5.1. Algoritmos de Ordenación Básicos.

UNIVERSIDAD
DE GRANADA

Julio José Reyes Hurtado
Antonio García Castillo

Información hardware

Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
Model name: Intel(R) Core(TM) i5-4258U CPU @ 2.40GHz
Mem RAM: 8 gb ram 1600 MHz DDR3
Stepping: 1 CPU MHz: 2400.000
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 3072K NUMA node0

Pruebas realizadas en una maquina virtual de ubuntu soportado por Mac OS.

Opciones de compilación: g++ -std=c++11 ejecutable.cpp -o ejecutable

Comandos gnuplot.

Se ha utilizado GNUplot para poder realizar las pruebas empíricas en cada uno de los algoritmos. En una primera aproximación se ha realizado un visionado de la gráfica, con plot 'archivo.dat', y posteriormente hemos utilizados las funciones generales para sacar las constantes y contrastarlo.

Se ha utilizado f(x) = a*x**2 + b*x + c para las funciones que tenian un comportamiento cuadrático seguido de fit f(x) 'tiempos.dat' via a,b,c para ajustarla a los tiempos extraídos de las pruebas, y por último con plot 'tiempos.dat', f(x) sacamos la imagen de los datos junto con la función de f(x).

Para las funciones de crecimiento lineal hemos usado una del tipo f(x) = a*x + b que es un crecimiento lineal.

Código algoritmos básicos de ordenación

```
#include <iostream>
#include <cstdlib>  // Para generación de números pseudoaleatorios
#include <chrono>// Recursos para medir tiempos
using namespace std;
using namespace std::chrono;

int insercion(int *a, int n)
{
  for (int i = 0 ; i<n ; i++){
      int x = a[i] ;
      int j = i-1;
      while(j>=0 && x<a[j]){
          a[j+1] = a[j];
          j = j-1;
      }
      a[j+1]=x;
  }
}

int seleccion(int *a, int n){
  for (int i = 0 ; i<n-1 ; i++){
    int pmin = i;
    for (int j=i+1 ; j<n ; j++){
      if ( a[pmin]>a[j]){
        pmin = j;
      }
    }
    swap(a[i],a[pmin]);
  }
}

int burbuja(int *a, int n)
{
    int c, d, t;
    for (c = 0; c < (n - 1); c++) {
        for (d = 0; d < n - c - 1; d++) {
            if (a[d] > a[d + 1]) {
                t = a[d];
                a[d] = a[d + 1];
                a[d + 1] = t;
            }
        }
    }
}

void sintaxis()
{
  cerr << "Sintaxis:" << endl;
  cerr << "  TAM: Tamaño del vector (>0)" << endl;
  cerr << "  VMAX: Valor máximo (>0)" << endl;
```

```
  cerr << "Se genera un vector de tamaño TAM con elementos aleatorios en [0,VMAX[" << endl;
  exit(EXIT_FAILURE);
}

int main(int argc, char * argv[])
{
  // Lectura de parámetros
  if (argc!=2)
    sintaxis();
  int tam=atoi(argv[1]);     // Tamaño del vector
  //int vmax=atoi(argv[2]);    // Valor máximo
  if (tam<=0 )
    sintaxis();

  // Generación del vector aleatorio
  // int *v=new int[tam];       // Reserva de memoria
  // srand(time(0));            // Inicialización del generador de números pseudoaleatorios
  // for (int i=0; i<tam; i++)  // Recorrer vector
  //   v[i] = rand() // % vmax ;    // Generar aleatorio [0,vmax[

  // generacion del vector ordenado
  int *ordenado = new int[tam];
  for ( int i = 0 ; i < tam ; i++){
      ordenado[i]=i;
  }

  // generacion del vector orden inverso
  int *orden_inverso = new int[tam];
  int cont = tam;
  for ( int i = 0 ; i<tam ; i++){
    orden_inverso[i] = cont;
    cont--;
  }

 high_resolution_clock::time_point start,//punto de inicio
                                   end; //punto de fin
 duration<double> tiempo_transcurrido;  //objeto para medir la duracion de end
                                        // y start
 start = high_resolution_clock::now(); //iniciamos el punto de inicio

  //int x = vmax+1;  // Buscamos un valor que no está en el vector
  seleccion(orden_inverso,tam); // de esta forma forzamos el peor caso

 end = high_resolution_clock::now(); //anotamos el punto de de fin
 //el tiempo transcurrido es
 tiempo_transcurrido  = duration_cast<duration<double> >(end - start);

  // Mostramos resultados
  cout << tam << "\t" <<tiempo_transcurrido.count() << endl;

  delete [] ordenado;     // Liberamos memoria dinámica
}
```

