ACTIVIDAD 07: **MÉTODOS DE ORDENAMIENTO RECURSIVOS**

ESTUDIANTE: **EFRAIN ROBLES PULIDO**

CODIGO: **221350095**

NOMBRE DE LA MATERIA: **ESTRUCTURAS DE DATOS I**

SECCIÓN: ***D12***

CLAVE: ***I5886***

FECHA: **DOMINGO 6 DE MARZO DE 2022**

Tabla de autoevaluación:

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

Introducción:

Después de leer las instrucciones de la plataforma, se reutilizo el programa de la clase Lista, en donde se le agrego el resto de los métodos de ordenamientos vistos en clase, como el MergeSort y el QuickSort, así como completar la implementación de la clase Entero, que el profesor nos compartió. Luego comprobé que mi algoritmo para generar los números aleatorios funcionara, después de que se iban agregando correctamente a mi lista como objetos de la clase Entero. Como mi generador de números aleatorios y el método para insertar los elementos están en mi método InputData de mi menu, para pasar a otro método llamado SortData, dentro de ella tendrá una iteración, que copiara la lista A a la lista B en cada iteración y dependiendo del valor de la iteración ejecutara un método diferente de ordenamiento mediante un switch case, donde antes de entrar al switch case se hace un punto en el tiempo transcurrido del programa para que cuando termine el switch case se restara el nuevo punto en el tiempo con el primer punto en el tiempo hecho para obtener el tiempo transcurrido que paso en el programa. Finalmente, se mostrará en pantalla mensajes en donde se nos avisa si está o no ordenada la lista, el método utilizado y el tiempo transcurrido en segundos.

