CUCEI

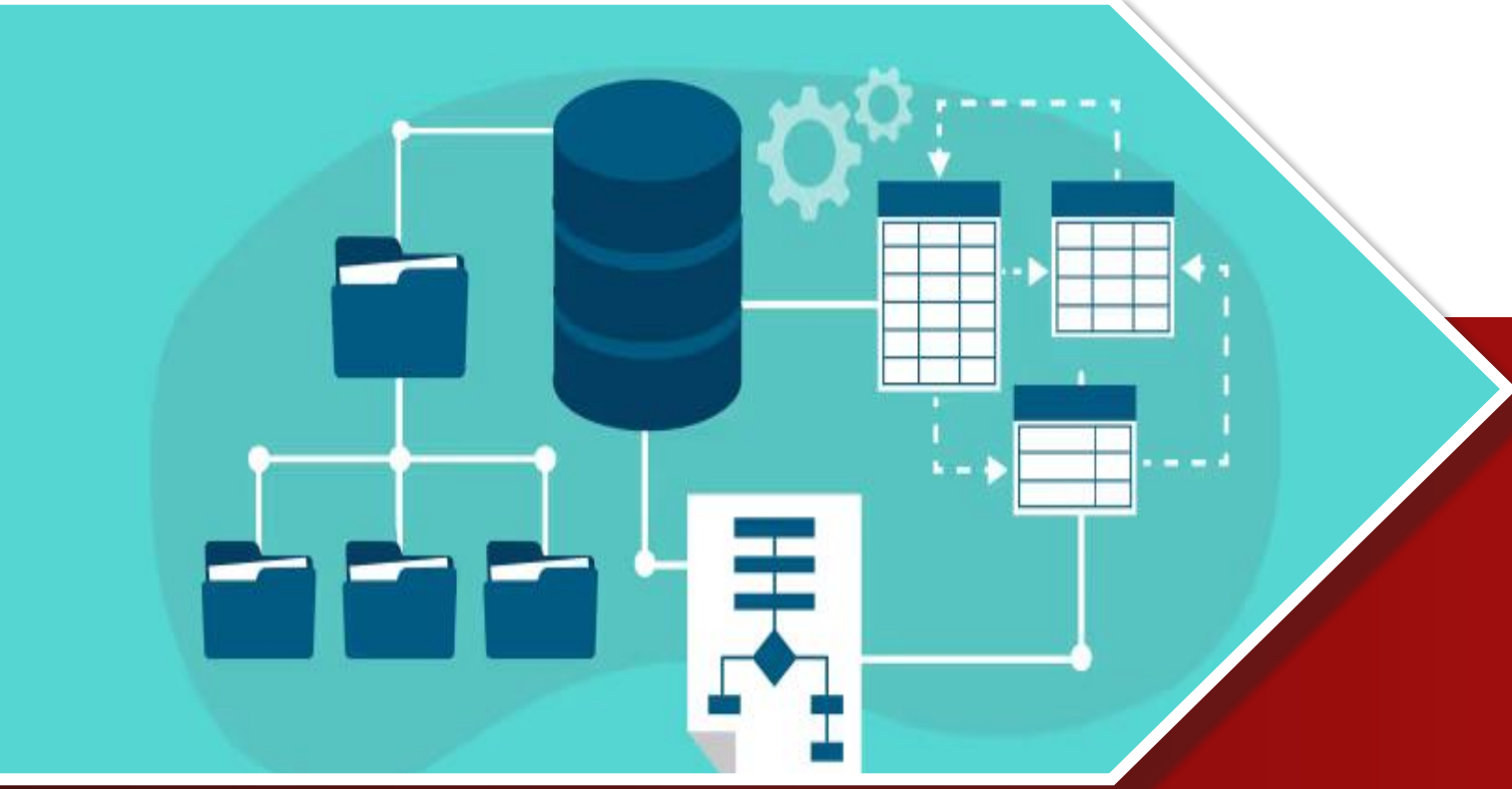
Actividad de Aprendizaje 08

Métodos de Ordenamiento Recursivos

Centro Universitario de Ciencias Exactas e Ingenierías

Materia: Estructuras de Datos

Clave: V0731 ***Sección***: D02



Alumno: Mariscal Rodríguez Omar Jesús

Código: 220858478

Profesor: Gutiérrez Hernández Alfredo

Fecha: 04 de Octubre de 2025

UNIVERSIDAD DE GUADALAJARA

PIENSA Y TRABAJA

Contenido

Test de Autoevaluación

Autoevaluación			
Concepto	***Sí***	***No***	***Acumulado***
Bajé el trabajo de internet o alguien me lo pasó (aunque sea de forma parcial)	***-100 pts***	***0 pts***	***0***
Incluí el código fuente ***en formato de texto (sólo si funciona cumpliendo todos los requerimientos)***	***+25 pts***	***0 pts***	***25***
Incluí las ***impresiones de pantalla (sólo si funciona cumpliendo todos los requerimientos)***	***+25 pts***	***0 pts***	***25***
Incluí una ***portada*** que identifica mi trabajo (nombre, código, materia, fecha, título)	***+25 pts***	***0 pts***	***25***
Incluí una ***descripción y conclusiones*** de mi trabajo	***+25 pts***	***0 pts***	***25***
		Suma:	***100***

Introducción y Abordaje del Problema

El propósito de esta actividad fue aplicar los métodos de ordenamiento que hemos estudiado hasta ahora, y de paso, ver cómo se comportan en igualdad de condiciones con un arreglo de 100,000 elementos desordenado, midiendo el tiempo que tardan en ordenar dicho arreglo.

Planeación del Programa

Imaginarnos cómo se verá el resultado final es el primer paso de aquí, y, si bien, tenemos que hacer una comparativa de cada uno de los métodos de ordenamiento juntos, para hacer un paso más allá, planeo realizar un menú sobre el cual, el usuario pueda elegir el ordenamiento solitario que quiere probar, y, por supuesto, una opción que corra todos los ordenamientos uno por uno; también, tendremos una opción para generar un nuevo dataset de números aleatorios, para compararlos en distintos escenarios. Dentro de una prueba aleatoria se confirma el desorden del arreglo previo al algoritmo, se confirma el correcto orden después y se informa del tiempo transcurrido en ms y su conversión en segundos. Para la opción que evalúa todos los algoritmos, primero se confirma que los arreglos están desordenados, se procede con cada uno de los algoritmos tomando el tiempo de cada uno y finalmente, se muestran los resultados en una tabla que muestra su tiempo en ms y s al ordenar el mismo conjunto de datos.

Dado el programa que se nos pide que hagamos, lo esencial que debe hacer es medir el tiempo de ejecución de un algoritmo, y, para no crear 6 arreglos de 100,000 (por que son 6 métodos de ordenamiento), crearemos a lo mucho 2, uno tendrá siempre el arreglo desordenado, el que se ordenará será el segundo, cuando acabe, asignaremos a este nuevamente los valores del arreglo desordenado para proceder con el siguiente arreglo. De tal manera, tendremos una clase llamada ***SortingBenchmark,*** esta funcionará la lógica detrás de tomar el tiempo de ordenamiento; siguiendo el principio S de SOLID, (Single Responsability), esta clase es lógica, no interactúa con el usuario y tiene las funciones de administrar las dos listas, tiene métodos para generara la data aleatoriamente; para la recopilación de datos que posteriormente se van a mostrar, tendremos una tercera lista (que se puede sustituir tranquilamente por una pila o una cola) de no tanto tamaño de arreglo como las de números que se encargará de registrar los resultados en tiempo y en su nombre de algoritmo, no lo hacemos de manera directa dado que esto hace que el

programa sea mucho más sostenible, si queremos añadir un nuevo algoritmo más a la comparación, no debemos crear otra tabla o modificar la existente y añadir nuevos métodos, basta con que la lista tenga este nuevo ordenamiento y una sencilla aplicación en el menú para que se realice sin mayor complicación.

