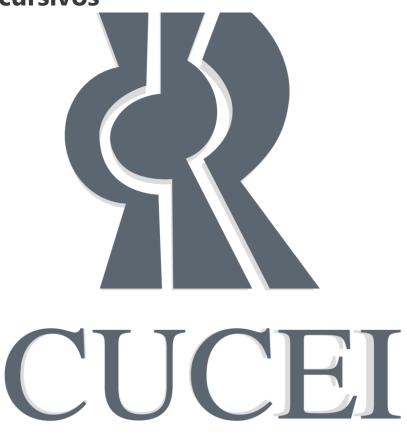
9-3-2019

Métodos de ordenamiento recursivos



david gutierrez alvarez

Estructura de datos I

RESUMEN PERSONAL Y FORMA DE ABORDAR EL PROBLEMA

En esta tarea lo que se me complico fue el echo de trabajar con números aleatorios, al principio no se generaban tan aleatorio y se repetina bastante haste que investigando encotre lo que me ayudo a solucionar este error y pude continuar, otro error que tuve fue que se congelaba en el método de ordenamiento merge, según lo que investigue era un desbordamiento de datos, pero lamentablemente no lo pude solucionar por lo que se que no tendre todos los puntos de esta actividad :'c

Esta actividad me agrado y me enfado bastante, empecé a encontrar error que nunca me habían pasado y fui conociendo nuevas funciones de las bibliotecas :3

```
Main.cpp
#include <iostream>
#include "menu.h"

using namespace std;

int main() {
    Menu menu;
    menu.init();

    return 0;
}
```

```
Menu.h
#ifndef MENU_H
#define MENU_H

#include <iostream>
#include "Rand.h"
#include "order.h"
#include "order.cpp"
#include <ctime>

class Menu {
public:
    Menu();
    void init();
};

#endif // MENU_H
```

```
Menu.cpp
#include "menu.h"
using namespace std;
Menu::Menu() { }
void Menu::init() {
    char option;
    long start, finish;
    do{
        cout << "\t\t\t\t\t.:ORDENAMINETO DE DATOS ALEATORIOS:." << endl</pre>
                   << "1.Ordenamiento Burbuja" << endl
                   << "2.Ordenamiento Seleccion" << endl</pre>
                   << "3.Ordenamiento Insersion" << endl
                   << "4.Ordenamiento Shell" << endl
                   << "5.Ordenamiento MergeSort "<< endl
                   << "6.Ordenamiento QuickSort" << endl</pre>
                   << "7.Salir" << endl
                   <<"Elige una opcion : ";</pre>
        cin >> option;
        Order < Rand, 100000 > numeros; /*genero los numeros aleatorios*/
        start = clock();/*inicializo el conteo del tiempo*/
        switch (option) {
            case '1': numeros.bubble();
            break;
            case '2': numeros.select();
```

```
break;
            case '3': numeros.insert();
            break;
            case '4': numeros.shell();
            break;
            case '5': numeros.merge();
            break;
            case '6': numeros.quick();
            break;
            case '7': cout << "Fin del programa." << endl;</pre>
            break;
        default: cout << "introduce un dato valido" << endl;</pre>
        if(option > '0' and option < '7') {/*asi solo me arroja el tiempo cuando uso
algun metodo de ordenamiento*/
            finish = clock(); /*despues de terminar el ordenamiento cuenta el tiempo de
nuevo*/
            numeros.print();/*imprime los numero*/
            double time = (double(finish-start)/CLOCKS PER SEC);
            cout << "el tiempo consumido en segundos es: " << time << endl;</pre>
    } while (option != '7');
```

```
Order.h
#ifndef ORDER H
#define ORDER H
#include <iostream>
template< class Type, const unsigned long int MAX>
class Order {
private:
    Type data[MAX];
    void merge(unsigned long, unsigned long);
    void quick(unsigned long, unsigned long);
public:
    Order();
    void sort(Type &, Type &);//ordena dos datos
    void bubble();
    void shell();
    void select();
    void insert();
    void merge();
    void quick();
    void print();
    int index();
};
#endif // ORDER_H
```

```
Order.cpp
#include "order.h"
```

```
template<class Type, const unsigned long int MAX>
Order<Type, MAX>::Order() { }
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::sort(Type &a, Type &b) {
    Type aux;
    aux = a;
    a = b;
    b = aux;
}
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::bubble() {
    int i = MAX, j;
    bool band;
    do{
        band = false;
        j=0;
        while(j < i) {</pre>
            if(data[j] > data[j+1]) {
                sort(data[j], data[j+1]);
                band = true;
            j++;
        i--;
    } while (band);
}
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::shell() {
    float fact = 3.0/4.0;
    unsigned long dif = MAX*fact;
    while(dif > 0) {
        for(unsigned long i=0; i <= MAX-dif; i++) {</pre>
            if(data[i] > data[i+dif]) {
                sort(data[i],data[i+dif]);
        dif*=fact;
    }
}
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::select() {
    unsigned long i = 0, j, menor;
    while(i < MAX-1) {</pre>
        menor = i;
        j = i+1;
        while(j < MAX) {</pre>
            if(data[j] < data[menor]){</pre>
               menor = j;
            j++;
        if(menor != i) {
           sort(data[i],data[menor]);
        i++;
```

```
}
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::insert() {
    unsigned long i = 1, j;
    Type element;
    while(i < MAX) {</pre>
        element = data[i];
        j = i;
        while(j > 0 and element < data[j-1]) {</pre>
            data[j] = data[j-1];
             j--;
        if(i != j) {
            data[j] = element;
        }
        i++;
    }
}
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::merge() {
    merge(0, MAX);
}
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::merge(unsigned long left, unsigned long right) {
    if(left >= right) {
       return;
    Type temp[MAX];
    for(unsigned long i(left); i <= right; i++){</pre>
       temp[i] = data[i];
    unsigned long medio((left+right)/2);
    merge(left, medio);
    merge(medio+1, right);
    unsigned long i(left), j(medio+1), x(left);
    while(i<=medio and j<=right) {</pre>
        while(i<=medio and temp[i] <= temp[j]){</pre>
            data[x++]=temp[i++];
        if(i<=medio) {</pre>
             while(j<=right and temp[j]<=temp[i]) {</pre>
            data[x++]=temp[j++];
    while(i<=medio) {</pre>
        data[x++] = temp[i++];
    while(j<=right) {</pre>
        data[x++] = temp[j++];
}
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::quick() {
    quick(0, MAX);
```

