Práctica 5. Recursividad II

Objetivos de la práctica.

- Implementar subalgoritmos recursivos que utilizan inmersión en C++.
- Implementar subalgoritmos recursivos de búsqueda en C++.
- Implementar subalgoritmos recursivos de ordenación en C++.
- Seguir aprendiendo a hacer pruebas de nuestros programas.

Parte 1.

En esta primera parte de la práctica practicaremos por un lado la definición de algoritmos recursivos que utilizan inmersión en C++, y por otro la definición de pruebas para nuestros algoritmos.

Sigue los siguientes pasos:

1. Crea un nuevo proyecto de consola en CodeBlocks llamado practica5_parte1.
2. Elimina de dicho proyecto el fichero main.cpp.
3. Ve al aula virtual y descarga los ficheros practica5_parte1.cpp y catch.hpp que encontrarás en la carpeta de la práctica 5.
4. Copia los ficheros que has descargado en la carpeta donde has creado el proyecto de Codeblocks.
5. Añade los ficheros practica5_parte1.cpp y catch.hpp al proyecto de Codeblocks.
6. Abre el fichero practica5_parte1.cpp. En dicho fichero verás que aparecen las cabeceras de varias funciones que tienes que completar. Además al final del fichero aparecen unos casos de prueba.
7. Define las siguientes funciones:
    1. Diseña e implementa una función recursiva que toma como dato de entrada una cadena de caracteres y calcula su longitud.
    2. Diseña e implementa una función recursiva que dada una cadena de caracteres sin espacios en blanco, devuelve `verdad` si dicha cadena es un palíndromo y `falso` en caso contrario.
    3. Diseña e implementa una función recursiva que dada una cadena de caracteres sin espacios en blanco, devuelve `verdad` si dicha cadena contiene la letra ‘A’ y `falso` en caso contrario.

    Notas. Para implementar todas las funciones anteriores es necesario utilizar inmersión. Además debéis tener en cuenta que las cadenas de caracteres terminan con el símbolo ‘\0’.

Parte 2.

En esta segunda parte de la práctica deberás implementar algoritmos de búsqueda en vectores ordenados crecientemente. En concreto, deberás implementar la búsqueda secuencial y la búsqueda binaria (o dicotómica) con las siguientes cabeceras:

```
void busqSecR (int v[], int n, int clave, bool & esta, int & pos)
{Pre: v de tamaño n y ordenado de manera creciente}
{Post: si clave se encuentra en las n primeras componentes de v, esta toma valor true y pos indica la posición en la que se encuentra; en caso contrario esta toma valor false y pos toma el valor -1}

void busqDicR (int v[], int n, int clave, bool & esta, int & pos)
{Pre: v de tamaño n y ordenado de manera creciente}
```

```
{Post: si clave se encuentra en las n primeras componentes de v, esta toma
valor true y pos indica la posición en la que se encuentra; en caso contrario
esta toma valor false y pos toma el valor -1}
```

Además de definir estas funciones deberás crear un principal para probar que has implementado de manera correcta tanto la búsqueda binaria como la dicotómica.

Parte 3.

En esta última parte de la práctica debes recuperar el código de la práctica 1 (métodos de ordenación – medida de tiempo) y añadir una acción que implemente el método de ordenación de vectores `quicksort` ordenando las componentes del vector de manera creciente. Compara los tiempos de ejecución de este método con los que obtuviste con el resto de métodos. Prueba el tiempo de ejecución de `quicksort` tanto con vectores aleatorios como con vectores ya ordenados (por ejemplo, ordenados crecientemente y decrecientemente con el algoritmo `heapsort`). Para ello, rellena la siguiente tabla cuando sea posible (cuando no, piensa qué ocurre):

	Quicksort creciente / Vector aleatorio	**Quicksort creciente / Vector ordenado crecientemente**	**Quicksort creciente / Vector ordenado decrecientemente**
n=100.000	seg	seg	seg
n=200.000	seg	seg	seg
n=300.000	seg	seg	seg