Código fuente:

```
///Efrain Robles Pulido
#include "Menu.h"
#include "Integer.h"

#include <iostream>

int main() {
    List<Integer,100000 > listA, listB;
    Menu interfaz(listA, listB);
    interfaz.mainMenu();
    return 0;
    }


#ifndef MENU_H_INCLUDED
#define MENU_H_INCLUDED

#include "List.h"
#include "Integer.h"

#include <iostream>
#include <random>
#include <chrono>
#include <conio.h>
#include <functional>


class Menu {
    private:
        ///Atributos
        List<Integer,100000>* myListA,*myListB;

        ///Metodos Privados
        void InputData();
        void stateList();
        void sortData();
        void enterMenu();

    public:
        ///Constructor
        Menu(List<Integer,100000>&, List<Integer,100000>&);

        ///Metodo publico
        void mainMenu();
    };

#endif // MENU_H_INCLUDED


#include "Menu.h"

using namespace std;
using namespace std::chrono;
```

```
///Metodos privados
void Menu::InputData() {
    int i(0);
    Integer myInt;

    std::default_random_engine
generator(chrono::system_clock::now().time_since_epoch().count());
    std::uniform_int_distribution<long long int> distribution(0,1000000);
    auto randomInt = bind(distribution, generator);

    while(i<100000) {
        try {
            myInt.setInteger(randomInt());

            myListA->insertElement(myListA->getLastPos(),myInt);
            }
        catch(ListException ex) {
            cout << endl << "\t\tOcurrio un error" << endl;
            cout << "\t\tError de sistema:" << ex.what() << endl;
            enterMenu();
            break;
            }
        i++;
        }
    }

void Menu::stateList() {
    if (myListB->isSorted()) {
        cout<<"-> La Lista esta ordenada    -> ";
        }
    else {
        cout<<"-> La Lista NO esta ordenada  ";
        }
    }


void Menu::sortData() {
    int i(0);
    char charBuffer[6];

    while(i<6) {
        cout<<"Copiando Lista...    ";
        *myListB = *myListA;
        stateList();
        cout<<endl;

        time_point<system_clock> initTime(system_clock::now());
        switch (i) {
            case 0:
                cout<<"Ordenando por Sort Bubble  ";
                myListB->sortDataBubble();
                break;

            case 1:
                cout<<"Ordenando por Sort Insert  ";
                myListB->sortDataInsert();
                break;
```

```
            case 2:
                cout<<"Ordenando por Sort Select  ";
                myListB->sortDataSelect();
                break;

            case 3:
                cout<<"Ordenando por Sort Shell   ";
                myListB->sortDataShell();
                break;

            case 4:
                cout<<"Ordenando por Sort Merge   ";
                myListB->sortDataMerge();
                break;

            case 5:
                cout<<"Ordenando por Sort Quick   ";
                myListB->sortDataQuick();
                break;
            }
        duration<float> timeLapse(system_clock::now() - initTime);

        stateList();
        sprintf(charBuffer, "%.3f", timeLapse.count());
        cout<<"Tiempo medido: "<<charBuffer<<" segundos"<<endl<<endl;
        i++;
        }
    }

void Menu::enterMenu() {
    cout<< endl<< "Presiona cualquier tecla para continuar";
    getch();
    }

///Constructor
Menu::Menu(List<Integer,100000>& _listA, List<Integer,100000>& _listB):
myListA(&_listA), myListB(&_listB) {}

///Metodo publico
void Menu::mainMenu() {
    InputData();
    sortData();
    enterMenu();
    }


#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED

#include <iostream>
#include <string>
#include "ListException.h"

//Definicion
template <class T, int ARRAYSIZE = 1024>
class List {
```

```
    private:
        //Atributos
        T data[ARRAYSIZE];
        int last;

        //Metodos privados
        void copyAll(const List<T, ARRAYSIZE>&);//Guardara el objeto
creado a una copia
        bool isValidPos(const int&);//Si esta en el rango adecuado
        void swapData(T&, T&); //Metodo utilizado para los ordenamientos

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

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

        //Sobrecarga de operador
        List<T, ARRAYSIZE>& operator = (const List<T, ARRAYSIZE>&);

        //Metodos para una Lista
        bool isEmpty();
        bool isFull();

        void insertElement(const int&, const T&);
        void deleteElement(const int&);
        void deleteAllElement();
        T retrieve(const int&);

        int getFirstPos();
        int getLastPos();
        int getPreviousPos(const int&);
        int getNextPos(const int&);

        std::string toString();

        bool isSorted();

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

        void sortDataMerge();
        void sortDataQuick();


    };

///Implementacion
using namespace std;

///Metodos privados
template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::copyAll(const List<T, ARRAYSIZE>& l) {
```

```
    for(int i(0); i <= l.last; i++) {
        data[i] = l.data[i];
        }
    last = l.last;
    }

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isValidPos(const int& p) {
    return p >= 0 && p <= last;
    }

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

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

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


///Sobrecarga de operador
template <class T, int ARRAYSIZE>
List<T, ARRAYSIZE>& List<T, ARRAYSIZE>::operator = (const List<T,
ARRAYSIZE>& l) {
    copyAll(l);
    return *this;
    }

///Metodos para una Lista
template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isEmpty() {
    return last == -1;
    }

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

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::insertElement(const int& pos, const T&
newElement) {
    if(isFull()) {
        throw ListException("Desbordamiento de datos, insertElement");
        }

    if(pos != -1 && !isValidPos(pos)) {
        throw ListException("Posicion Invalida, insertElement");
```

```
        }

//Detectara si la posicion introducida esta dentro del rango la lista
creada */
    if(pos > -1 && pos < last) {
        int i(last);//Insercion en el punto de interes
        while(i >= pos) {
            data[i+1] = data[i];
            i--;
            }
        data[pos] = newElement;
        last++;

        }
    else {
//Insercion despues del punto de interes de la posicion -1 y ultima
posicion
        data[pos + 1] = newElement;
        last++;
        }
    }

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::deleteElement(const int& p) {
    if(!isValidPos(p)) {
        throw ListException("Posicion Invalida, deleteElement");
        }

    int i(p);
    while(i < last) {
        data[i] = data[i+1];
        i++;
        }
    last--;
    }

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

template <class T, int ARRAYSIZE>
T List<T, ARRAYSIZE>::retrieve(const int& p) {
    if(!isValidPos(p)) {
        throw ListException("Posicion Invalida, retrive");
        }
    return data[p];
    }

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getFirstPos() {
    if(isEmpty()) {
        return -1;
        }
    return 0;
    }
```

```
template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getLastPos() {
    return last;
    }

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getPreviousPos(const int& p) {
    if(isEmpty() or p < 1 or p > last) {
        return -1;
        }
    else {
        return p-1;
        }
    }

template <class T, int ARRAYSIZE>
int List<T, ARRAYSIZE>::getNextPos(const int& p) {
    if(isEmpty() or p < 0 or p > last-1) {
        return -1;
        }
    return p+1;
    }

template <class T, int ARRAYSIZE>
string List<T, ARRAYSIZE>::toString() {
    string myStr;
    for(int i(0); i <= last; i++) {
        myStr +=  data[i].toString() + ", ";
        }
    return myStr;
    }

template <class T, int ARRAYSIZE>
bool List<T, ARRAYSIZE>::isSorted() {
    int i(0);
    while(i<last) {
        if (data[i]>data[i+1]) {
            return false;
            }
        i++;
        }
    return true;
    }

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataBubble() {
    int i(last), j;
    bool flag;
    do {
        flag = false;
        j=0;

        while(j<i) {
            if(data[j] > data[j+1]) {
                swapData(data[j],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<=last) {
        aux = data[i];
        j=i;

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

            j--;
            }

        if(i != j) {
            data[j] = aux;
            }
        i++;
        }
    }

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

    while(i < last) {
        m = i;
        j = i + 1;

        while(j <= last) {
            if(data[j] < data[m]) {
                m = j;
                }
            j++;
            }
        if(i!=m) {
            swapData(data[i], data[m]);
            }
        i++;
        }
    }

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataShell() {
    float factor(0.5);
    int dif((last + 1)*factor),i,j;

    while(dif > 0) {
```