Dentro de los datos que usaremos tenemos dos cosas que aclarar, primero, para almacenar el historial de resultados en la tercera lista, crearemos una nueva clase llamada ***BenchmarkResult*** que contendrá precisamente los resultados del benchmark, que será el nombre del algoritmo que se usó y el tiempo en una variable de la biblioteca chrono; también, como nuestra lista tiene métodos que atienden a otros métodos de objetos como el .toString(), si llenamos la lista con enteros, este método no serviría ya que el dato primitivo no tiene esto; así que lo que haremos es un wrapper del entero, una clase que se comporte exactamente como un entero pero añadiéndole métodos de interés como este, también le añadimos operadores con el friend, para que podamos hacer tanto Clase + entero como entero + Clase y funcione correctamente.

Programación

Con el programa y clases planteadas, seguimos con la programación, y el mayor dilema aquí fue el siguiente, dentro de ***SortingBenchmark*** no quería tener 6 métodos que cada uno ejecute una función de ordenamiento diferente, sino que se ve que son bastante repetitivas, así que lo ideal es tener una sola función que ejecute una u otra función de ordenamiento, para ello, habrá una función llamada ***runSingleBenchmark*** que se le pasará como parámetro una función, aprovechando el conocimiento que ya hemos tenido clases pasadas; pero los ordenamientos, al no ser estáticos, usamos una plantilla específica para esta función que permita llamar funciones no estáticas, y al llamarlas, las llamamos como lambda; esté método sólo se encarga de registrar el tiempo, limpiar la lista (copiando la lista desordenada a la ordenada) y añadirlo al historial, pero en sí quien llama a la función de ordenamiento es otro método llamado ***measureTime*** que también recibe una función con la misma dinámica y es esta quien almacena y retorna el tiempo, de esta manera tenemos separadas las responsabilidades.

Una vez con eso solucionado, solo era cuestión de añadir las funciones a la vista del usuario, poner algunas validaciones y revisar que funcionara correctamente.

Código Fuente

Carpeta Include

benchmarkresult.hpp

```
#ifndef __BENCHMARKRESULT_H__
#define __BENCHMARKRESULT_H__

#include <chrono>
#include <iomanip>
#include <iostream>
#include <sstream>

class BenchMarkResult {
 private:
  std::string algorithName;
  std::chrono::duration<double, std::milli> duration;

 public:
  BenchMarkResult();
  BenchMarkResult(const std::string&,
                  std::chrono::duration<double, std::milli>);
  BenchMarkResult(const BenchMarkResult&);

  // Setter's
  void setAlgorithName(const std::string&);
  void setDuration(const std::chrono::duration<double, std::milli>);

  // Getter's
  std::string toString() const;
  std::string getAlgorithName() const;
  std::chrono::duration<double, std::milli> getDuration() const;

  bool operator==(const BenchMarkResult&);
  bool operator!=(const BenchMarkResult&);
  bool operator<(const BenchMarkResult&);
  bool operator>(const BenchMarkResult&);
  bool operator<=(const BenchMarkResult&);
  bool operator>=(const BenchMarkResult&);

  BenchMarkResult& operator=(const BenchMarkResult&);
};

#endif  // __BENCHMARKRESULT_H__
```

configure.hpp

```
#ifndef __CONFIGURE_H__
#define __CONFIGURE_H__

namespace ListConfigure{
    static const long int ARRAYSIZE = 100000;
    static const int NumberAlgorithms = 100;
}

namespace RandomConfigure{
    static const long int maximunRand = 1000000;
    static const long int minimunRand = 0;
}
#endif // __CONFIGURE_H__
```

integer.hpp

```
#ifndef __INTEGER_H__
#define __INTEGER_H__

#include <cstdint>
#include <fstream>
#include <iostream>
#include <stdexcept>
#include <string>

class Integer {
 private:
  int64_t value = 0;

 public:
  Integer();
  Integer(const Integer&);
  Integer(const int64_t&);
  Integer(const std::string&);

  int64_t getValue() const;

  std::string toString() const;
  static std::string toString(const Integer&);

  void setValue(const int64_t&);

  Integer& operator=(const int64_t&);
  Integer& operator=(const Integer&);
  Integer& operator=(const std::string&);

  // Operador
  bool operator==(const Integer&) const;
  bool operator!=(const Integer&) const;
  bool operator<(const Integer&) const;
  bool operator>(const Integer&) const;
  bool operator<=(const Integer&) const;
  bool operator>=(const Integer&) const;

  static int compare(const Integer&, const Integer&);
  int compareTo(const Integer&);

  Integer operator+(const Integer&) const;
  Integer operator-(const Integer&) const;
  Integer operator*(const Integer&) const;
  Integer operator/(const Integer&) const;
  Integer operator%(const Integer&) const;
```

```
    Integer operator+(const int64_t&) const;
    Integer operator-(const int64_t&) const;
    Integer operator*(const int64_t&) const;
    Integer operator/(const int64_t&) const;
    Integer operator%(const int64_t&) const;

    Integer& operator++();
    Integer operator++(int);

    Integer& operator--();
    Integer operator--(int);

    Integer& operator+=(const Integer&);
    Integer& operator-=(const Integer&);
    Integer& operator*=(const Integer&);
    Integer& operator/=(const Integer&);
    Integer& operator%=(const Integer&);

    friend std::ostream& operator<<(std::ostream&, const Integer&);
    friend std::istream& operator>>(std::istream&, Integer&);

    // Operadores globales
    // Para poder hacer tanto Integer + int como int + Integer
    friend Integer operator+(const int64_t&, const Integer&);
    friend Integer operator-(const int64_t&, const Integer&);
    friend Integer operator*(const int64_t&, const Integer&);
    friend Integer operator/(const int64_t&, const Integer&);
    friend Integer operator%(const int64_t&, const Integer&);

    friend bool operator==(const int64_t&, const Integer&);
    friend bool operator!=(const int64_t&, const Integer&);
    friend bool operator<(const int64_t&, const Integer&);
    friend bool operator>(const int64_t&, const Integer&);
    friend bool operator<=(const int64_t&, const Integer&);
    friend bool operator>=(const int64_t&, const Integer&);
};

#endif  // __INTEGER_H__
```