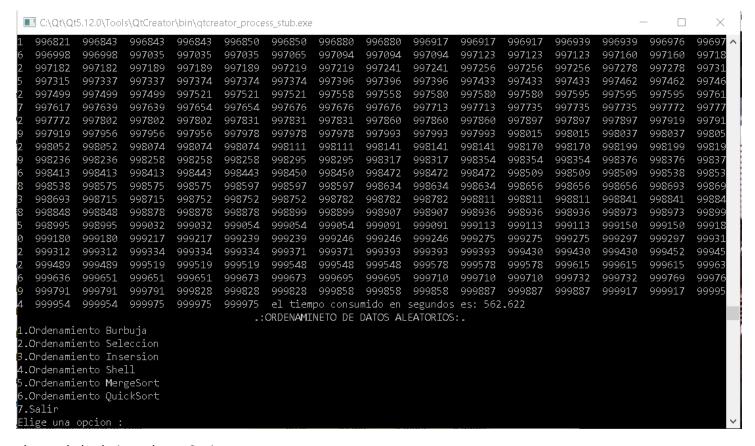
```
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::quick(unsigned long left, unsigned long right) {
    if(left>=right) {
        return;
    }
    unsigned long i(left), j(right);
    unsigned long pivote((left+right)/2);
    sort(data[pivote], data[right]);
    while(i < j) {</pre>
        while(i < j and data[i] <= data[right]) {</pre>
            i++;
        while(i < j and data[j] >= data[right]) {
            j--;
        }
        if(i != j) {
            sort(data[i], data[j]);
        }
    }
    if(i != right) {
        sort(data[i],data[right]);
    quick(left, i-1);
    quick(i+1, right);
}
template<class Type, const unsigned long int MAX>
void Order<Type, MAX>::print() {
    for(unsigned long i(0); i < MAX; i++) {</pre>
        std::cout<< data[i] << " ";</pre>
}
template<class Type, const unsigned long int MAX>
int Order<Type, MAX>::index() {
    return MAX;
```

```
Rand.h
#ifndef DATA H
#define DATA H
#include <iostream>
class Rand {
private:
    long value;
public:
    Rand();
    Rand(const Rand &);
    long redo();
    Rand operator = (const Rand &elem);
    bool operator == (const Rand &elem)const;
    bool operator != (const Rand &elem) const;
    bool operator < (const Rand &elem) const;</pre>
    bool operator > (const Rand &elem) const;
```