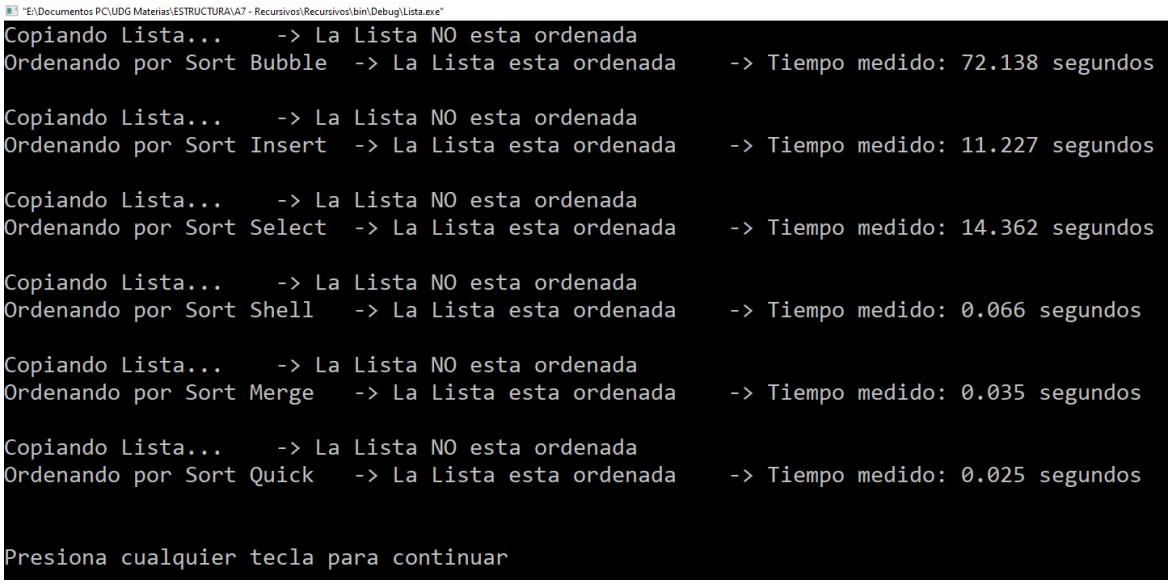
```
        i = dif;
        while(i <= last) {
            j=i;
            while(j >=dif and data[j-dif] > data[j]) {
                swapData(data[j-dif],data[j]);

                j-= dif;
                }
            i++;
            }
        dif*=factor;
        }
    }

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

template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataMerge(const int& leftEdge, const int&
rightEdge) {
    if (leftEdge >= rightEdge) { //Criterio de paro
        return;
        }
    int m((leftEdge + rightEdge)/2);//Divide y venceras

    sortDataMerge(leftEdge, m);
    sortDataMerge(m + 1,rightEdge);

    static T aux[ARRAYSIZE];//Copia al auxiliar
    for(int n(leftEdge); n<= rightEdge; n++) {
        aux[n] = data[n];
        }

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

    while(i <=m and j<= rightEdge) {
        while(i <=m and aux[i]<= aux[j]) {
            data[x++] = aux[i++];
            }
        if(i<=m) {
            while (j <= rightEdge and aux[j]<= aux[i]) {
                data[x++] = aux[j++];
                }
            }
        }
    while(i <= m) {
        data[x++] = aux[i++];
        }
    while(j <= rightEdge) {
        data[x++] = aux[j++];
        }
    }
```

