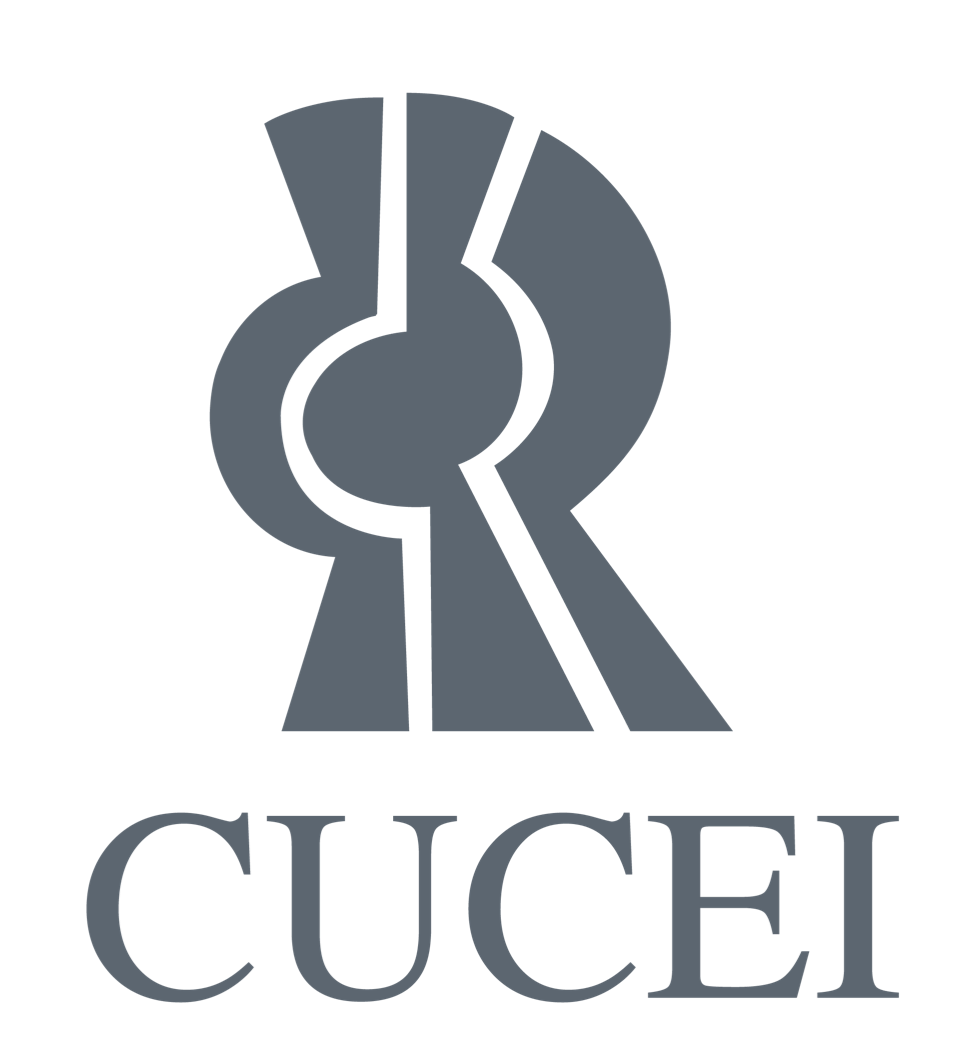
9-3-2019

**david gutierrez alvarez**

**Estructura de datos I**

****

## Métodos de ordenamiento recursivos

|  |
| --- |
| **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**  **<< "1.Ordenamiento Burbuja" << endl**  **<< "2.Ordenamiento Seleccion" << endl**  **<< "3.Ordenamiento Insersion" << endl**  **<< "4.Ordenamiento Shell" << endl**  **<< "5.Ordenamiento MergeSort "<< endl**  **<< "6.Ordenamiento QuickSort" << endl**  **<< "7.Salir" << endl**  **<<"Elige una opcion : ";**  **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;**  ***break*;**  ***default*: cout << "introduce un dato valido" << endl;**  **}**  ***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;**  **}**  **} *while* (option != '7');**  **}** |