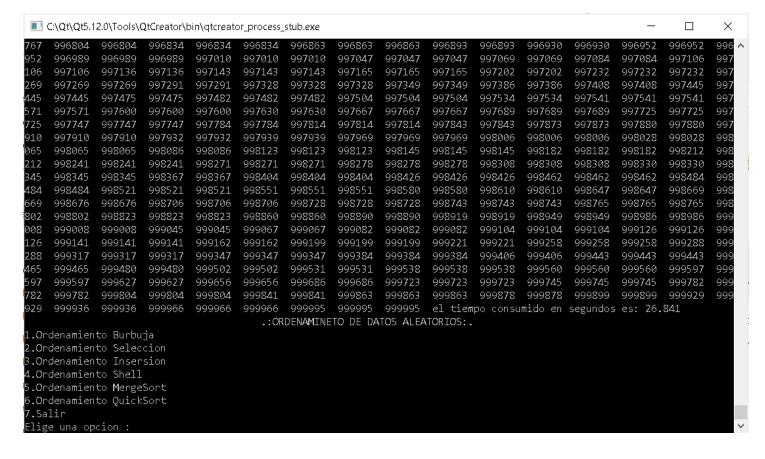
```
bool operator <= (const Rand &elem)const;
bool operator >= (const Rand &elem)const;
    //Serializar el objeto
    friend std::ostream &operator << (std::ostream& os, Rand& elem);
};
#endif // DATA_H</pre>
```

```
Rand.cpp
#include "Rand.h"
#include <random>
#include <chrono>
Rand::Rand() : value(redo()) { }
Rand::Rand(const Rand &copy) : value(copy.value) { }
long Rand::redo() {
    std::default random engine
engine{std::chrono::steady clock::now().time since epoch().count()};
    std::uniform int distribution<int> range{0, 1000000};
    long random generated = range(engine);
    return random generated;
}
Rand Rand::operator = (const Rand &element) {
    value = element.value;
    return *this;
}
bool Rand::operator ==(const Rand &element) const {
    return this->value ==element.value;
}
bool Rand::operator != (const Rand &element) const {
    return this->value !=element.value;
}
bool Rand::operator <=(const Rand &element) const {</pre>
    return this->value <=element.value;</pre>
bool Rand::operator >= (const Rand &element) const {
    return this->value >=element.value;
bool Rand::operator <(const Rand &element) const {</pre>
    return this->value < element.value;</pre>
bool Rand::operator > (const Rand &element) const {
    return this->value > element.value;
}
std::ostream& operator << (std::ostream& os, Rand &element) {</pre>
os << element.value << " ";
 return os;
```

