ALGORITMOS DE ORDENAMIENTO

OBJETIVOS

Durante esta actividad, los alumnos serán capaces de:

- Resolver diferentes problemas relacionados con algoritmos de ordenamiento.

Esta actividad promueve las siguientes habilidades, valores y actitudes: análisis y síntesis, capacidad de resolver problemas, creatividad, y uso eficiente de la informática y las telecomunicaciones.

DESCRIPCIÓN DE LA ACTIVIDAD

Esta actividad puede ser elaborada de manera individual.

A partir de la clase `Sorts<T>` elaborada en clase, escribe los métodos que se describen a continuación. Todos los métodos deben de ir en el archivo `Sorts.h`.

En la parte superior del archivo coloca en comentarios los datos personales de los autores de la tarea. Por ejemplo:

```
/*----------------------------------------------------------------
 * Actividad de programación: Algoritmos de ordenamiento
 * Fecha: 30-Sep-2015
 * Autor:
 *        1160611 Anthony Stark
 *----------------------------------------------------------------*/
```

1. `std::vector<T> bucketSort(const std::vector<T> &source)`

   Devuelve un nuevo vector con el resultado de efectuar el algoritmo de *ordenamiento por casilleros* sobre `source`. Los elementos contenidos en `source` deben ser números enteros del 0 al 99. El método no debe modificar a `source`.

   Descripción del algoritmo: Empieza con diez listas vacías numeradas de la 0 a la 9. Recorre todos los elementos de `source`, colocando cada elemento en la lista que le corresponda: los elementos del 0 al 9 van en la lista 0, los elementos del 10 al 19 van en la lista 1, los elementos del 20 al 29 van en la lista 2, y así sucesivamente. Posteriormente ordena individualmente cada una de las listas de la 0 a la 9 (utiliza algunos de los métodos de la clase `Sorts` diseñados en clase). Finalmente regresa el resultado de concatenar en orden todas las listas numeradas.

2. `std::list<T> mergeList(const std::list<T> &lst1, const std::list<T> &list2)`

   Devuelve una nueva lista con el resultado de efectuar el algoritmo de ordenamiento por mezcla sobre `lst1` y `lst2`. Las listas `lst1` y `lst2` deben llegar ya ordenadas de forma ascendente. La función no debe modificar a `lst1` ni a `lst2`.

Descripción del algoritmo: Empieza con una lista resultante vacía y copias de `lst1` y `lst2`. En cada iteración determina quien tiene al inicio el elemento x más pequeño de entre las copias de `lst1` y `lst2` (recuerda que tanto `lst1` y `lst2` están ordenadas de forma ascendente, por lo que los elementos más pequeños siempre estarán al mero inicio de cada una de estas listas). Remueve a x de la lista que lo contiene y añádelo al final de la lista resultante. Las iteraciones terminan cuando alguna de las copias de `lst1` o `lst2` queda vacía. En ese momento hay que copiar a la lista resultante todos los elementos que quedaron en la lista que aún no está vacía. No olvides regresar la lista resultante.

¿QUÉ SE DEBE ENTREGAR?

Sube el archivo `Sorts.h` a Blackboard, en la sección de "Envío de tareas".

EVALUACIÓN

Esta actividad será evaluada utilizando los siguientes criterios:

100	La actividad cumple con todos los requerimientos.
-10	No se incluyó en comentario los datos del autor.
10	El programa fuente produce uno o más errores al momento de compilarlo.
50-90	El programa funciona, pero produce algunos errores a tiempo de ejecución y/o los resultados no son del todo correctos.
DA	La solución es un plagio.