list.hpp

```
#ifndef __LIST_H__
#define __LIST_H__

#include <fstream>
#include <iomanip>
#include <iostream>
#include <sstream>

#include "ownexceptions.hpp"

template <class T, int ARRAYSIZE = 1024>
class List {
 private:
  T data[ARRAYSIZE];
  int last;

  void copyAll(const List<T, ARRAYSIZE>&);

  void swapData(T&, T&);

  void sortDataMerge(const int&, const int&);
  void sortDataMerge(const int&, const int&, int(const T&, const T&));

  void sortDataQuick(const int&, const int&);
  void sortDataQuick(const int&, const int&, int(const T&, const T&));

 public:
  List<T, ARRAYSIZE>();
  List<T, ARRAYSIZE>(const List<T, ARRAYSIZE>&);

  bool isEmpty() const;
  bool isFull() const;
  bool isSorted() const;

  void insertElement(const T&, const int&);
  void deleteData(const int&);
  T* retrieve(const int&);

  // Getter's
  int getFirstPosition() const;
  int getLastPosition() const;

  int getPrevPosition(const int&) const;
  int getNextPosition(const int&) const;

  std::string toString() const;
  std::string toString(const T&, int(const T&, const T&));
```

```
  void deleteAll();

  bool isValidPosition(const int&) const;

  int findDataL(const T&);
  int findDataB(const T&);

  void insertSortedData(const T&);

  void sortDataBubble();
  void sortDataInsert();
  void sortDataSelect();
  void sortDataShell();
  void sortDataMerge();
  void sortDataQuick();

  List<T, ARRAYSIZE>& operator=(const List<T, ARRAYSIZE>&);

  template <class X>
  friend std::ostream& operator<<(std::ostream&, const List<X>&);
  template <class X>
  friend std::istream& operator>>(std::istream&, List<X>&);

  // Métodos Extras al Modelo
  bool isSorted(int(const T&, const T&)) const;

  int findDataL(const T&, int(const T&, const T&));
  int findDataB(const T&, int(const T&, const T&));

  void sortDataBubble(int(const T&, const T&));
  void sortDataInsert(int(const T&, const T&));
  void sortDataSelect(int(const T&, const T&));
  void sortDataShell(int(const T&, const T&));
  void sortDataMerge(int(const T&, const T&));
  void sortDataQuick(int(const T&, const T&));

  void insertSortedData(const T&, int(const T&, const T&));
};

template <class T, int ARRAYSIZE>
List<T, ARRAYSIZE>::List() : last(-1) {}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::copyAll(const List<T, ARRAYSIZE>& other) {
  for (int i = 0; i < other.last; i++)
    this->data[i] = other.data[i];
  this->last = other.last;
```

```
}

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isValidPosition(const int& position) const {
  return !(position > last || position < 0);
}

template <class T, int ARRAYSIZE>
List<T, ARRAYSIZE>::List(const List<T, ARRAYSIZE>& other) {
  copyAll(other);
}

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isEmpty() const {
  return this->last == -1;
}

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isFull() const {
  return this->last == (ARRAYSIZE - 1);
}

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isSorted() const {
  for (int i = 0; i < this->last; i++)
    if (this->data[i] > this->data[i + 1])
      return false;

  return true;
}

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isSorted(int cmp(const T&, const T&)) const {
  for (int i = 0; i < this->last; i++)
    if (cmp(this->data[i], this->data[i + 1]) > 0)
      return true;

  return true;
}

// Inserción en el Punto de Interés
template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::insertElement(const T& newData, const int&
position) {
  if (isFull())
    throw DataContainersExceptions::MemoryDeficiency(
        "Lista Llena, InsertElement(List)");
```

```
    if (!isValidPosition(position) && position != last + 1)
      throw DataContainersExceptions::InvalidPosition(
          "Posicion Invalida, InsertElement(List)");

  for (int i = last; i >= position; i--)
    this->data[i + 1] = this->data[i];
  this->data[position] = newData;
  last++;
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::deleteData(const int& position) {
  if (!isValidPosition(position))
    throw DataContainersExceptions::InvalidPosition(
        "Poscion Invalida, delteData(List)");

  for (int i = position; i < last; i++)
    this->data[i] = this->data[i + 1];
  last--;
}

template <class T, int ARRAYSIZE>
T* List<T, ARRAYSIZE>::retrieve(const int& position) {
  if (!isValidPosition(position))
    throw DataContainersExceptions::InvalidPosition(
        "Posicion Invalida, retrieve(List)");
  return &data[position];
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getFirstPosition() const {
  return isEmpty() ? -1 : 0;
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getLastPosition() const {
  return this->last;
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getPrevPosition(const int& position) const {
  return (!isValidPosition(position) || position == 0) ? -1 : (position -
1);
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getNextPosition(const int& position) const {
```

```
   return (!isValidPosition(position) || position == last) ? -1 :
(position + 1);
}

template <class T, int ARRAYSIZE>
std::string List<T, ARRAYSIZE>::toString() const {
  std::ostringstream oss;
  for (int i = 0; i <= this->last; i++)
    oss << this->data[i].toString() << std::endl;

  return oss.str();
}

template <class T, int ARRAYSIZE>
std::string List<T, ARRAYSIZE>::toString(const T& searched,
                                         int cmp(const T&, const T&)) {
  std::ostringstream oss;
  for (int i = 0; i <= this->last; i++)
    if (cmp(this->data[i], searched) == 0)
      oss << this->data[i].toString() << std::endl;

  return oss.str();
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::deleteAll() {
  this->last = -1;
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::insertSortedData(const T& newData) {
  int i(0);

  while ((i <= this->last) && (newData > this->data[i]))
    i++;

  insertElement(newData, i);
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::findDataL(const T& searchedData) {
  for (int i = 0; i <= this->last; i++)
    if (this->data[i] == searchedData)
      return i;
  return -1;
}

template <class T, int ARRAYSIZE>
```

```
int List<T, ARRAYSIZE>::findDataB(const T& searchedData) {
  int i(0), j(this->last), middle;

  while (i <= j) {
    middle = (i + j) / 2;
    if (this->data[middle] == searchedData)
      return middle;
    if (searchedData < this->data[middle])
      j = middle - 1;
    else
      i = middle + 1;
  }
  return -1;
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::swapData(T& a, T& b) {
  T aux = a;
  a = b;
  b = aux;
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataBubble() {
  int i(this->last), j;
  bool flag;

  do {
    flag = false;
    j = 0;

    while (j < i) {
      if (this->data[j] > this->data[j + 1]) {
        swapData(this->data[j], this->data[j + 1]);
        flag = true;
      }
      j++;
    }

    i--;
  } while (flag);
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataInsert() {
  int i(1), j;
  T aux;
```

```
    while (i <= this->last) {
      aux = this->data[i];

      j = i;
      while (j > 0 && aux < this->data[j - 1]) {
        this->data[j] = this->data[j - 1];

        j--;
      }

      if (i != j) {
        this->data[j] = aux;
      }

      i++;
    }
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataSelect() {
  int i(0), j, menor;

  while (i <= this->last) {
    menor = i;

    j = i + 1;

    while (j <= this->last) {
      if (this->data[j] < this->data[menor]) {
        menor = j;
      }
      j++;
    }

    if (i != menor) {
      this->swapData(this->data[i], this->data[menor]);
    }
    i++;
  }
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataShell() {
  int series[] = {4181, 2584, 1597, 987, 610, 377, 233, 144, 89, 55,
                  34,   21,   13,   8,   5,   3,   2,   1,   0};
  int pos(0), dif(series[pos]), i, j;

  while (dif > 0) {
```

```
      i = dif;
      while (i <= this->last) {
        j = i;

        while (j >= dif && this->data[j - dif] > this->data[j]) {
          this->swapData(this->data[j - dif], this->data[j]);
          j -= dif;
        }

        i++;
      }

      dif = series[++pos];
    }
  }

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataMerge() {
  this->sortDataMerge(0, this->last);
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataMerge(int cmp(const T&, const T&)) {
  this->sortDataMerge(0, this->last, cmp);
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataMerge(const int& leftEdge,
                                       const int& rightEdge) {
  // Criterio de Paro
  if (leftEdge >= rightEdge)
    return;

  int m((leftEdge + rightEdge) / 2);

  // Divide y Vencerás
  this->sortDataMerge(leftEdge, m);
  this->sortDataMerge(m + 1, rightEdge);

  // Intercalación
  static T temp[ARRAYSIZE];

  for (int n(leftEdge); n <= rightEdge; n++)
    temp[n] = this->data[n];

  int i(leftEdge), j(m + 1), x(leftEdge);

  while (i <= m && j <= rightEdge) {
```

```
    while (i <= m && temp[i] <= temp[j])
      this->data[x++] = temp[i++];

    if (i <= m)
      while (j <= rightEdge && temp[j] <= temp[i])
        this->data[x++] = temp[j++];
  }

  while (i <= m)
    this->data[x++] = temp[i++];
  while (j <= m)
    this->data[x++] = temp[j++];
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataMerge(const int& leftEdge,
                                       const int& rightEdge,
                                       int cmp(const T&, const T&)) {
  // Criterio de Paro
  if (leftEdge >= rightEdge)
    return;

  int m((leftEdge + rightEdge) / 2);

  // Divide y Vencerás
  this->sortDataMerge(leftEdge, m, cmp);
  this->sortDataMerge(m + 1, rightEdge, cmp);

  // Intercalación
  static T temp[ARRAYSIZE];

  for (int n(leftEdge); n <= rightEdge; n++)
    temp[n] = this->data[n];

  int i(leftEdge), j(m + 1), x(leftEdge);

  while (i <= m && j <= rightEdge) {
    while (i <= m && cmp(temp[i], temp[j]) <= 0)
      this->data[x++] = temp[i++];

    if (i <= m)
      while (j <= rightEdge && cmp(temp[j], temp[i]) <= 0)
        this->data[x++] = temp[j++];
  }

  while (i <= m)
    this->data[x++] = temp[i++];
  while (j <= m)
```

```
    this->data[x++] = temp[j++];
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataQuick() {
  this->sortDataQuick(0, this->last);
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataQuick(int cmp(const T&, const T&)) {
  this->sortDataQuick(0, this->last, cmp);
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataQuick(const int& leftEdge,
                                       const int& rightEdge) {
  // Criterio de paro
  if (leftEdge >= rightEdge)
    return;

  if (leftEdge == rightEdge + 1) {
    if (this->data[leftEdge] > this->data[rightEdge])
      this->swapData(this->data[leftEdge], this->data[rightEdge]);
    return;
  }

  // Separación
  int i(leftEdge), j(rightEdge);

  while (i < j) {
    while (i < j && this->data[i] <= this->data[rightEdge])
      i++;

    while (i < j && this->data[j] >= this->data[rightEdge])
      j--;

    if (i != j)
      this->swapData(this->data[i], this->data[j]);
  }

  if (i != rightEdge)
    this->swapData(this->data[i], this->data[rightEdge]);

  // Divide y Vencerás
  this->sortDataQuick(leftEdge, i - 1);
  this->sortDataQuick(i + 1, rightEdge);
}
```

```
template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataQuick(const int& leftEdge,
                                       const int& rightEdge,
                                       int cmp(const T&, const T&)) {
  // Criterio de paro
  if (leftEdge >= rightEdge)
    return;

  if (cmp(this->data[leftEdge], this->data[rightEdge + 1]) == 0) {
    if (cmp(this->data[leftEdge], this->data[rightEdge]) > 0)
      this->swapData(this->data[leftEdge], this->data[rightEdge]);
    return;
  }

  // Separación
  int i(leftEdge), j(rightEdge);

  while (i < j) {
    while (i < j && cmp(this->data[i], this->data[rightEdge]) <= 0)
      i++;

    while (i < j && cmp(this->data[j], this->data[rightEdge]) >= 0)
      j--;

    if (i != j)
      this->swapData(this->data[i], this->data[j]);
  }

  if (i != rightEdge)
    this->swapData(this->data[i], this->data[rightEdge]);

  // Divide y Vencerás
  this->sortDataQuick(leftEdge, i - 1, cmp);
  this->sortDataQuick(i + 1, rightEdge, cmp);
}

template <class T, int ARRAYSIZE>
List<T, ARRAYSIZE>& List<T, ARRAYSIZE>::operator=(
    const List<T, ARRAYSIZE>& other) {
  copyAll(other);

  return *this;
}

template <class X>
std::ostream& operator<<(std::ostream& os, const List<X>& list) {
  int i = 0;
  while (i <= list.last)
```

```
    os << list.data[i++] << "," << std::endl;

  return os;
}

template <class X>
std::istream& operator>>(std::istream& is, List<X>& list) {
  X obj;
  std::string aux;

  try {
    while (is >> obj) {
      if (!list.isFull())
        list.data[++list.last] = obj;
    }
  } catch (const std::invalid_argument& ex) {
  }
  return is;
}

// Extras al Modelo de la Lista:
template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::findDataL(const T& searchedData,
                                  int cmp(const T&, const T&)) {
  for (int i = 0; i <= this->last; i++)
    if (cmp(searchedData, this->data[i]) == 0)
      return i;

  return -1;
}

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::findDataB(const T& searchedData,
                                  int cmp(const T&, const T&)) {
  int i(0), j(this->last), middle;

  while (i <= j) {
    middle = (i + j) / 2;

    if (cmp(searchedData, this->data[middle]) == 0)
      return middle;
    if (cmp(searchedData, this->data[middle]) < 0)
      j = middle - 1;
    else
      i = middle + 1;
  }

  return -1;
```

```
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::insertSortedData(const T& newData,
                                          int cmp(const T&, const T&)) {
  int i(0);

  while ((i <= this->last) && (cmp(newData, this->data[i]) > 0))
    i++;

  insertElement(newData, i);
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataBubble(int cmp(const T&, const T&)) {
  int i(this->last), j;
  bool flag;

  do {
    flag = false;
    j = 0;

    while (j < i) {
      if (cmp(this->data[j], this->data[j + 1]) > 0) {
        swapData(this->data[j], this->data[j + 1]);
        flag = true;
      }
      j++;
    }

    i--;
  } while (flag);
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataInsert(int cmp(const T&, const T&)) {
  int i(1), j;
  T aux;

  while (i <= this->last) {
    aux = this->data[i];

    j = i;
    while (j > 0 && cmp(aux, this->data[j - 1]) < 0) {
      this->data[j] = this->data[j - 1];

      j--;
    }
```

```
    if (i != j) {
      this->data[j] = aux;
    }

    i++;
  }
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataSelect(int cmp(const T&, const T&)) {
  int i(0), j, menor;

  while (i <= this->last) {
    menor = i;

    j = i + 1;

    while (j <= this->last) {
      if (cmp(this->data[j], this->data[menor]) < 0) {
        menor = j;
      }
      j++;
    }

    if (i != menor) {
      this->swapData(this->data[i], this->data[menor]);
    }
    i++;
  }
}

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataShell(int cmp(const T&, const T&)) {
  int series[] = {4181, 2584, 1597, 987, 610, 377, 233, 144, 89, 55,
                  34,   21,   13,   8,   5,   3,   2,   1,   0};
  int pos(0), dif(series[pos]), i, j;

  while (dif > 0) {
    i = dif;
    while (i <= this->last) {
      j = i;

      while (j >= dif && cmp(this->data[j - dif], this->data[j]) > 0) {
        this->swapData(this->data[j - dif], this->data[j]);
        j -= dif;
      }
```

```
            i++;
        }

        dif = series[++pos];
    }
}

#endif  // __LIST_H__
```