Ordenación por inserción

En esta práctica se analizan varios algoritmos tanto desde el punto de vista teórico como desde el punto de vista empírico. Primeramente realizaremos el análisis de la eficiencia de los algoritmos básicos de ordenación de vectores, por inserción, selección y burbuja.

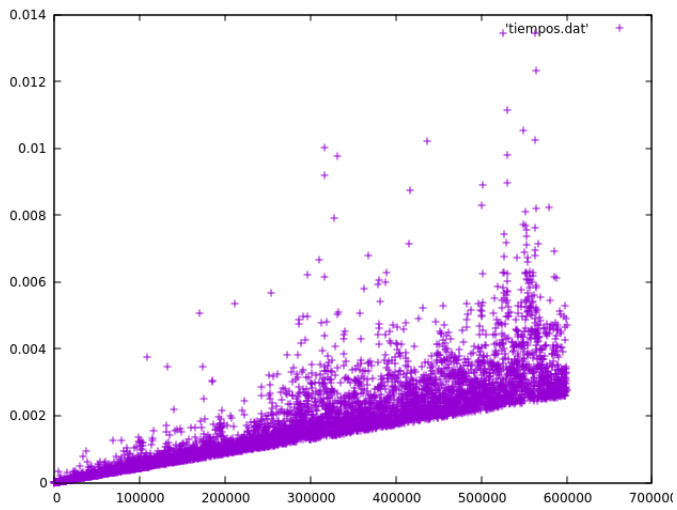
A continuación se estudiará el algoritmo de ordenación por inserción.

```
int insercion(int *a, int n)
{
  for (int i = 0 ; i<n ; i++){
      int x = a[i] ;
      int j = i-1;
      while(j>=0 && x<a[j]){
          a[j+1] = a[j];
          j = j-1;
      }
      a[j+1]=x;
  }
}
```

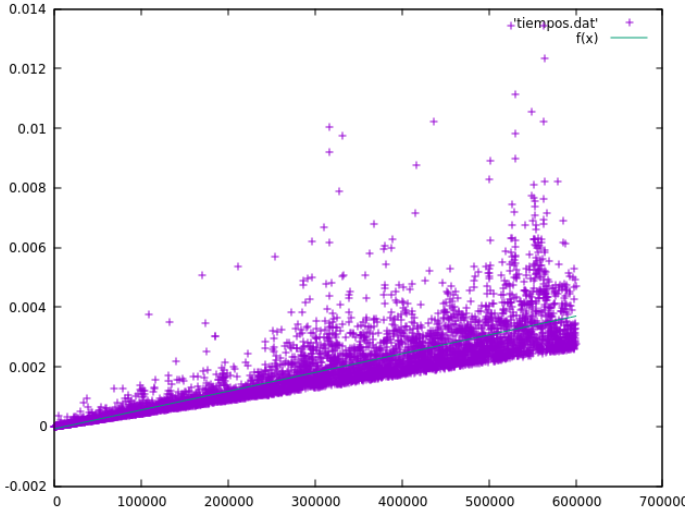
Como podemos observar este algoritmo consta de un bucle for que contiene un while anidado lo que nos puede llevar a pensar que es muy posible que su eficiencia en alguno de los casos tenga un crecimiento cuadrático. Este algoritmo en su mejor caso consigue llegar a una eficiencia lineal ya que si el vector está ordenado solamente entrara en el primer bucle pero nunca en el segundo.

El siguiente punto será confirmar de manera empírica por lo que utilizaremos gnuplot. A continuación se mostrarán varias imagenes para corroborar lo anteriormente mostrado.

