PRACTICA 3

Pucho Zevallos Kelvin Paul

Abril 2020

1. Analisis de Algoritmos

Para la comparación de los algoritmos de ordenamiento de insercion y burbuja se utilizo una herramienta de graficos, en este caso se utilizo excel.

ARREGLOS ALEATORIOS

```
#include<iostream>
#include <chrono>
#include <ctime>

using namespace std;

class Timer
{
public:
    void start()
    {
        m_StartTime = chrono::system_clock::now();
        m_bRunning = true;
    }

    void stop()
    {

        m_EndTime = chrono::system_clock::now();
        m_bRunning = false;
    }

    double elapsedMilliseconds()
    {
        chrono::time_point<std::chrono::system_clock> endTime;

        if(m_bRunning)
        {
            endTime = chrono::system_clock::now();
        }
        else
        {
            endTime = m_EndTime;
        }
```

```
        return chrono::duration_cast<chrono::nanoseconds>(endTime -
            m_StartTime).count();
    }

    double elapsedSeconds()
    {
        return elapsedMilliseconds() / 1000000.0;
    }

private:
    chrono::time_point<chrono::system_clock> m_StartTime;
    chrono::time_point<chrono::system_clock> m_EndTime;
    bool m_bRunning = false;
};
int* generadors(int tam){
    int* arreglo = new int[tam];
    srand(static_cast<unsigned int>(time(0)));
    int ran;

    for(int i=0;i<tam;i++){
        ran=0+rand()%10000;
        arreglo[i] = ran;
    }

    return arreglo;
}

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
```

```
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}


void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = (r+l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}

void insert_sort(int* arreglo,int tam){
    int j;
    for(int i=1;i<tam;i++){
        int clave = arreglo[i];
        j=i-1;

        while(j>=0 && arreglo[j]>clave){
            arreglo[j+1]=arreglo[j];
            j=j-1;
        }
        arreglo[j+1]=clave;

    }
    cout<<endl;
}


int main(){

    int cont=0;
    int fin;
    int ini;
    cout<<"coloque el inicio de la reiteracion: ";
    cin>>ini;
    cout<<"coloque el final de la reiteracion: ";
    cin>>fin;

    for(int tam = ini;tam<=fin;tam++){
```

```
        cout<<"\nPara tama o de "<<tam<<endl;

        int* arreglo = generadors(tam);

        int* arreglo2 = new int[tam];
        for(int i=0;i<tam;i++){
            arreglo2[i] = arreglo[i];
        }

        cout<<endl;

        //INSERT_SORT
        Timer timer;
        timer.start();
        insert_sort(arreglo,tam);
        timer.stop();

        cout << "Milliseconds: " << timer.elapsedSeconds() <<endl;
        cout << "Nanoseconds: " << timer.elapsedMilliseconds() <<endl;
        auto in = timer.elapsedMilliseconds();


        //MERSE_SORT

        Timer timer1;
        timer1.start();
        mergeSort(arreglo2, 0, tam - 1);
        timer1.stop();

        cout<<endl;

        cout << "Milliseconds: " << timer1.elapsedSeconds() <<endl;
        cout << "Nanosegundos: " << timer1.elapsedMilliseconds() <<endl
            ;
        auto in2= timer1.elapsedMilliseconds();

        if(in < in2){
            cout<<timer.elapsedMilliseconds()<<" nanosegundos de insert
                es menor que el "<<timer1.elapsedMilliseconds()<<"
                nanosegundos del merge\n";
            cont++;
        }
        delete []arreglo;
        delete []arreglo2;

        }
        cout<<cont<<endl;

        return 0;
    }
}
```

Tamaño	Tiempo Real en nanosegundos	Tiempo Real en nanoegundos
Tamaño de arreglos	Insert Sort	Merge Sort sort
40	13092	19791
41	12258	17534
43	14977	16253
45	14995	16828
46	11413	17092
50	12652	18690
52	13319	21708
57	16501	43511
59	16292	18641
61	16396	19621
62	17000	18182
65	19937	21559
67	18964	21417
69	19254	22766
72	33208	43631
73	20962	26051
79	24011	24292
81	24893	27156
89	28832	31010
91	29087	34425
101	19119	21934
104	19058	21335
109	23121	30170
113	23749	27507
117	21905	24321
121	21796	22454
123	18364	18587
125	23722	27791
157	68267	79724
169	80567	86643
180	26443	34011
197	23772	32541
198	25162	17015
200	23758	16524