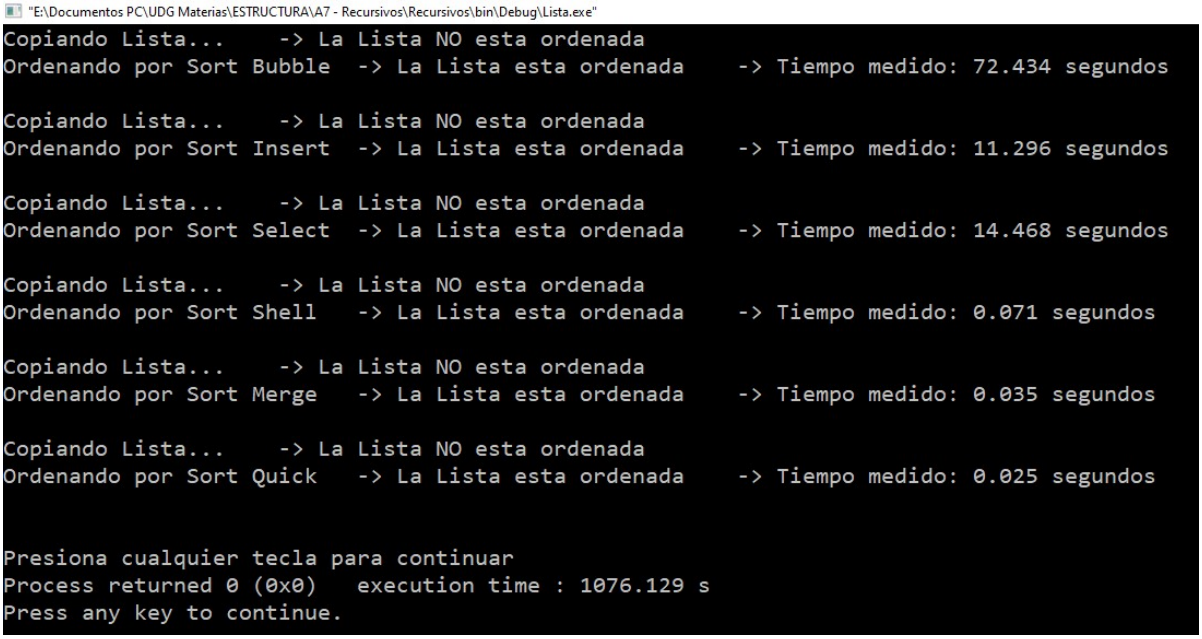
```
template <class T, int ARRAYSIZE>
void List<T, ARRAYSIZE>::sortDataQuick() {
    sortDataQuick(0,last);
    }

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

    while (i<j) {
        while(i < j and data[i] <= data[rightEdge]) {
            i++;
            }
        while (i<j and data[j] >= data[rightEdge]) {
            j--;
            }
        if (i != j) {
            swapData(data[i], data[j]);
            }
        }
    if(i!= rightEdge) {
        swapData(data[i], data[rightEdge]);
        }

    sortDataQuick(leftEdge, i - 1);//Divide y venceras
    sortDataQuick(i + 1, rightEdge);
    }

#endif // LIST_H_INCLUDED


#ifndef INTEGER_H_INCLUDED
#define INTEGER_H_INCLUDED

#include <iostream>
#include <string>

class Integer {
    private:
        long long int data;

    public:
        Integer();

        Integer(const Integer&);

        Integer(const long long int&);

        std::string toString() const;

        void setInteger(const long long int&);

        long long int getInteger()const;
```