menu.hpp

```
#ifndef __MENU_H__
#define __MENU_H__

#include <iostream>

#include "configure.hpp"
#include "ownexceptions.hpp"
#include "sortingbenchmark.hpp"

class Menu {
 private:
  SortingBenchmark& comparator;

  void enterToContinue();
  int readInteger(std::string, const int&, const int&);
  char readChar(const std::string&, const char*);
  bool handleOption(const char&);
  std::string windowHeader(const int&, const std::string&) const;

  void invalidOption();

  void mainMenu();

  void runSingleBenchmark(std::string);
  void runAllBenchmarks();

 public:
  Menu();
  Menu(SortingBenchmark&);
  Menu(const Menu&);
};

#endif  // __MENU_H__
```

ownexceptions.hpp

```
#ifndef __OWNEXCEPTIONS_H__
#define __OWNEXCEPTIONS_H__

#include <stdexcept>
#include <string>

namespace DataContainersExceptions {
class MemoryDeficiency : public std::runtime_error {
 public:
  explicit MemoryDeficiency(const std::string& msg = "Insuficiencia de
Memoria")
      : std::runtime_error(msg) {}
};

class MemoryOverflow : public std::runtime_error {
 public:
  explicit MemoryOverflow(const std::string& msg = "Desbordamiento de
Memoria")
      : std::runtime_error(msg) {}
};

class InvalidPosition : public std::runtime_error {
 public:
  explicit InvalidPosition(
      const std::string& msg = "La posicion Ingresada es Invalida")
      : std::runtime_error(msg) {}
};
}  // namespace DataContainersExceptions

namespace InputExceptions {
class InvalidOption : public std::runtime_error {
 public:
  explicit InvalidOption(
      const std::string& msg = "La opcion ingresada esta fuera de rango")
      : runtime_error(msg) {}
};

class EmptyString : public std::runtime_error {
 public:
  explicit EmptyString(
      const std::string& msg = "El string no puede estar vacio")
      : runtime_error(msg) {};
};

class OperationCanceledException : public std::runtime_error {
 public:
  explicit OperationCanceledException(
```

```
        const std::string msg = "Operacion Cancelada")
      : runtime_error(msg) {}
};
}  // namespace InputExceptions

#endif  // __OWNEXCEPTIONS_H__
```

