevo tramo de búsqueda asignando a extremo_izquierdo el valor posición_central + 1 o extremo_derecho el valor posicion_central – 1.

Esta operación se repite hasta que en el contenido de la posición_central coincida con el valor a buscar o los extremos se crucen (es decir extremo_izquierdo > extremo_derecho)

```
int busquedaBinaria(int [] datos, int claveBusqueda) {

     int inicio = 0;
```

```
    int fin   = datos.length - 1;
    while (inicio <= fin) {
              int centro = (inicio + fin )/2;
         if (datos[centro] < claveBusqueda) {
              inicio = centro + 1;
         } else if (datos[centro] > claveBusqueda) {
              fin   = centro - 1;
         } else {
              return centro;
          }
    }
    return -1;
}
```

4 Método de Ordenación de la Burbuja

https://www.youtube.com/watch?v=lyZQPjUT5B4

Consiste en recorrer el array de izquierda a derecha comparando el elemento apuntado con el elemento que le sigue.

Si es mayor entonces se realiza el intercambio del contenido de las celdas. En cada iteración el elemento de mayor valor acaba en la ultima posición del array.

El proceso se vuelve a repetir desde el principio del array pero hasta llegar a la posición anterior al elemento que se ordenado en la vuelta anterior.

Mejora: Si en una iteración no hay intercambios significa que el array esta ordenado y por tanto no hace falta más iteraciones y el proceso finaliza.

El algoritmo de la burbuja no es el algoritmo de ordenación más rápido, pero sí es el más fácil de entender, por eso se suele explicar el primero.

```
void ordenacionPorBurbuja(int[] datos) {
       for (int i = 1; i < datos.length; i++) {
           boolean ordenado=true;
           for (int j = 0; j < datos.length-i; j++) {
               if (datos[j + 1] < datos[j]) {
               // Intercambio (swapping)
                   int aux = datos[j + 1];
                   datos[j + 1] = datos[j];
                   datos[j] = aux;
                   ordenado=false;
```

```
                }
            }
            if (ordenado) return;
        }
    }
```

5 Método de ordenación Quicksort

Es uno de los algoritmos más rápidos, su orden de complejidad es O(n log n)

```
void quicksort(int data[], int izq, int der) {
    int i = izq, j = der;   int pivote = data[izq + (der - izq) /
2]; //elem. pivote (mitad)
    // Division en dos subarrays
    while (i <= j) {
        while (data[i] < pivote) {
            i++; //valores
        }
        // menores al pivote
        while (data[j] > pivote) {
            j--; //valores
        }
        //mayores al pivote
        if (i <= j) { // intercambiar y seguir
            int tmp = data[i];
            data[i] = data[j];
            data[j] = tmp;
            i++;
            j--;
        }
    }
    if (izq < j) {
```

```
            quicksort(data, izq, j); // Recursion subarray <
        }
        if (i < der) {
            quicksort(data, i, der); // Recursion subarray >
        }
}
```

Llamada

```
Quicksort(array, 0, array.length-1)
```

6 Otros algoritmos de ordenación

Hay otros algoritmos de ordenación como Mergesort, Inserción, Selección, Quicksort, etc. Podéis encontrar información sobre ellos en el libro recomendado de la asignatura o en cualquier otro libro de Programación.

7 Orden de complejidad de los algoritmos

Algorithm	Location	Big O
Searching Algorithms		
Linear search	Section 19.2.1	$O(n)$
Binary search	Section 19.2.2	$O(\log n)$
Recursive linear search	Exercise 19.8	$O(n)$
Recursive binary search	Exercise 19.9	$O(\log n)$
Sorting Algorithms		
Selection sort	Section 19.3.1	$O(n^2)$
Insertion sort	Section 19.3.2	$O(n^2)$
Merge sort	Section 19.3.3	$O(n \log n)$
Bubble sort	Exercises 19.5 and 19.6	$O(n^2)$

8 Ordenando objetos con Comparable

Parte del código del ejemplo realizado en clase sobre alumnos. Aprenderemos más en temas posteriores.

```
public class Alumno implements Comparable{


   //DATOS - ATRIBUTOS


   private String matricula;
   private String nombre;
   private String apellidos;
   private int nota1Eva;
   private int nota2Eva;
   private int nota3Eva;



   @Override
   public int compareTo(Object o) {
      Alumno a = (Alumno)o;
      return (this.apellidos.compareTo(a.apellidos));



….


  }
```

En el main ordenamos el ArrayList de alumnos utilizando la clase Collection

```
Collections.sort(lista_alumnos);
```