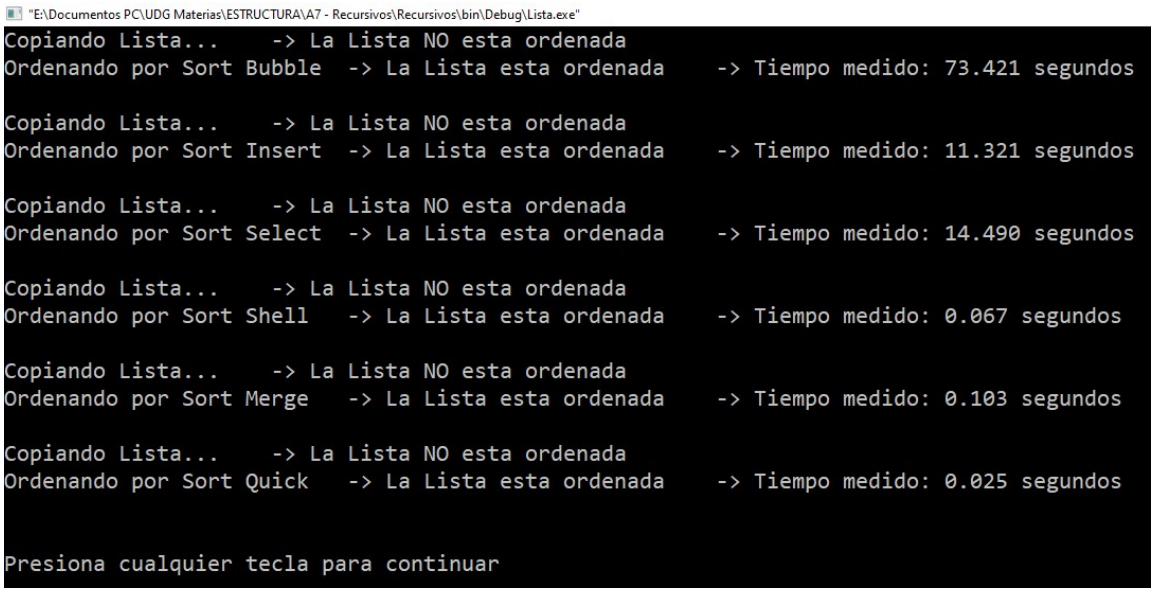
```
        Integer& operator ++ ();

        Integer operator ++ (int);

        Integer& operator -- ();

        Integer operator -- (int);

        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;

        Integer& operator = (const Integer&);

        Integer& operator += (const Integer&);

        Integer& operator -= (const Integer&);

        Integer& operator *= (const Integer&);

        Integer& operator /= (const Integer&);

        Integer& operator %= (const Integer&);

        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;

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

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

#endif // INTEGER_H_INCLUDED

#include <iostream>
#include <string>

#include "Integer.h"
```

```cpp
using namespace std;

///Costructor
Integer::Integer() {}

Integer::Integer(const Integer& _integer): data(_integer.data) {}

Integer::Integer(const long long int& _data) : data(0) {}

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

void Integer::setInteger(const long long int& i) {
    data = i;
    }

long long int Integer::getInteger() const {
    return data;
    }

Integer& Integer::operator++() {//Version prefija
    return *this;
    }

Integer Integer::operator++(int) {//Version postfija
    Integer old = *this;
    operator++();
    return old;
    }

Integer& Integer::operator--() {//Version prefija
    return *this;
    }

Integer Integer::operator--(int) {//Version postfija
    Integer old = *this;
    operator--();
    return old;
    }

Integer Integer::operator+(const Integer& i) const {
    return data + i.data;
    }

Integer Integer::operator-(const Integer& i) const {
    return data - i.data;
    }

Integer Integer::operator*(const Integer& i) const {
    return data * i.data;
    }

Integer Integer::operator/(const Integer& i) const {
    return data / i.data;
    }
```

```
Integer Integer::operator%(const Integer& i) const {
    return data % i.data;
    }

Integer Integer::operator-() const {
    return data;
    }

Integer& Integer::operator=(const Integer& i) {
    data = i.data;
    return *this;
    }

Integer& Integer::operator+=(const Integer& i) {
    data = data + i.data;
    return *this;
    }

Integer& Integer::operator-=(const Integer& i) {
    return *this = *this - i.data;
    }

Integer& Integer::operator*=(const Integer& i) {
    return *this = *this * i.data;
    }

Integer& Integer::operator/=(const Integer& i) {
    return *this = *this / i.data;
    }

Integer& Integer::operator%=(const Integer& i) {
    return *this = *this % i.data;
    }

bool Integer::operator==(const Integer& i) const {
    return data == i.data;
    }

bool Integer::operator!=(const Integer& i) const {
    return !(*this == i);
    }

bool Integer::operator<(const Integer& i) const {
    return data < i.data;
    }

bool Integer::operator<=(const Integer& i) const {
    return (*this < i) or (*this == i);
    }

bool Integer::operator>(const Integer& i) const {
    return !(*this <= i);
    }

bool Integer::operator>=(const Integer& i) const {
    return !(*this < i);
    }
```