Caso mejor: Tm(n) = 13*(n-1) +2

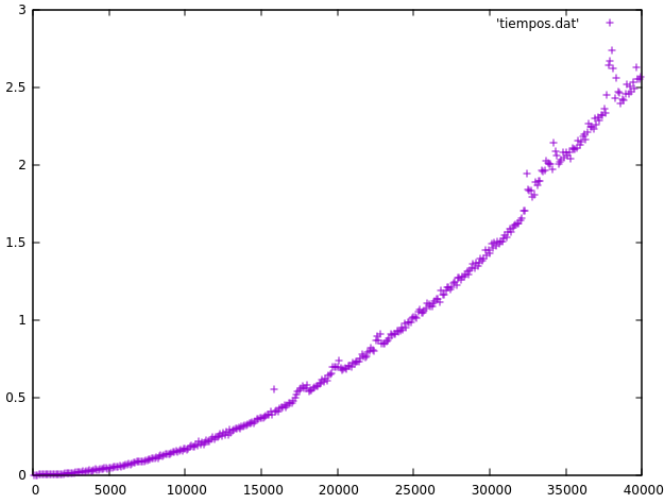


Podemos ver en el caso mejor excepto algunas de las mediciones la mayoria sigue una trayectoria lineal pero para corroborar con ayuda de gnuplot como se comporta una funcion con un ajuste lineal.

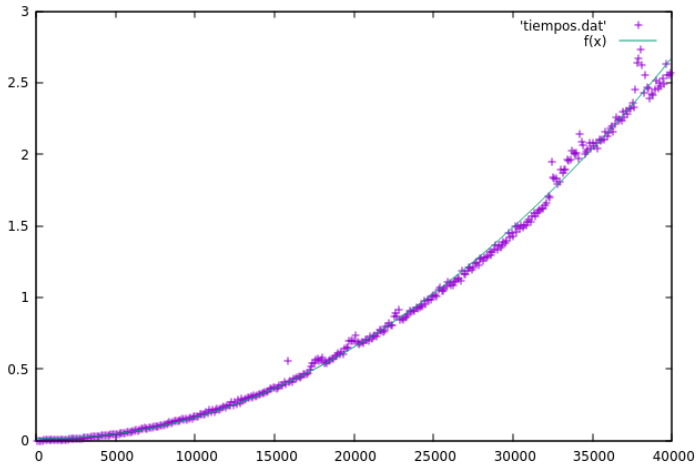


Podemos observar que tienen una tendencia lineal semejante por lo que podemos confirmar que la eficiencia teórica calculada se corresponde con la empírica en el caso mejor.

Caso peor: $5*n^2 – 5*n + 13*n – 11 = 5*n^2 + 8*n + 11$

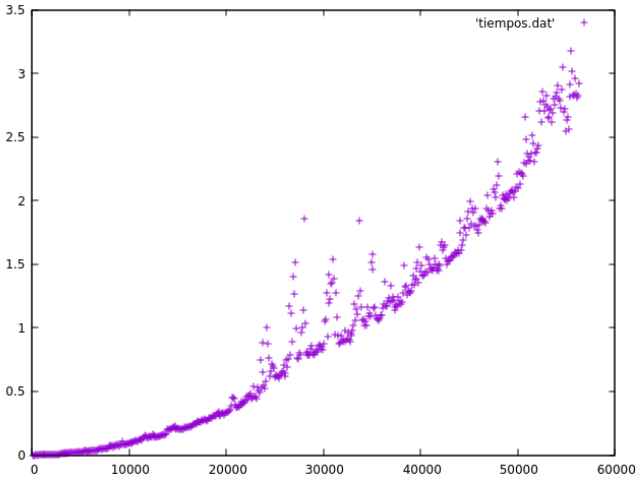


En este caso podemos apreciar que el crecimiento es cuadrático y para confirmarlo utilizaremos como en el ejemplo anterior una función auxiliar del tipof(x) = a*x^2 + b*x + c.