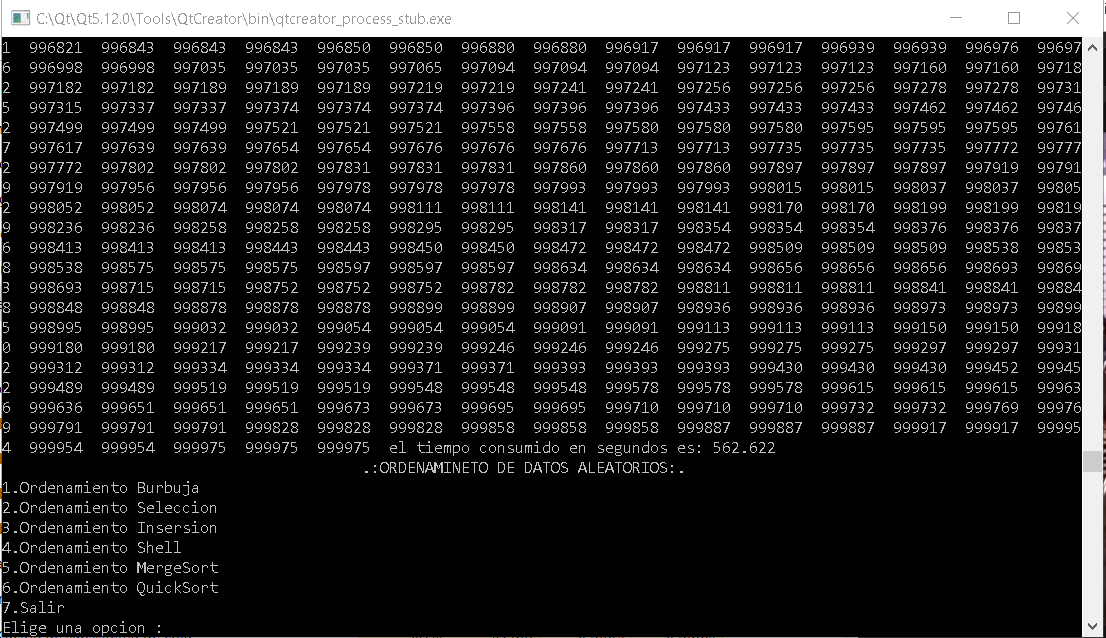
|  |
| --- |
| **Order.h** |
| **#ifndef ORDER\_H**  **#define ORDER\_H**  **#include <iostream>**  ***template*< *class* Type, *const* unsigned long int MAX>**  ***class* Order {**  ***private*:**  **Type data[MAX];**  **void merge(unsigned long, unsigned long);**  **void quick(unsigned long, unsigned 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) {**  ***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++) {**  ***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) {**  **menor = i;**  **j = i+1;**  ***while*(j < MAX) {**  ***if*(data[j] < data[menor]){**  **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) {**  **element = data[i];**  **j = i;**  ***while*(j > 0 *and* element < data[j-1]) {**  **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++){**  **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){**  ***while*(i<=medio *and* temp[i] <= temp[j]){**  **data[x++]=temp[i++];**  **}**  ***if*(i<=medio){**  ***while*(j<=right *and* temp[j]<=temp[i]){**  **data[x++]=temp[j++];**  **}**  **}**  **}**  ***while*(i<=medio){**  **data[x++] = temp[i++];**  **}**  ***while*(j<=right){**  **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) {**  ***while*(i < j *and* data[i] <= data[right]) {**  **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++) {**  **std::cout<< data[i] << " ";**  **}**  **}**  ***template*<*class* Type, *const* unsigned long int MAX>**  **int Order<Type, MAX>::index() {**  ***return* MAX;**  **}** |