"E:\Documentos PC\UDG Materias\ESTRUCTURA\A7 - Recursivos\Recursivos\bin\Debug\Lista.exe"
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Bubble -> La Lista esta ordenada   -> Tiempo medido: 72.138 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Insert -> La Lista esta ordenada   -> Tiempo medido: 11.227 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Select -> La Lista esta ordenada   -> Tiempo medido: 14.362 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Shell   -> La Lista esta ordenada   -> Tiempo medido: 0.066 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Merge   -> La Lista esta ordenada   -> Tiempo medido: 0.035 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Quick   -> La Lista esta ordenada   -> Tiempo medido: 0.025 segundos
Presiona cualquier tecla para continuar


"E:\Documentos PC\UDG Materias\ESTRUCTURA\A7 - Recursivos\Recursivos\bin\Debug\Lista.exe"
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Bubble -> La Lista esta ordenada   -> Tiempo medido: 72.434 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Insert -> La Lista esta ordenada   -> Tiempo medido: 11.296 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Select -> La Lista esta ordenada   -> Tiempo medido: 14.468 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Shell   -> La Lista esta ordenada   -> Tiempo medido: 0.071 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Merge   -> La Lista esta ordenada   -> Tiempo medido: 0.035 segundos
Copiando Lista...   -> La Lista NO esta ordenada
Ordenando por Sort Quick   -> La Lista esta ordenada   -> Tiempo medido: 0.025 segundos
Presiona cualquier tecla para continuar
Process returned 0 (0x0)   execution time : 1076.129 s
Press any key to continue.



Conclusión:

Esta práctica fue sencilla de realizar porque se le agrego dos métodos de ordenamiento que son recursivos en la clase lista de la práctica pasada, siendo el MergeSort y el QuickSort. Cuando hice mis pruebas para verificar que se generaban mis números aleatorios, así como para medir los tiempos, que me basé en del video del profesor para medir los tiempos transcurridos en el programa, y se me agregaban a mi lista, tuve problemas debido a que no estaba metiendo correctamente mi objeto Entero a mi lista que contendrá Enteros, debido a que estaba metiendo el atributo de la clase entero (data). Así que separe mis acciones por métodos, uno para crear mi lista con mis números aleatorios y meterlos a mi lista de Enteros, otro método para que se vayan ordenado según la iteración el método programado, en donde se me estará avisando si está o no ordenada mi lista. Además, tuve que investigar en cómo se debía de implementar la clase Entero debido a que se hacia otras sobrecargar de funciones que no conocía como el de incremento postfijo y el prefijo. Finalmente pude notar que los 2 últimos métodos de ordenamientos son los mas eficientes debidos a que son recursivos y dividimos nuestra lista lo mas posible para que sea más fácil el ordenamiento.