2. Diagrama de la tabla

Existen algunos arreglos de tamaños menores a 198, donde al ser ordenados por el algoritmo merge sort se demoran mas o un poco mas que ser ordenados por el algoritmo insert sort como se muestra en la figura 1 pero partir de un arreglo de tamaño 200 el algoritmo merge sort, realiza el ordenamiento en menor tiempo que el algoritmo insert sort

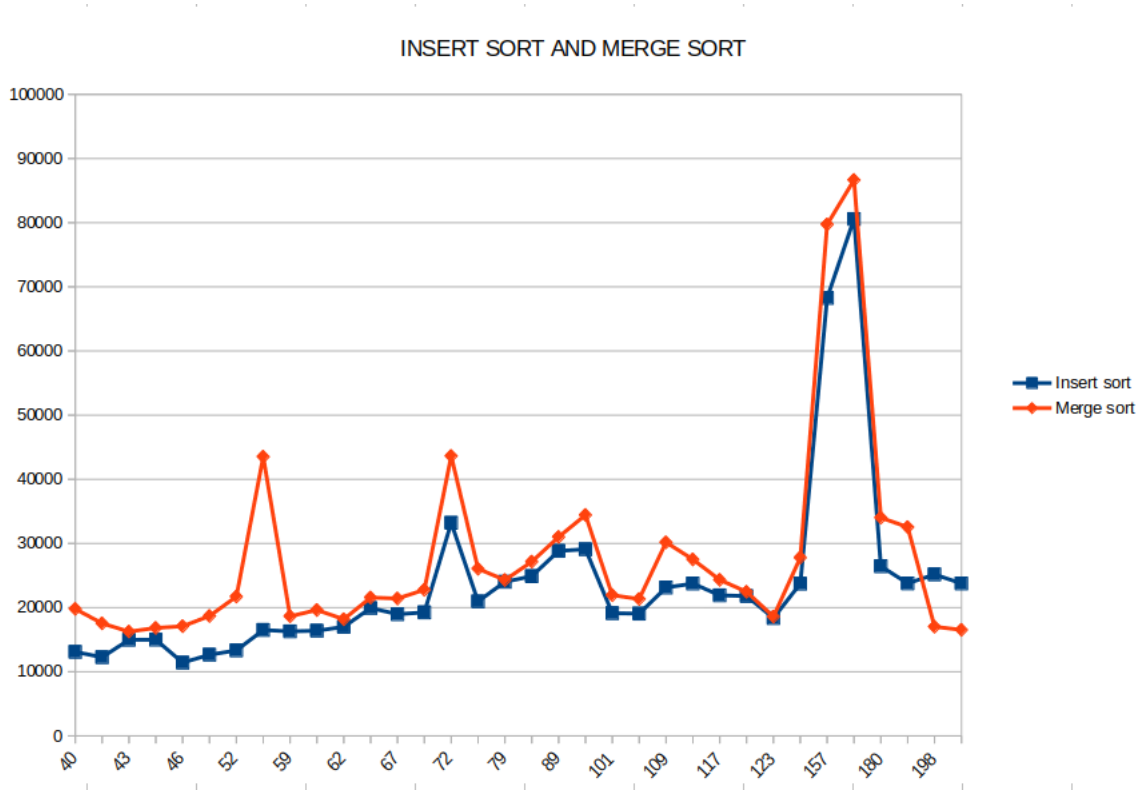

Figura 1:

3. Segundo ejercicio

```
#include<iostream>
#include <chrono>
#include <ctime>

using namespace std;
class Timer
{
public:
    void start()
    {
        m_StartTime = chrono::system_clock::now();
        m_bRunning = true;
    }

    void stop()
    {

        m_EndTime = chrono::system_clock::now();
        m_bRunning = false;
    }

    double elapsedMilliseconds()
    {
        chrono::time_point<std::chrono::system_clock> endTime;

        if(m_bRunning)
        {
            endTime = chrono::system_clock::now();
        }
        else
        {
            endTime = m_EndTime;
        }

        return chrono::duration_cast<chrono::nanoseconds>(endTime -
            m_StartTime).count();
    }

    double elapsedSeconds()
    {
        return elapsedMilliseconds() / 1000000.0;
    }

private:
    chrono::time_point<chrono::system_clock> m_StartTime;
    chrono::time_point<chrono::system_clock> m_EndTime;
    bool m_bRunning = false;
};


int* generadors(int tam){
    int* arreglo = new int[tam];
    srand(static_cast<unsigned int>(time(0)));
    int ran;
```

```
    for(int i=0;i<tam;i++){
        ran=0+rand()%10000;
        arreglo[i] = ran;
    }

    return arreglo;
}
void insert_sort(int* arreglo,int ini,int tam){
    int j;
    for(int i=ini;i<=tam;i++){
        int clave = arreglo[i];
        j=i-1;

        while(j>=ini && arreglo[j]>clave){
            arreglo[j+1]=arreglo[j];
            j=j-1;
        }
        arreglo[j+1]=clave;

    }
    cout<<endl;
}

void intercalar(int* arreglo,int p,int q,int r){
    int* aux = new int[r+1];
    for(int i=p;i<=q;i++){
        aux[i] = arreglo[i];
    }
    for(int j=q+1;j<=r;j++){
        aux[r+q+1-j] = arreglo[j];
    }
    cout<<endl;

    int i = p;
    int j = r;
    for(int k=p ;k<=r;k++){
        if(aux[i]<=aux[j]){
            arreglo[k] = aux[i];
            i=i+1;
        }else{
            arreglo[k] = aux[j];
            j=j-1;
        }
    }
}
void intercalar2(int* arreglo,int p,int q,int r){
    int aux[q+1],aux2[r+1];
    q=q+1;
    int tam1 = q;//3
    int tam2 = r-q; //3

    for (int i = 0; i < q; i++)
        aux[i] = arreglo[i];
    for (int j = 0; j <= r-q ; j++) //q=3 y r=6
        aux2[j] = arreglo[q+j];

    int i=0;
```

```
    int j=0;
    int k=0;
    while (i < tam1 && j <= tam2)
    {
        if (aux[i] <= aux2[j])
        {
            arreglo[k] = aux[i];
            i++;
        }
        else
        {
            arreglo[k] = aux2[j];
            j++;
        }
        k++;
    }


    while (i < tam1)
    {
        arreglo[k] = aux[i];
        i++;
        k++;
    }

    while (j <= tam2)
    {
        arreglo[k] = aux2[j];
        j++;
        k++;
    }

    cout<<endl;
}


int main(){

    int tam;
    int ini =0;

    cout<<"coloque tama o: \n";
    cin>>tam;
    int* arreglo = generadors(tam);

    //antes


    insert_sort(arreglo,ini,(tam/2)-1);
    insert_sort(arreglo,(tam/2),tam-1);


    //despues

    int* arreglo2 = new int[tam];

    for (int i=0;i<tam;i++){
```

```
        arreglo2[i] = arreglo[i];
    }

    cout<<endl;

    //INTERCALAR
    int r=tam-1;
    int q = (tam/2)-1;
    int p = 0;

    Timer timer;
    timer.start();

    intercalar(arreglo,p,q,r);
    timer.stop();

    cout<<endl;

    cout << "Milliseconds: " << timer.elapsedSeconds() <<endl;
    cout << "Nanosegundos: " << timer.elapsedMilliseconds() <<endl;

    //
    cout<<endl;

    //INTERCALAR SIN INVERTIR

    Timer timer1;
    timer1.start();
    intercalar2(arreglo2,p,q,r);
    timer1.stop();
    cout << endl;
    cout << "Milliseconds: " << timer1.elapsedSeconds() <<endl;
    cout << "Nanosegundos: " << timer1.elapsedMilliseconds() <<endl;
    cout<<endl;
    return 0;
}
```

Tamaño	Tiempo Real en nanosegundos	Tiempo Real en nanoegundos
Tamaño de arreglos	Intercalacion con valores invertidos	Intercalacion sin valores invertidos
10	1151	1009
20	8812	11020
50	11157	19194
100	11517	12471
500	22931	20620
1000	51568	46344
10000	102995	96278
100000	192327	177252
200000	1202560	1222020
500000	3185050	3023080
1000000	6394300	6085330

4. Diagrama de la tabla

Respuestas Positivas:
Sobre la implementacion de dicho algoritmo de intercalacion, en la parte donde invierte los elementos se emplea una asignacion de memoria de acuerdo al tamaño del arreglo, lo mismo se realiza para una intercalacion sin invertir el arreglo.
Tiene menos cantidades de asignacion de memoria y en asignaciones de variables;