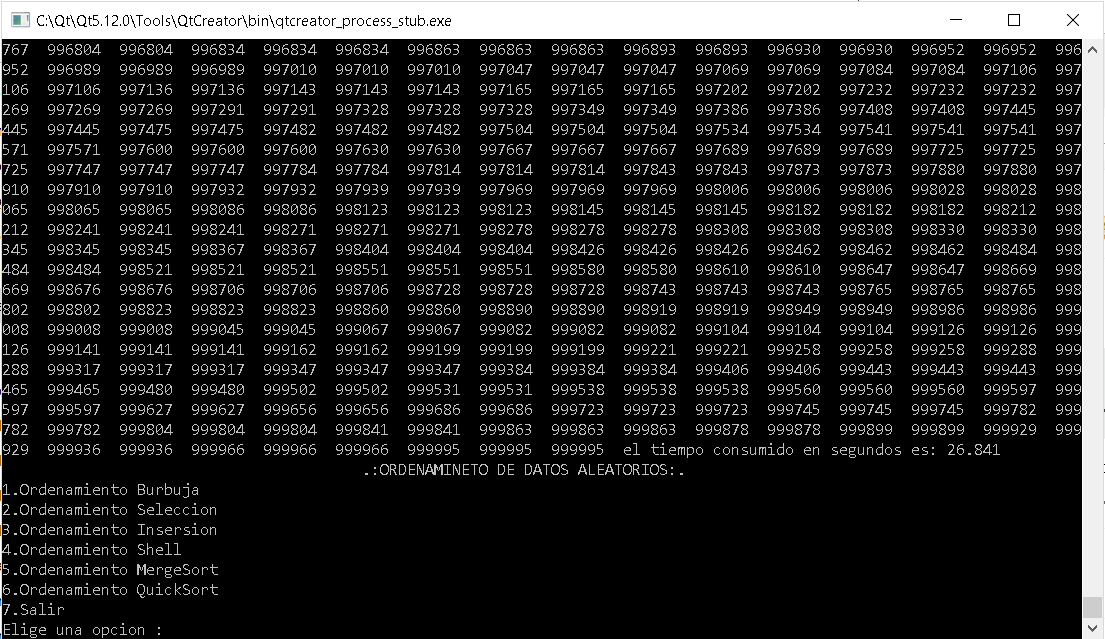
|  |
| --- |
| **Rand.h** |
| **#ifndef DATA\_H**  **#define DATA\_H**  **#include <iostream>**  ***class* Rand {**  ***private*:**  **long value;**  ***public*:**  **Rand();**  **Rand(*const* Rand &);**  **long redo();**  **Rand *operator* = (*const* Rand &elem);**  **bool *operator* == (*const* Rand &elem)*const*;**  **bool *operator* != (*const* Rand &elem)*const*;**  **bool *operator* < (*const* Rand &elem)*const*;**  **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*** |

|  |
| --- |
| **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* {**  ***return* *this*->value <=element.value;**  **}**  **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* {**  ***return* *this*->value > element.value;**  **}**  **std::ostream& *operator* << (std::ostream& os, Rand &element){**  **os << element.value << " ";**  ***return* os;**  **}** |

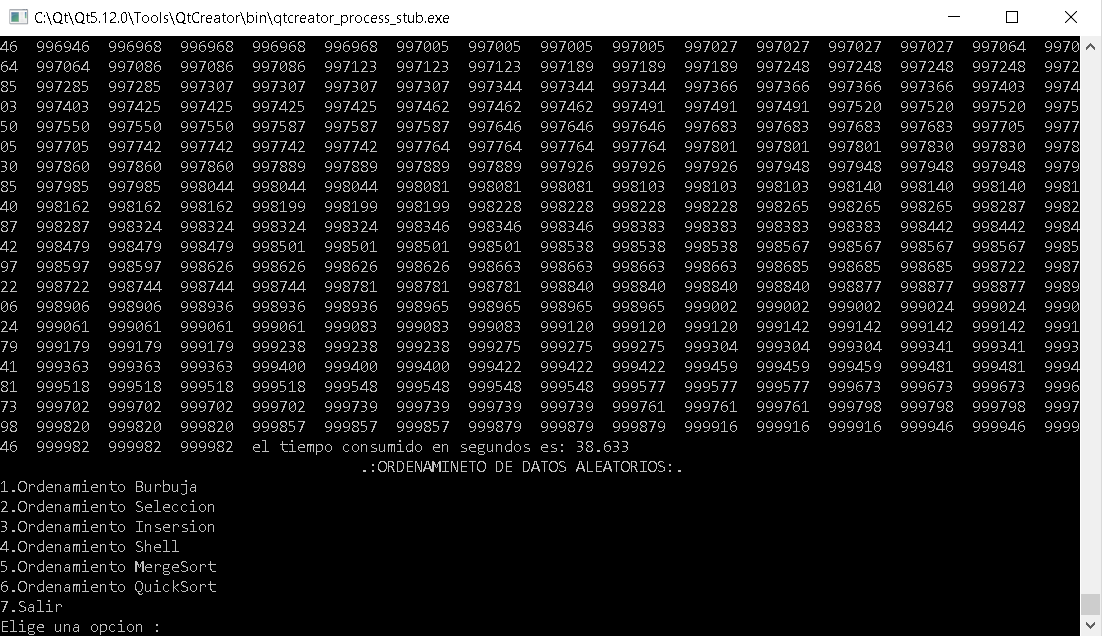
|  |
| --- |
| **CAPTURAS DE PANTALLA** |