sortingbenchmark.hpp

```
#ifndef __SORTINGBENCHMARK_H__
#define __SORTINGBENCHMARK_H__

#include <chrono>
#include <iostream>
#include <random>
#include <string>

#include "benchmarkresult.hpp"
#include "configure.hpp"
#include "integer.hpp"
#include "list.hpp"

class SortingBenchmark {
 private:
  List<Integer, ListConfigure::ARRAYSIZE> originalList;
  List<Integer, ListConfigure::ARRAYSIZE> workingList;
  List<BenchMarkResult, ListConfigure::NumberAlgorithms> results;

  template <typename F>
  std::chrono::duration<double, std::milli> measureTime(F&&);

 public:
  SortingBenchmark();
  SortingBenchmark(
      const List<Integer, ListConfigure::ARRAYSIZE>&,
      const List<Integer, ListConfigure::ARRAYSIZE>&,
      const List<BenchMarkResult, ListConfigure::NumberAlgorithms>&);
  SortingBenchmark(const SortingBenchmark&);

  void generateRandomData(const int& = RandomConfigure::minimunRand,
                          const int& = RandomConfigure::maximunRand);

  void resetWorkingList();
  void clearResults();

  List<Integer, ListConfigure::ARRAYSIZE>& getOriginalList();
  List<Integer, ListConfigure::ARRAYSIZE>& getWorkingList();
  List<BenchMarkResult, ListConfigure::NumberAlgorithms>& getResults();

  // BenchMarck Individual
  template <typename SortFunc>
  BenchMarkResult runSingleBenchmark(const std::string&, SortFunc);

  void runAllBenchmarks();

  std::string getResultsTable() const;
```

```
    std::string getSingleTable(const BenchMarkResult&);

  SortingBenchmark& operator=(const SortingBenchmark&);
};

template <typename F>
std::chrono::duration<double, std::milli> SortingBenchmark::measureTime(
    F&& sortFunction) {
  auto start = std::chrono::high_resolution_clock::now();
  sortFunction();
  auto end = std::chrono::high_resolution_clock::now();
  return end - start;
}

template <typename SortFunc>
BenchMarkResult SortingBenchmark::runSingleBenchmark(
    const std::string& algorithmName,
    SortFunc sortFunc) {
  resetWorkingList();
  auto duration = this->measureTime([&]() { sortFunc(this->workingList);
});
  BenchMarkResult result(algorithmName, duration);
  try {
    this->results.insertElement(result, this->results.getLastPosition() +
1);
  } catch (const DataContainersExceptions::MemoryDeficiency&
              ex) {  // Si el historial de resultados está lleno, lo
vaciamos y
                     // añadimos el resultado
    this->clearResults();
    this->results.insertElement(result, this->results.getLastPosition() +
1);
  }
  return result;
}
#endif  // __SORTINGBENCHMARK_H__
```

Carpeta src

benchmarkresult.cpp

```
#include "benchmarkresult.hpp"

BenchMarkResult::BenchMarkResult() : algorithName("default"), duration(0)
{}

BenchMarkResult::BenchMarkResult(const std::string& a,
                                 std::chrono::duration<double,
std::milli> d)
    : algorithName(a), duration(d) {}

BenchMarkResult::BenchMarkResult(const BenchMarkResult& other)
    : algorithName(other.algorithName), duration(other.duration) {}

void BenchMarkResult::setAlgorithName(const std::string& algorithName) {
  this->algorithName = algorithName;
}

void BenchMarkResult::setDuration(
    const std::chrono::duration<double, std::milli> duration) {
  this->duration = duration;
}

std::string BenchMarkResult::toString() const {
  std::ostringstream oss;
  int nameWidth = 29;
  int msWidth = 19;
  int sWidth = 18;
  oss << "|" << std::left << std::setw(nameWidth) << this->algorithName
<< "|"
      << std::left << std::setw(msWidth) << this->duration.count() << "|"
      << std::left << std::setw(sWidth) << this->duration.count() / 1000
<< "|";
  return oss.str();
}

std::string BenchMarkResult::getAlgorithName() const {
  return this->algorithName;
}

std::chrono::duration<double, std::milli> BenchMarkResult::getDuration()
const {
  return this->duration;
}

bool BenchMarkResult::operator==(const BenchMarkResult& other) {
```

```
    return this->duration == other.duration;
}

bool BenchMarkResult::operator!=(const BenchMarkResult& other) {
  return this->duration != other.duration;
}

bool BenchMarkResult::operator<(const BenchMarkResult& other) {
  return this->duration < other.duration;
}

bool BenchMarkResult::operator>(const BenchMarkResult& other) {
  return this->duration > other.duration;
}

bool BenchMarkResult::operator<=(const BenchMarkResult& other) {
  return this->duration <= other.duration;
}

bool BenchMarkResult::operator>=(const BenchMarkResult& other) {
  return this->duration >= other.duration;
}

BenchMarkResult& BenchMarkResult::operator=(const BenchMarkResult& other)
{
  this->algorithmName = other.algorithmName;
  this->duration = other.duration;
  return *this;
}
```

integer.cpp

```cpp
#include <integer.hpp>

Integer::Integer() : value(0) {}

Integer::Integer(const Integer& other) : value(other.value) {}

Integer::Integer(const int64_t& v) : value(v) {}

Integer::Integer(const std::string& value) : value(std::stoi(value)) {}

int64_t Integer::getValue() const {
  return value;
}

std::string Integer::toString() const {
  return std::to_string(value);
}

std::string Integer::toString(const Integer& integer) {
  return integer.toString();
}

void Integer::setValue(const int64_t& v) {
  this->value = v;
}

Integer& Integer::operator=(const Integer& other) {
  if (this != &other)
    this->value = other.value;
  return *this;
}

Integer& Integer::operator=(const std::string& value) {
  this->value = std::stoi(value);
  return *this;
}

Integer& Integer::operator=(const int64_t& value) {
  this->value = value;
  return *this;
}

bool Integer::operator==(const Integer& other) const {
  return this->value == other.value;
}

bool Integer::operator!=(const Integer& other) const {
```

```
    return this->value != other.value;
}

bool Integer::operator<(const Integer& other) const {
  return this->value < other.value;
}

bool Integer::operator>(const Integer& other) const {
  return this->value > other.value;
}

bool Integer::operator<=(const Integer& other) const {
  return this->value <= other.value;
}

bool Integer::operator>=(const Integer& other) const {
  return this->value >= other.value;
}

int Integer::compare(const Integer& integerA, const Integer& integerB) {
  return integerA.value - integerB.value;
}

int Integer::compareTo(const Integer& other) {
  return this->value - other.value;
}

Integer Integer::operator+(const Integer& other) const {
  return Integer(this->value + other.value);
}

Integer Integer::operator-(const Integer& other) const {
  return Integer(this->value - other.value);
}

Integer Integer::operator*(const Integer& other) const {
  return Integer(this->value * other.value);
}

Integer Integer::operator/(const Integer& other) const {
  return Integer(this->value / other.value);
}

Integer Integer::operator%(const Integer& other) const {
  return Integer(this->value % other.value);
}

Integer Integer::operator+(const int64_t& value) const {
```

```
    return Integer(this->value + value);
}

Integer Integer::operator-(const int64_t& value) const {
  return Integer(this->value - value);
}

Integer Integer::operator*(const int64_t& value) const {
  return Integer(this->value * value);
}

Integer Integer::operator/(const int64_t& value) const {
  return Integer(this->value / value);
}

Integer Integer::operator%(const int64_t& value) const {
  return Integer(this->value % value);
}

Integer& Integer::operator++() {
  this->value++;
  return *this;
}

Integer Integer::operator++(int) {
  Integer temp(*this);
  ++value;
  return temp;
}

Integer& Integer::operator--() {
  this->value--;
  return *this;
}

Integer Integer::operator--(int) {
  Integer temp(*this);
  --value;
  return temp;
}

Integer& Integer::operator+=(const Integer& other) {
  this->value += other.value;
  return *this;
}

Integer& Integer::operator-=(const Integer& other) {
  this->value -= other.value;
```

```
    return *this;
}

Integer& Integer::operator*=(const Integer& other) {
  this->value *= other.value;
  return *this;
}

Integer& Integer::operator/=(const Integer& other) {
  this->value /= other.value;
  return *this;
}

Integer& Integer::operator%=(const Integer& other) {
  this->value %= other.value;
  return *this;
}

std::ostream& operator<<(std::ostream& os, const Integer& integer) {
  os << integer.getValue();
  return os;
}

std::istream& operator>>(std::istream& is, Integer& integer) {
  int64_t temp;
  is >> temp;
  if (is) {
    integer.setValue(temp);
  }
  return is;
}

Integer operator+(const int64_t& value, const Integer& integer) {
  return Integer(value + integer.value);
}

Integer operator-(const int64_t& value, const Integer& integer) {
  return Integer(value - integer.value);
}

Integer operator*(const int64_t& value, const Integer& integer) {
  return Integer(value * integer.value);
}

Integer operator/(const int64_t& value, const Integer& integer) {
  return Integer(value / integer.value);
}
```

```
Integer operator%(const int64_t& value, const Integer& integer) {
  return Integer(value % integer.value);
}

bool operator==(const int64_t& value, const Integer& integer) {
  return value == integer.value;
}

bool operator!=(const int64_t& value, const Integer& integer) {
  return value != integer.value;
}

bool operator<(const int64_t& value, const Integer& integer) {
  return value < integer.value;
}

bool operator>(const int64_t& value, const Integer& integer) {
  return value > integer.value;
}

bool operator<=(const int64_t& value, const Integer& integer) {
  return value <= integer.value;
}

bool operator>=(const int64_t& value, const Integer& integer) {
  return value >= integer.value;
}
```

main.cpp

```cpp
#include "menu.hpp"

int main() {
  Menu(*new SortingBenchmark);
  return 0;
}
```

menu.cpp

```
#include "menu.hpp"

using namespace std;
void Menu::enterToContinue() {
  cout << "[Enter] para continuar..." << endl;
  getchar();
}

int Menu::readInteger(string oss,
                      const int& lowerLimit,
                      const int& upperLimit) {
  string aux("");
  int result;
  while (true) {
    try {
      system("CLS");
      cout << oss;
      getline(cin, aux);
      result = stoi(aux);

      if (result > upperLimit || result < lowerLimit)
        throw InputExceptions::InvalidOption("Numero Fuera de Rango");
      break;
    } catch (const std::invalid_argument& ex) {
      system("CLS");
      cout << "Entrada invalida" << endl;
      cout << "Intente nuevamente" << endl;
      enterToContinue();
    } catch (const InputExceptions::InvalidOption& msg) {
      system("CLS");
      cout << msg.what() << endl;
      enterToContinue();
    }
  }

  return result;
}

char Menu::readChar(const std::string& prompt, const char* posibilities)
{
  char result, comparation;

  while (true) {
    int i = 0;
    system("CLS");
    cout << prompt;
    cin >> result;
```

```
    result = toupper(result);
    do {
      comparation = *(posibilities + i);
      if (result == comparation)
        return result;
      i++;
    } while (comparation != '\0');

    system("CLS");
    cout << "Opcion Invalida" << endl;
    cout << "Intentelo Nuevamente" << endl;
    system("PAUSE");
  }
}

bool Menu::handleOption(const char& op) {
  system("CLS");

  switch (op) {
    case 'A':
      this->comparator.generateRandomData();
      cout << "Dataset generado Correctamente!" << endl;
      this->enterToContinue();
      return true;
      break;
    case 'B':
      this->runSingleBenchmark("BubbleSort");
      return true;
      break;
    case 'C':
      this->runSingleBenchmark("InsertSort");
      return true;
      break;
    case 'D':
      this->runSingleBenchmark("SelectSort");
      return true;
      break;
    case 'E':
      this->runSingleBenchmark("ShellSort");
      return true;
      break;
    case 'F':
      this->runSingleBenchmark("MergeSort");
      return true;
      break;
    case 'G':
      this->runSingleBenchmark("QuickSort");
```

```
        return true;
      break;
    case 'H':
      this->runAllBenchmarks();
      return true;
      break;
    case 'S':
      cout << "Saliendo del Programa..." << endl;
      return false;
      break;
    default:
      this->invalidOption();
      return true;
      break;
  }
}

string Menu::windowHeader(const int& widthBorder, const string& prompt)
const {
  ostringstream oss;

  oss << left << setfill('=') << setw(widthBorder) << "" << endl;
  oss << setfill(' ');

  // Título de Ventana
  oss << setw(widthBorder / 2 - (prompt.size() / 2)) << "| " << prompt
      << setw((widthBorder / 2) - (prompt.size() / 2) - 2) << "" << "|"
<< endl;
  oss << setfill('-') << setw(widthBorder) << "" << endl;
  oss << setfill(' ');

  return oss.str();
}

void Menu::invalidOption() {
  system("CLS");
  cout << "+-----------------------------------------------------+" <<
endl;
  cout << "+                 Opcion Incorrecta                   +" <<
endl;
  cout << "+               Intentelo Nuevamente                  +" <<
endl;
  cout << "+-----------------------------------------------------+" <<
endl;
  this->enterToContinue();
}

void Menu::mainMenu() {
```

```
 ostringstream oss;
  bool repeater = true;
  char options[9] = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'S'}, op;

  do {
    oss.clear();
    oss.str("");
    system("CLS");
    oss << windowHeader(70, "BENCHMARKS DE ORDENAMIENTOS");
    oss << "Seleccione una de las siguientes opciones: " << endl;
    oss << "[A] Generar un nuevo dataset aleatorio." << endl;
    oss << "[B] Benchmark a BubbleSort" << endl;
    oss << "[C] Benchmark a InsertSort" << endl;
    oss << "[D] Benchmark a SelectSort" << endl;
    oss << "[E] Benchmark a ShellSort (Con secuencia Fibonacci)" << endl;
    oss << "** ** *Algoritmos Recursivos* ** **" << endl;
    oss << "[F] Benchmark a MergeSort" << endl;
    oss << "[G] Benchmark a QuickSort" << endl;
    oss << "[H] Realizar un Benchmark General" << endl << endl;
    oss << "[S] salir del Programa" << endl;
    oss << "Seleccione una opcion: ";

    op = this->readChar(oss.str(), options);
    cin.ignore();
    this->handleOption(op);
  } while (repeater);
}

void Menu::runSingleBenchmark(std::string algorithm) {
  system("CLS");
  cout << "Ordenando " << ListConfigure::ARRAYSIZE << " Elementos
Mediante "
       << algorithm << endl;
  if (!this->comparator.getWorkingList().isSorted()) {
    cout << "Desorden de la Lista Verificado" << endl;
    cout << "Ordenamiento por " << algorithm << " en proceso..." << endl;

    if (algorithm == "BubbleSort")
      this->comparator.runSingleBenchmark(
          "BubbleSort", [](List<Integer, ListConfigure::ARRAYSIZE>& list)
{
            list.sortDataBubble();
          });
    if (algorithm == "InsertSort")
      this->comparator.runSingleBenchmark(
          "InsertSort", [](List<Integer, ListConfigure::ARRAYSIZE>& list)
{
            list.sortDataInsert();
```

```
            });
    if (algorithm == "SelectSort")
      this->comparator.runSingleBenchmark(
          "SelectSort", [](List<Integer, ListConfigure::ARRAYSIZE>& list)
{
            list.sortDataSelect();
          });
    if (algorithm == "ShellSort")
      this->comparator.runSingleBenchmark(
          "ShellSort", [](List<Integer, ListConfigure::ARRAYSIZE>& list)
{
            list.sortDataShell();
          });
    if (algorithm == "MergeSort")
      this->comparator.runSingleBenchmark(
          "MergeSort", [](List<Integer, ListConfigure::ARRAYSIZE>& list)
{
            list.sortDataMerge();
          });
    if (algorithm == "QuickSort")
      this->comparator.runSingleBenchmark(
          "QuickSort", [](List<Integer, ListConfigure::ARRAYSIZE>& list)
{
            list.sortDataQuick();
          });

    cout << "Ordenamiento Terminado" << endl;
    cout << "Comprobando Correcto Ordenamiento..." << endl;
    if (this->comparator.getWorkingList().isSorted()) {
      cout << "Se comprobo que la lista esta ordenada correctamente" <<
endl;
      cout << "Resultados..." << endl << endl;
      cout << this->comparator.getSingleTable(
          *this->comparator.getResults().retrieve(
              this->comparator.getResults().getLastPosition()));
    } else
      cout << "El Ordenamiento no fue Exitoso" << endl;

  } else {
    cout << "El arreglo esta ordenado\n Se recomienda generar una nueva "
            "sucesion de numeros"
         << endl;
  }

  this->enterToContinue();
}

void Menu::runAllBenchmarks() {
```

```
    system("CLS");
    cout << "Ejecutando todos los ordenamientos..." << endl;
    cout << "Este proceso puede tardar un poco de tiempo..." << endl;
    this->comparator.runAllBenchmarks();
    cout << "Ordenamientos Concluidos." << endl;
    cout << "Resultados: " << endl;
    cout << this->comparator.getResultsTable();
    this->enterToContinue();
}

Menu::Menu() : comparator(*new SortingBenchmark) {
    this->mainMenu();
}

Menu::Menu(SortingBenchmark& c) : comparator(c) {
    this->mainMenu();
}

Menu::Menu(const Menu& other) : comparator(other.comparator) {
    this->mainMenu();
}
```

sortingbenchmark.cpp

```
#include "sortingbenchmark.hpp"

SortingBenchmark::SortingBenchmark() : originalList(), workingList(),
results() {
    this->generateRandomData();
}

SortingBenchmark::SortingBenchmark(const List<Integer,
ListConfigure::ARRAYSIZE>& o, const List<Integer,
ListConfigure::ARRAYSIZE>& w, const List<BenchMarkResult,
ListConfigure::NumberAlgorithms>& r) : originalList(o), workingList(w),
results(r) {}
SortingBenchmark::SortingBenchmark(const SortingBenchmark& other) :
originalList(other.originalList), workingList(other.workingList),
results(other.results) {}

void SortingBenchmark::generateRandomData(const int& minimunData, const
int& maximunData){
    this->originalList.deleteAll();

    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<int> dist(minimunData, maximunData);
    for(int i = 0; i < ListConfigure::ARRAYSIZE; i++)
        this->originalList.insertElement(Integer(dist(gen)), this-
>originalList.getLastPosition()+1);

    resetWorkingList();
}

void SortingBenchmark::resetWorkingList(){
    this->workingList = this->originalList;
}

void SortingBenchmark::clearResults(){
    this->results.deleteAll();
}

List<Integer, ListConfigure::ARRAYSIZE>&
SortingBenchmark::getOriginalList(){
    return this->originalList;
}

List<Integer, ListConfigure::ARRAYSIZE>&
SortingBenchmark::getWorkingList(){
    return this->workingList;
}
```

```
List <BenchMarkResult, ListConfigure::NumberAlgorithms>&
SortingBenchmark::getResults(){
    return this->results;
}

void SortingBenchmark::runAllBenchmarks(){
    this->clearResults();
    this->runSingleBenchmark("BubbleSort", [](List<Integer,
ListConfigure::ARRAYSIZE>& list) {list.sortDataBubble(); });
    this->runSingleBenchmark("InsertSort", [](List<Integer,
ListConfigure::ARRAYSIZE>& list) {list.sortDataInsert(); });
    this->runSingleBenchmark("SelectSort", [](List<Integer,
ListConfigure::ARRAYSIZE>& list) {list.sortDataSelect(); });
    this->runSingleBenchmark("ShellSort", [](List<Integer,
ListConfigure::ARRAYSIZE>& list) {list.sortDataShell(); });
    this->runSingleBenchmark("MergeSort", [](List<Integer,
ListConfigure::ARRAYSIZE>& list) {list.sortDataMerge(); });
    this->runSingleBenchmark("QuickSort", [](List<Integer,
ListConfigure::ARRAYSIZE>& list) {list.sortDataQuick(); });
}

std::string SortingBenchmark::getResultsTable() const{
    std::ostringstream oss;
    int nameBorder(30), durationBorder(20), widthBorder(nameBorder +
durationBorder*2);
    std::string tittle = "COMPARACION DE ALGORITMOS";

    //Cabecera
    oss << std::setfill('=') << std::setw(widthBorder) << "" <<
std::endl;
    oss << std::setfill(' ');
    oss << "|" << std::setw(((widthBorder - tittle.size()) / 2) + 2) <<
"" << tittle << std::setw((widthBorder - tittle.size()) / 2 - 2) << "|"
<< std::endl;
    oss << std::setfill('=') << std::setw(widthBorder) << "" <<
std::endl;
    oss << std::setfill(' ');

    oss << "|Algoritmo" << std::setw(nameBorder - 10) << "" << "|Duracion
en ms" << std::setw(durationBorder-15) << "" << "|Duracion en S" <<
std::setw(durationBorder-15) << "" << "|" <<std::endl;
    oss << std::setfill('-') << std::setw(widthBorder) << "" <<
std::endl;
    oss << std::setfill(' ');

    oss << this->results.toString();
```

```
     oss << std::setfill('=') << std::setw(widthBorder) << "" <<
std::endl;
    oss << std::setfill(' ');

    return oss.str();
}

std::string SortingBenchmark::getSingleTable(const BenchMarkResult&
benchmark){
    std::ostringstream oss;
    int widthBorder = 70;
    std::string tittle = "Resultados del BenchMark";
    oss << std::setfill('=') << std::setw(widthBorder) << "" <<
std::endl;
    oss << std::setfill(' ');
    oss << "|" << std::setw(((widthBorder - tittle.size()) / 2) + 2) <<
"" << tittle << std::setw((widthBorder - tittle.size()) / 2 - 2) << "|"
<< std::endl;
    oss << std::setfill('=') << std::setw(widthBorder) << "" <<
std::endl;
    oss << std::setfill(' ');

    oss << "| Elementos totales del arreglo: " <<
ListConfigure::ARRAYSIZE << std::setw(widthBorder - 32 -
std::to_string(ListConfigure::ARRAYSIZE).size()) << "|" <<std::endl;
    oss << std::setfill('-') << std::setw(widthBorder) << "" <<
std::endl;
    oss << std::setfill(' ');
    oss << "Algoritmo Usado: " << benchmark.getAlgorithName() <<
std::endl;
    oss << "Tiempo total de Ordenamiento en ms: " <<
benchmark.getDuration().count() << std::endl;
    oss << "Tiempo total de Ordenamiento en s: " <<
benchmark.getDuration().count() / 1000 << std::endl;
    oss << std::setfill('-') << std::setw(widthBorder) << "" <<
std::endl;
    oss << std::setfill(' ');

    return oss.str();
}
SortingBenchmark& SortingBenchmark::operator = (const SortingBenchmark&
other){
    this->originalList = other.originalList;
    this->workingList = other.workingList;
    this->results = other.results;

    return *this;
}
```

Ejecución del Programa

El programa inicia con la siguiente pantalla:

```
================================================================================
|                      BENCHMARKS DE ORDENAMIENTOS                             |
--------------------------------------------------------------------------------
Seleccione una de las siguientes opciones:
[A] Generar un nuevo dataset aleatorio.
[B] Benchmark a BubbleSort
[C] Benchmark a InsertSort
[D] Benchmark a SelectSort
[E] Benchmark a ShellSort (Con secuencia Fibonacci)
** ** *Algoritmos Recursivos* ** **
[F] Benchmark a MergeSort
[G] Benchmark a QuickSort
[H] Realizar un Benchmark General

[S] salir del Programa
Seleccione una opcion:
```

SortingBenchmark inicia ya con una lista desordenada con las características especificadas (100,000 integer en un rango desde 0 a 1,000,000), pero podemos crear un dataset aleatorio también.

Como los programas anteriores, tenemos validaciones de entrada de datos, por lo que si intentamos poner una entrada diferente a las que tenemos nos muestra el siguiente mensaje:

```
Opcion Invalida
Intentelo Nuevamente
Presione una tecla para continuar . . .
```

Así que empecemos con la opción [A] para generar un nuevo dataset aleatorio:

```
Dataset generado Correctamente!
[Enter] para continuar...
```

El proceso prácticamente no tarda nada, se nos avisa que se generó correctamente y regresamos al menú principal. Ahora, elijamos un algoritmo, como por ejemplo, la opción [B] para ordenar por BubbleSort:

```
Ordenando 100000 Elementos Mediante BubbleSort
Desorden de la Lista Verificado
Ordenamiento por BubbleSort en proceso...
```

Se nos avisa del tamaño del arreglo que estamos ordenando y por cuál método, se nos avisa también que se verificó que la lista está en desorden y da un mensaje de que el ordenamiento ha comenzado, cierto tiempo después que varía en función del tamaño del

arreglo y del algoritmo que hayamos seleccionado se nos muestran los resultados del ordenamiento:

```
Ordenando 100000 Elementos Mediante BubbleSort
Desorden de la Lista Verificado
Ordenamiento por BubbleSort en proceso...
Ordenamiento Terminado
Comprobando Correcto Ordenamiento...
Se comprobo que la lista esta ordenada correctamente
Resultados...

========================================================================
|                        Resultados del BenchMark                      |
========================================================================
| Elementos totales del arreglo: 100000                                |
------------------------------------------------------------------------
Algoritmo Usado: BubbleSort
Tiempo total de Ordenamiento en ms: 61888.1
Tiempo total de Ordenamiento en s: 61.8881
------------------------------------------------------------------------
[Enter] para continuar...
```

Podemos observar como se nos avisa que el ordenamiento ha terminado, se corrobora que la lista está ordenada y nos muestra en formato de tabla los resultados del benchmark, en este caso, BubbleSort tardó 61s en ordenar la lista, después de un enter, volvemos al menú principal.

Si ahora, en el menú principal intentamos seleccionar otro algoritmo como la opción [C] para InsertSort, nos aparecerá lo siguiente:

```
Ordenando 100000 Elementos Mediante InsertSort
El arreglo esta ordenado
 Se recomienda generar una nueva sucesion de numeros
[Enter] para continuar...
```

Se detecta que el arreglo está ordenado previo a empezar el algoritmo, por lo que recomienda generar una nueva sucesión de números aleatorios, así que tenemos que pasar nuevamente por la opción [A],

La pantalla de cada singleBenchmark es similar y todo se logra mediante una sola función que toma como parámetro una función, este por ejemplo es el resultado de InsertSort con otra sucesión mezclada:

```
Ordenando 100000 Elementos Mediante InsertSort
Desorden de la Lista Verificado
Ordenamiento por InsertSort en proceso...
Ordenamiento Terminado
Comprobando Correcto Ordenamiento...
Se comprobo que la lista esta ordenada correctamente
Resultados...

========================================================================
|                        Resultados del BenchMark                      |
========================================================================
| Elementos totales del arreglo: 100000                                |
------------------------------------------------------------------------
Algoritmo Usado: InsertSort
Tiempo total de Ordenamiento en ms: 20958.4
Tiempo total de Ordenamiento en s: 20.9584
------------------------------------------------------------------------
[Enter] para continuar...
```

Tardó 20 segundos en el ordenamiento.

Finalmente, la opción [H] para realizar un benchmark general muestra lo siguiente:

```
Ejecutando todos los ordenamientos...
Este proceso puede tardar un poco de tiempo...
```

Aquí no es necesario mezclar antes la lista, ya que antes del primer algoritmo y después de cada uno, la lista ordenada se le asignan los valores de la lista mezclada cada vez, para que cada uno de los algoritmos estén en igualdad de condiciones, y tras un determinado tiempo, se nos muestran los resultados de la comparativa de los seis algoritmos de ordenamiento representados de la siguiente manera:

```
Ejecutando todos los ordenamientos...
Este proceso puede tardar un poco de tiempo...
Ordenamientos Concluidos.
Resultados:
=============================================================================
|                          COMPARACION DE ALGORITMOS                        |
=============================================================================
|Algoritmo                    |Duracion en ms     |Duracion en S          |
-----------------------------------------------------------------------------
|BubbleSort                   |60795.2            |60.7952                |
|InsertSort                   |13089.1            |13.0891                |
|SelectSort                   |15130.6            |15.1306                |
|ShellSort                    |75.1376            |0.0751376              |
|MergeSort                    |30.031             |0.030031               |
|QuickSort                    |20.8633            |0.0208633              |
=============================================================================
[Enter] para continuar...
```

Tenemos el apartado de cada algoritmo, su duración en ms y su equivalencia en ms. Todos con los mismos valores en el mismo orden.

Lo interesante del programa es que añadir nuevos algoritmos de ordenamiento es sumamente fácil, con tener el algoritmo en la lista y añadir la condición en la clase menú, ya lo tenemos, ni modificar la tabla ni hacer grandes cambios.

Conclusiones

Crear este programa fue muy interesante, desde el principio conceptualice la función que se encarga de medir el tiempo como una función que recibe una función como parámetro para agilizar mucho la programación y el supuesto posterior mantenimiento (si bien es posible que no volvamos a utilizar este programa, como buena practica es dejar un código que no solo funcione, sino que sea legible y sostenible), pero el hecho de que los métodos de ordenamiento no son estáticos, tuve que explorar otras vías, y fue así como di con la solución de hacer las funciones correspondientes con plantillas para funciones y mandarlas a llamar como funciones lambda que sean ejecutables para la función.

Otra cosa que fue interesante fue conocer los bench mark's de los métodos de ordenamiento en función del tiempo que tardan en order un conjunto de datos, como hasta ahora trabajábamos con conjuntos de datos bastante pequeños (como de 70 objetos a lo mucho), los ordenamientos eran en menos de un segundo casi sin importar el método que usáramos y no podíamos ver la diferencia entre ello o por qué elegir uno sobre otro, con un salto a 100,000 elementos con valores que oscilan entre 0 y 1,000,000 es mucho más clara la diferencia, y como burbuja se queda bastante atrás en el ordenamiento; es sorprendente como los recursivos Merge y Quick aún con una lista relativamente grande pueden ordenarlo en fracciones de segundo, y lo que también es muy impresionante es que ShellSort, siendo un método iterativo y en costo computacional no tan demandante consigue casi equipararse a estos algoritmos, con este programa también podríamos modificarlo para evaluar el tiempo de ordenamiento de ShellSort dado diferentes sucesiones o factores; para este programa (y lo especifica en el menú) utilizamos la sucesión de Fibonacci que crece bastante, pero podríamos evaluarla contra otras sucesiones Knuth, Pratt o cualquier otra.

Esta evaluación de ordenamientos es sumamente interesante y nos da información para que como ingenieros en el procesamiento de la información, sepamos elegir el mejor algoritmo dado un sistema sobre el cual podamos estar manejando y las posibles limitaciones que tengan, ya que QuickSort puede ser muy rápido pero llega a ser computacionalmente costoso, y BubbleSort aunque contrarreste esto, con conjuntos de datos más grandes es muy ineficiente como vimos en esta comparativa.