Respuestas Negativas:
De acuedo a esta separacion de memoria de los dos algoritmo son similares, por lo tanto gastan el mismo tiempo de asignacion y eso puede generar mucho tiempo de demora para arreglos grandes.
En la parte de intercalacion sin inversion existe dos arreglos por separador,donde i reitera en el primero y el j reitera en el segundo, por lo tanto ambos se comparan, y colocan de manera creciente en el arreglo principal ordenandolo, esta operacion no tiene problemas en arreglos de tamaños pares ni en impares se puede notar que un arreglo sera mayor que el otro , asique el ultimo valor no tiene donde comparar y se coloca en los espacios restantes del arreglo principal que sera solamente uno, pero si un arreglo ya fue puesto de manera completa dentro del arreglo principal habiendo comparado con por ejemplo solo un elemento del otro arreglo entonces los valores que quedan por comparar se colocan de manera creciente en los espacios restantes del arreglo principal, dejando el costo por comparacion de lado, sin embargo en el algoritmo de intercalacion con valores invertidos realiza las comparaciones de cada elementos del arreglo para luego ordenar de manera creciente en el arreglo principal, por lo tanto esta produce una demora mas que el algoritmo sin elementos invertidos.

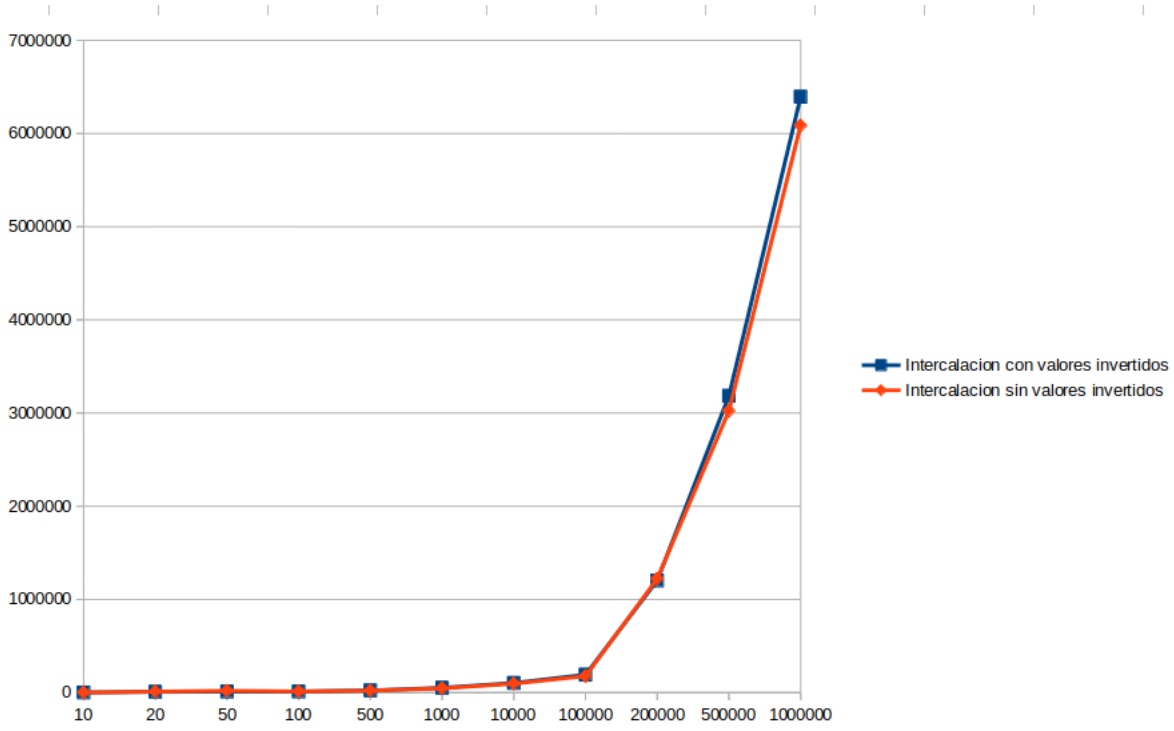
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
10
20
50
100
500
1000
10000
100000
200000
500000
1000000
Intercalacion con valores invertidos
Intercalacion sin valores invertidos

Figura 2: