Ejercicios de recursividad

1. **Invertir cadena de caracteres:** Dada una secuencia de caracteres, diseñar un algoritmo recursivo que devuelva la misma cadena cuyos caracteres están en orden inverso.

2. **Array palíndromo:** Diseñar el siguiente procedimiento utilizando recursividad:
   ```
   public boolean comprobarPalindromo(char tablaCaracteres[], int posIzquierda, int posDerecha)
   /*Pre:
   *Post: devuelve True si y solo si posIzquierda <= i <= Derecha, T[i] = T[N – i]
   */
   ```
   Para verificar si un array es un palíndromo, la llamada inicial será de esta forma:
   ```
   comprobarPalindromo (tabla, 0, tabla.length 1);
   ```

3. **Búsqueda de un valor en una lista:** Dado un entero X y un array de enteros, diseñar un algoritmo recursivo que determine si N pertenece al array.

4. **Suma de los elementos de una lista:** Dado un array de enteros, diseñar un algoritmo recursivo que calcule la suma de todos los componentes del array.

5. **Buscar en lista ordenada:** Dada una lista de enteros **ordenada** (de menor a mayor) y un valor entero, diseñar un algoritmo recursivo que diga si el valor pertenece o no a la lista. En caso de que el valor pertenezca a la lista devolverá su posición y si no devolverá la posición en que debería colocarse.

6. **Insertar en lista ordenada:** Dada una lista de enteros **ordenada** y un valor entero, diseñar un algoritmo recursivo que inserte el valor en el lugar apropiado.

7. **Borrar en lista ordenada:** Dada una lista de enteros **ordenada** y un valor entero, diseñar un algoritmo recursivo que borre ese elemento de la lista.

8. La función de Ackerman se define de la siguiente forma:

$$\text{ACK}(m, n) = \begin{cases} n + 1 & \text{si } m = 0 \\ \text{ACK}(m - 1, 1) & \text{si } m > 0 \text{ y } n = 0 \\ \text{ACK}(m - 1, \text{ACK}(m, n - 1)) & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

9. **El problema de la suma de subconjuntos**: Diseña un subprograma que dado una lista de N enteros $A_1, A_2, \ldots A_N$ y un entero K, determine si existe un subgrupo de enteros que suma exactamente K. Describe un algoritmo de orden $O(NK)$ para resolver dicho problema.

10. **El problema de la devolución de cambios**: Diseña un subprograma que dado los valores de un conjunto de monedas ($m_1, m_2, \ldots, m_N$) y un número entero K (céntimos de euros), calcule y devuelva el número de las distintas posibilidades que hay para devolver exactamente los K céntimos en cambios.

11. **El problema de la permutación**: Implementa el siguiente procedimiento:

```
public static void permute (String str);
```

    que muestre en pantalla todas las permutaciones de los caracteres de la cadena `str`. Si `str` es *"abc"*, entonces las cadenas que se muestran serán *abc*, *bac*, *bca*, *cab* y *cba*. Utiliza la recursión.

12. Repite el problema anterior, pero de forma que el procedimiento `permute` devuelva una lista de cadena de caracteres (`List<String>`) que contenga todas las permutaciones posibles.

13. **El problema de calcular todas las sumas**: Diseña e implementa el subprograma `allSums` que dado un array de números enteros, calcule y devuelva en una lista de números enteros (`List<Integer>`) todas las posibles sumas que se pueden realizar sumando cualquier número del array, pero utilizándolo cómo máximo una vez.

    Por ejemplo, si los datos de entrada contienen los números 3, 4, 7 , entonces allSums devuelve la lista [0, 3, 4, 7, 7, 10, 11, 13]. **Notas**: los valores no hay que devolverlos en un orden determinado, pero en caso de que algún valor se obtenga de varias formas, ésta aparecerá repetida.

    Si el tamaño del array de entrada es N, cuál es el tamaño de la lista devuelta?

14. **El problema de la suma de subconjuntos**: Diseña un subprograma que dado una lista de N enteros A_1, A_2, … A_N y un entero K, determine si existe un subgrupo de enteros que suma exactamente K. Describe un algoritmo de orden $O(2^N)$ para resolver dicho problema.

15. Diseña e implementa la función `findMinAndMax`, que dado un array de números reales calcula y devuelve el mínimo y el máximo entre todos los valores (si el array solamente tiene un elemento el mínimo y el máximo serán el mismo).

```
/** Devuelve el mínimo y el máximo, agrupados en un único objeto */
public static MinMax findMinAndMax(double[] arr);

public class MinMax {
      private int min;
      private int max;
      public MinMax(double min, double max){
            this.min = min ;
            this.max = max;
      }
}
```

    Diseña e implementa el subprograma interno recursivo, que da solución a la función pública definida arriba. El programa recursivo debe dividir el problema en dos partes.