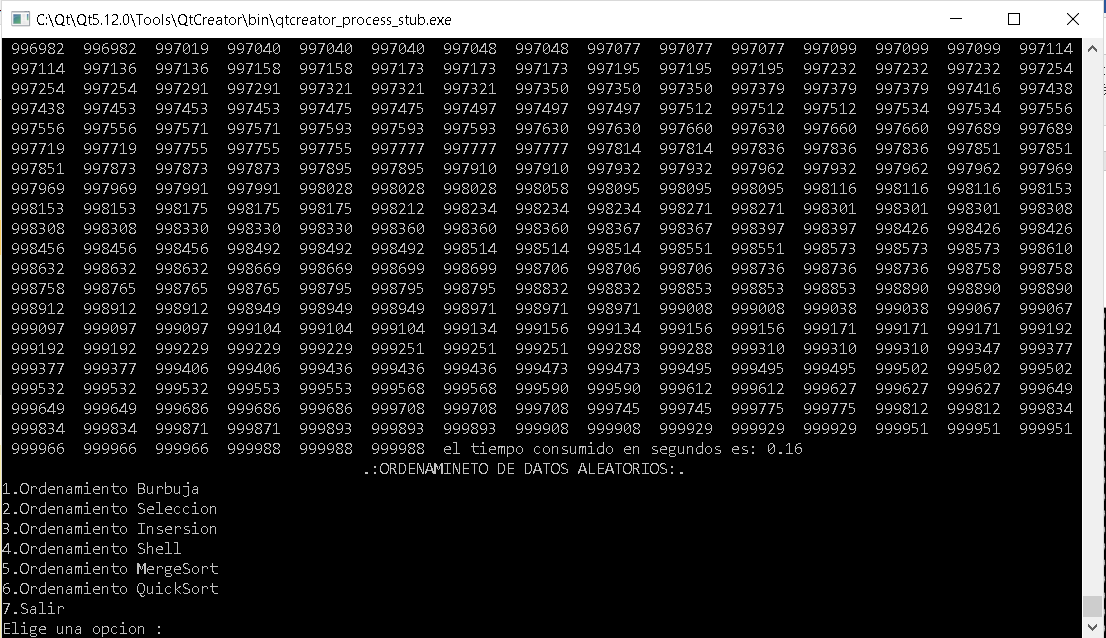
el metodo burbuja tarda cas 6 minutos



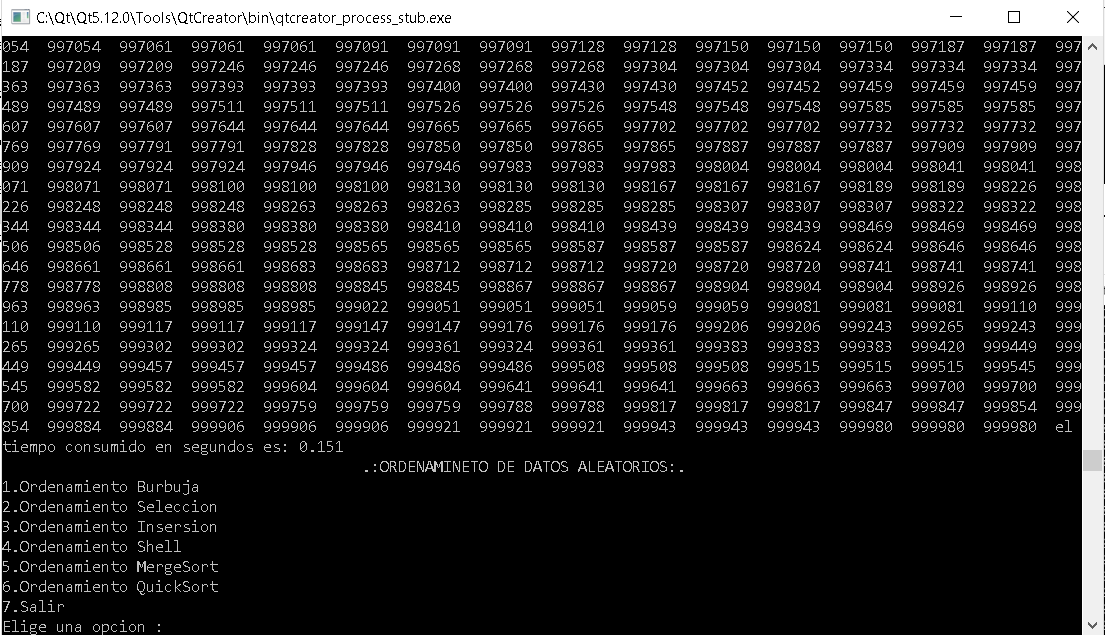
El método de selección tarda un poco menos de medio minuto



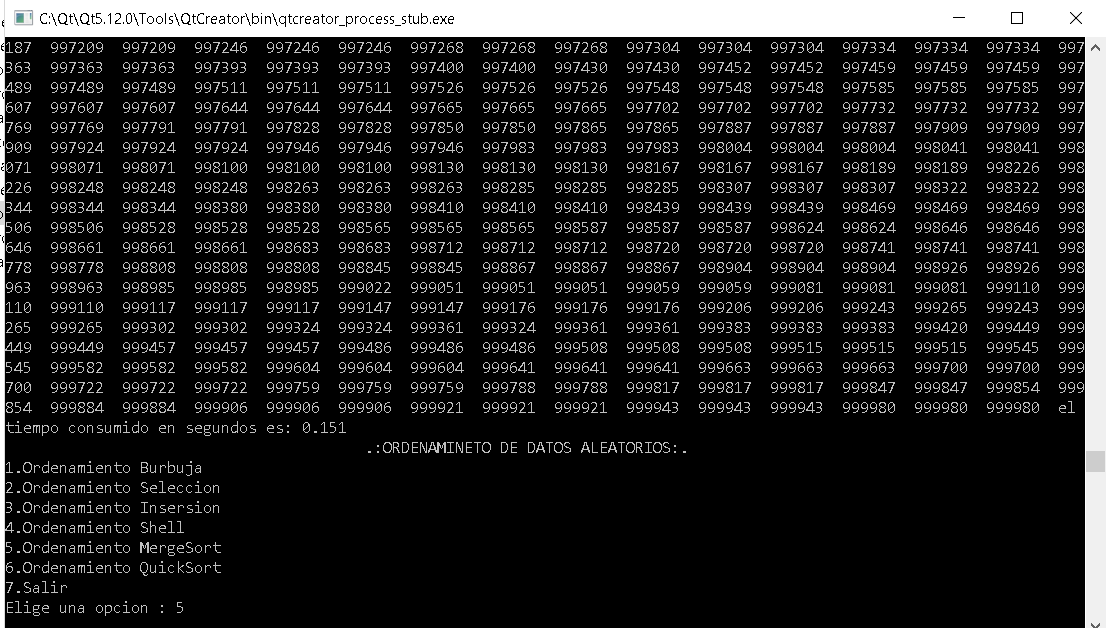
Inserccion tarda mas de medio minuto



Shell lo a echo en menos de un segundo



quickSort duro un poco menos



Y mergeSort no dejaba de explotar, solo funciono al ordenar una cantidad menor de datos