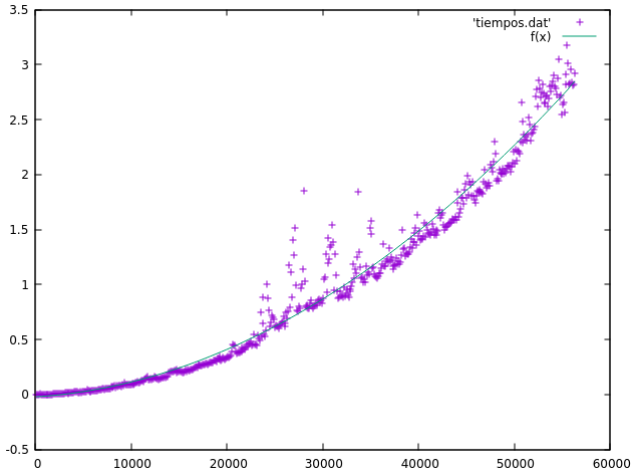


Vemos que se puede apreciar perfectamente como ambos crecimientos están superpuestos exceptuando algunas mediciones.

Caso promedio: $5/2*n^2 + 31/2*n - 16$



Al igual que en el estudio teórico tiene un comportamiento cuadrático y a continuación como en los casos anteriores vamos a realizar un ajuste con una funcion cuadratica generalizada.



Podemos ver que las mediciones están ligeramente por debajo de la función cuadrática general cosa que confirma la medición teórica obtenida en el análisis.

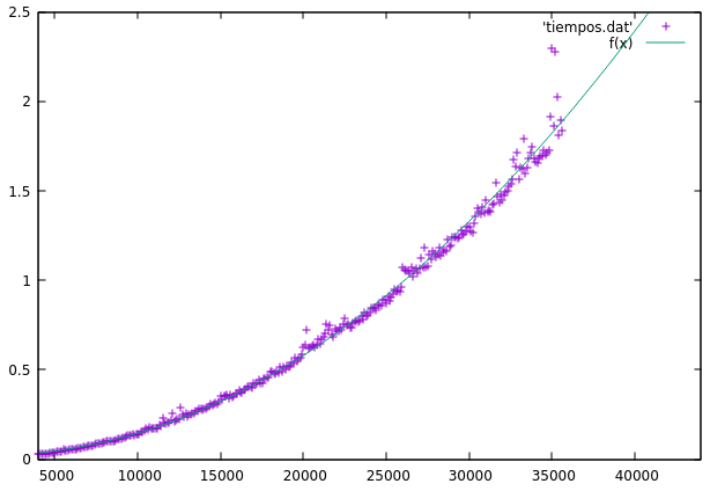
Ordenación por selección

El siguiente algoritmo que estudiaremos será el algoritmo de ordenación por selección:

```
int seleccion(int *a, int n){
  for (int i = 0 ; i<n-1 ; i++){
    int pmin = i;
    for (int j=i+1 ; j<n ; j++){
      if ( a[pmin]>a[j]){
        pmin = j;
      }
    }
    swap(a[i],a[pmin]);
  }
}
```

Caso peor, promedio: $3*n^2 + 8n - 8$, $5*n^2 + 39/4 + 8$

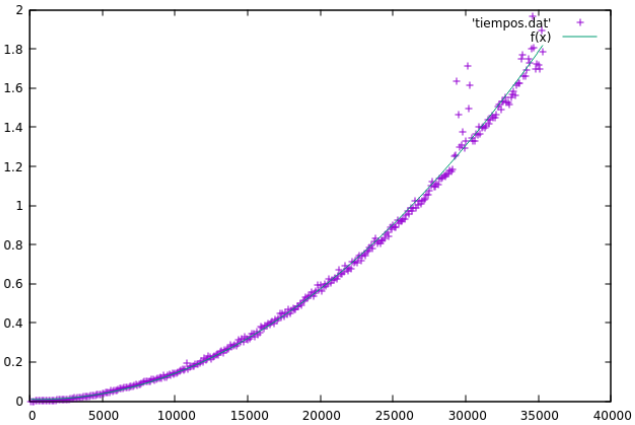
Este algoritmo al igual que el anteriormente estudiado contiene dos bucles anidados. En el caso peor la eficiencia del algoritmo por selección es $O(n^2)$. A continuación se procederá a verificar de manera empírica este comportamiento.



Como podemos observar tienen un crecimiento similar así como sería en el caso promedio para el cuál el vector debe tener la mitad de los dígitos ordenados y la otra mitad no, para que así se cumpla la condición la mitad de veces.

En el caso mejor: $(5*n^2)/2 + 17n/2 - 8$

Así como en los anteriores la eficiencia teoríca es cuadrática ya que en todos los casos se entre en ambos bucles. Las siguientes gráficas mostrarán el comportamiento en el caso mejor de este mismo algoritmo.



Como se puede observar tiene un comportamiento similar a los anteriores ya que crece como una función n^2. Como en el caso anterior se mostrará un ajuste con una función de este tipo. Que cuadra perfectamente.

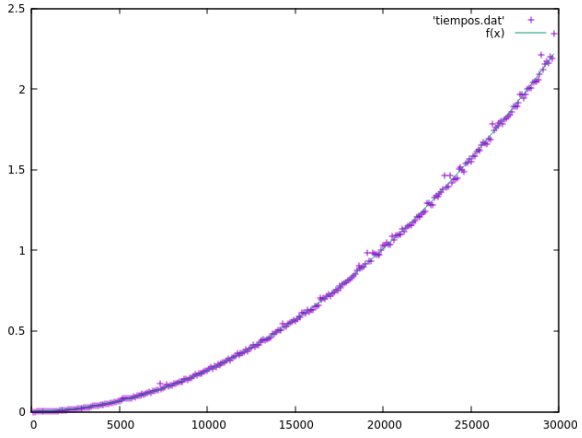
Ordenación por burbuja

El tercer algoritmo básico que se estudiará será el algoritmo burbuja cuyo comportamiento en los tres casos, peor, mejor y promedio es cuadrático ya que siempre recorre los dos bucles anidados que lo conforman de forma parecida al de selección

```c
int burbuja(int *a, int n)
{
    int c, d, t;

    for (c = 0; c < (n - 1); c++) {
        for (d = 0; d < n - c - 1; d++) {
            if (a[d] > a[d + 1]) {
                t = a[d];
                a[d] = a[d + 1];
                a[d + 1] = t;
            }
        }
    }
}
```

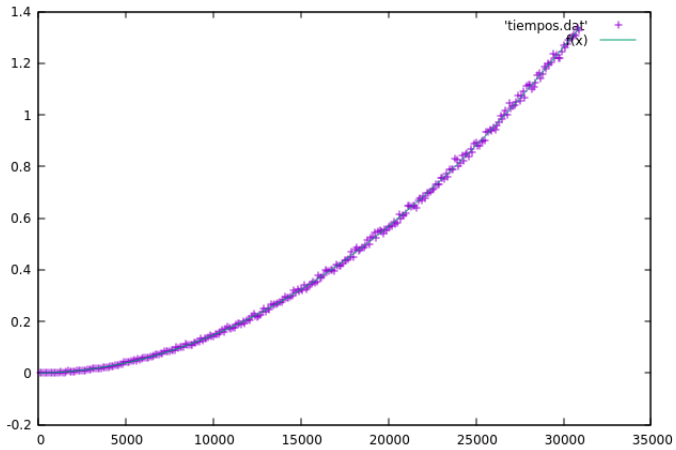
Caso peor, promedio: Tm(n) = n^2



Podemos observar su crecimiento de n^2 pero si lo comparamos con los anteriores vemos que para un mismo tamaño el tiempo es mayor. Como se puede observar cuadra perfectamente con la función generica con el mismo crecimiento que la prueba empírica. Así como sucedia antes el caso promedio con el vector parcialmente ordenado tiene el mismo crecimiento.

Para el caso mejor: $Tm(n) = n^2$

Ocurre algo similar ya que como se ha comentado el crecimiento sigue siendo cuadrático. Como se muestra en las siguientes gráficas.



Podemos observar que también se comporta de manera menos eficiente que los anteriores en los mejores casos pero la diferencia no es demasiado elevada. Por tanto, confirmamos que con esta última gráfica que el comportamiento es el mismo que el de una función cuadrática al igual que los anteriores.

Podemos pues decir que menos el mejor caso del algoritmo de inserción que tiene una eficiencia lineal en los demas casos sus gráficas siguen un comportamiento cuadrático.