CAPTURAS DE PANTALLA

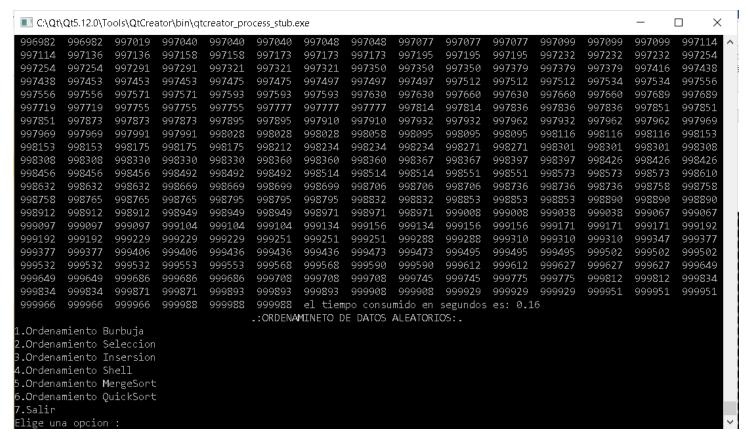


el metodo burbuja tarda cas 6 minutos





Inserccion tarda mas de medio minuto



	C:\Qt\Qt5.1	2.0\Tools\¢	QtCreator\b	oin\qtcreato	or_process_	stub.exe							_		×
054	997054	997061	997061	997061	997091	997091	997091	997128	997128	997150	997150	997150	997187	997187	997 ^
187	997209	997209	997246	997246	997246	997268	997268	997268	997304	997304	997304	997334	997334	997334	997
363	997363	997363	997393	997393	997393	997400	997400	997430	997430	997452	997452	99 74 59	99 74 59	99 74 59	997
489	997489	99 748 9	99 7 511	997511	997511	99 7 526	997526	997526	997548	997548	997548	99 7 585	997585	99 7 585	997
607	997607	997607	997644	997644	997644	997665	997665	997665	997702	997702	997702	997732	997732	997732	997
769	997769	997791	997791	997828	997828	99 78 50	99 78 50	997865	997865	997887	997887	997887	997909	997909	997
909	997924	997924	997924	997946	997946	997946	997983	997983	997983	998004	998004	998004	998041	998041	998
071	998071	998071	998100	998100	998100	998130	998130	998130	998167	998167	998167	998189	998189	998226	998
226	998248	998248	998248	998263	998263	998263	998285	998285	998285	998307	998307	998307	998322	998322	998
344	998344	998344	998380	998380	998380	99 84 10	99 84 10	998410	998439	998439	99 84 39	99 84 69	99 84 69	99 84 69	998
506	998506	998528	998528	998528	998565	998565	998565	998587	998587	998587	998624	998624	998646	998646	998
646	998661	998661	998661	998683	998683	998712	998712	998712	998720	998720	998720	998741	998741	998741	998
778	998778	998808	998808	998808	998845	998845	998867	998867	998867	998904	998904	998904	998926	998926	998
963	998963	998985	998985	998985	999022	999051	999051	999051	999059	999059	999081	999081	999081	999110	999
110	999110	999117	999117	999117	999147	999147	999176	999176	999176	999206	999206	999243	999265	999243	999
265	999265	999302	999302	999324	999324	999361	999324	999361	999361	999383	999383	999383	999420	999 44 9	999
449	999449	999457	999457	999457	999486	999486	999486	999508	999508	999508	999515	999515	999515	999545	999
545	999582	999582	999582	999604	999604	999604	999641	999641	999641	999663	999663	999663	999700	999700	999
700	999722	999722	999722	999 7 59	999759	999 7 59	999788	999788	999817	999817	999817	999847	999847	999854	999
854	999884	999884	999906	999906	999906	999921	999921	999921	999943	999943	999943	999980	999980	999980	el
tiem	tiempo consumido en segundos es: 0.151														
	.:ORDENAMINETO DE DATOS ALEATORIOS:.														
	1.Ordenamiento Burbuja														
2.Ordenamiento Seleccion															
3.Ordenamiento Insersion															
	denamien														
	denamien	_													
	denamien	to Quick	Sort												
7.Salir															
Elig	e una op	cion :													~

quickSort duro un poco menos

