5.2. Algoritmos de Ordenación con estructuras jerárquicas.

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).

En las eficiencias de los apo y caso mejor y promedio del abb que son del tipo nlog(n) utilizaremos log_b(x)=log(x)/log(2) para cambiarle la base al logaritmo y f(x)=a*(x*(log_b(x))) para tener la función y ya ajustarla con fit.

APO (Árbol parcialmente ordenado)

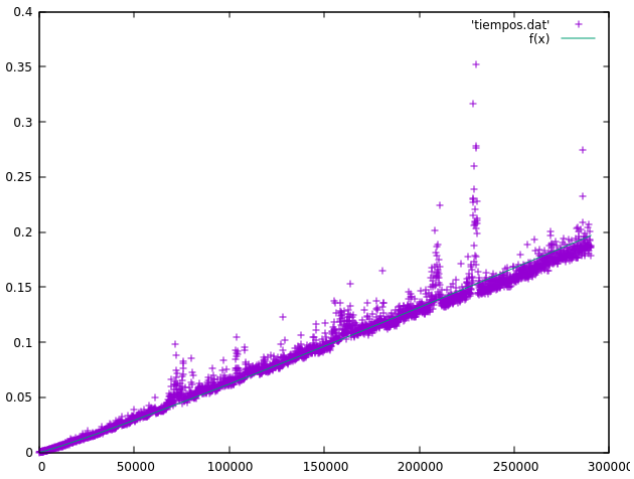
```
#include "apo.h"
#include <cstdlib>  // Para generación de números pseudoaleatorios
#include <chrono>// Recursos para medir tiempos
using namespace std;
using namespace std::chrono;


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[]){

 if (argc!=2){
    sintaxis();
}
int tam=atoi(argv[1]);


  // 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


APO<int>ap_int;
APO<int>ap_int2;
int a_insertar;
int tam_maximo = 1000;
srand(time(0));

        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 contador = tam;
        for ( int i = 0 ; i<tam ; i++ )
        {
                //a_insertar=rand()%tam_maximo; // este para aleatorio, osea promedio
                //a_insertar=i;                                 // este para ordenado
                a_insertar = contador;
                contador--;
                ap_int.insertar(a_insertar);
```

```
        }
        //cout<<"APO introducido: "<<ap_int <<endl;
        for (int i = 0 ; i<tam ; i++)
        {

                ap_int2.insertar(ap_int.minimo());
                ap_int.borrar_minimo();
        }

        //cout<<"APO ordenado: "<<ap_int2 <<endl;

        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;

}
```

A continuación se presentará el estudio de la eficiencia de las estructuras jerárquicas, concretamente las de los T.D.A apo (árbol parcialmente ordenado) y abb. Para comenzar realizaremos el estudio de los apo. El apo en los tres casos, mejor, peor y promedio presenta una eficiencia semejante nlog2(n).

Caso peor: Tm(n) = n*log(n)

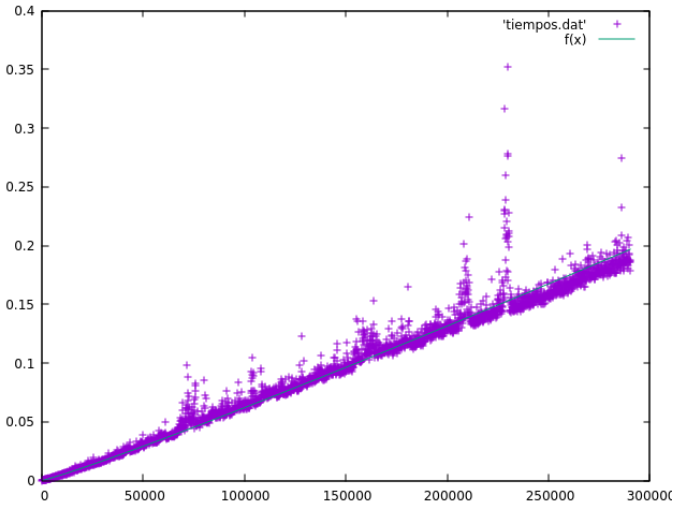
El primer caso a analizar es el caso peor utilizando un vector con los elementos ordenados de manera inversa, que mostraremos un ajuste entre los datos obtenidos y una función general para ver que los datos empíricos concuerdan con la forma de crecimiento de la función general.



Podemos confirmar que efectivamente nuestro análisis empírico ha tenido un comportamiento esperado ya que la función f(x) que como anteriormente se ha nombrado contiene la función general para nlog2(n).

Caso promedio: Tm(n) = n*log(n)

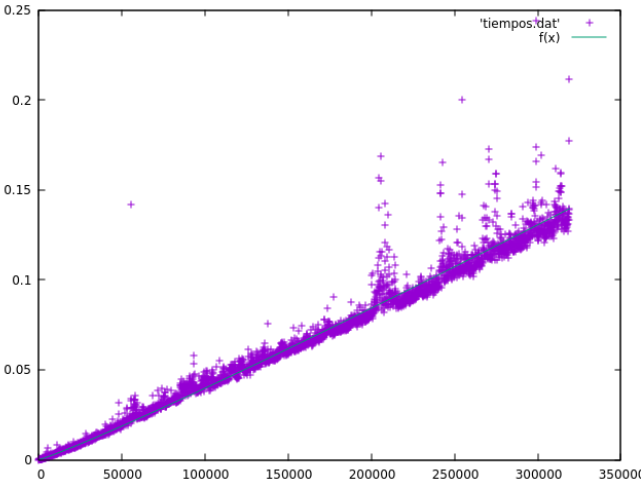
El siguiente caso a analizar es el caso promedio realizaremos la prueba con un vector con valores aleatorios. Utilizaremos el método realizado anteriormente comparando la gráfica obtenida con los valores introducidos con otra con una función general de la eficiencia teórica.



Podemos observar que los datos obtenidos siguen el patrón de la función general e incluso conforme va creciendo podemos observar que una alta cantidad de las mediciones se queda por debajo del crecimiento de la función general.

Caso mejor: Tm(n) = n*log(n)

Para el caso mejor hemos utilizado un vector ordenado. Ahora veremos que la eficienca para los tres casos es similar ya que en el ajuste con la función general cuadra como las anteriores.



Podemos observar que comparando con la gráfica anterior este algoritmo tiene un crecimiento mas lento en el tiempo ya que para mas muestras tiene un tiempo mas pequeño.

ABB (Árbol de busqueda binario)

```
#include "abb.h"
#include <string>
#include <sstream>
#include <vector>
#include <cstdlib>  // Para generación de números pseudoaleatorios
#include <chrono>// Recursos para medir tiempos
using namespace std::chrono;
using namespace std;

// tenemos que mirar la eficiencia en el metodo insertar

// en promedio n*log2N
// en el peor n^2
// en el apo tenemos que coger, borrarlo y ponerlas para medir el tiempo
// en gnuplot el factorial es gamma de n+1

void ListarAbb(ABB<int> &ab_bus)
{
        ABB<int>::nodo n;
        // cout << endl
        //          << "Elementos ordenados: ";

        for (n = ab_bus.begin(); n != ab_bus.end(); ++n)
        {
                *n;//    cout << *n << " ";
        }
}
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[]){
	if (argc!=2){
		sintaxis();
	}
	int tam=atoi(argv[1]);
	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
	vector<int>ordenado;
	vector<int>orden_inverso;
	vector<int>aleatorio;

	int tam_maximo = 1000;
	int a = tam;
	int al;
	srand(time(0));

	ABB<int> ab_bus;
	ABB<int> ab_bus2;
	ABB<int> ab_bus3;

	for (int i = 0 ; i<tam ; i++){
		//ordenado.push_back(i); // nos va a dar el peor
		 orden_inverso.push_back(a); // nos va a dar el mejor
		 a--;
		//aleatorio.push_back(rand()%tam_maximo); // promedio
	}


	for (int i = 0; i < ordenado.size(); i++){
		ab_bus2.Insertar(orden_inverso[i]);
```

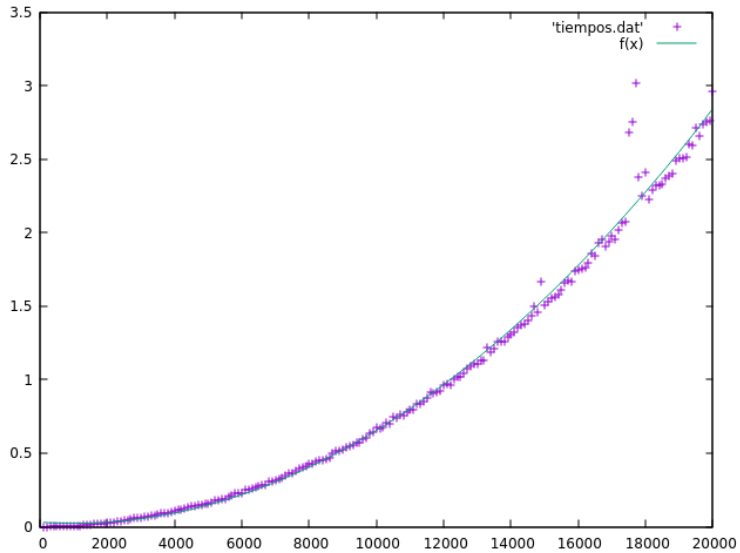
```
        }

        ListarAbb(ab_bus2);

        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;
}
```

Procederemos ahora a ver como se comporta la siguiente estructura jerárquica, el abb, para ver las diferencias con respecto a los distintos casos.

Cuando expongamos los tres casos diferentes podremos observar que cambia segun el orden de los datos que le vayamos introduciendo.

Caso peor: Tm(n) = n^2

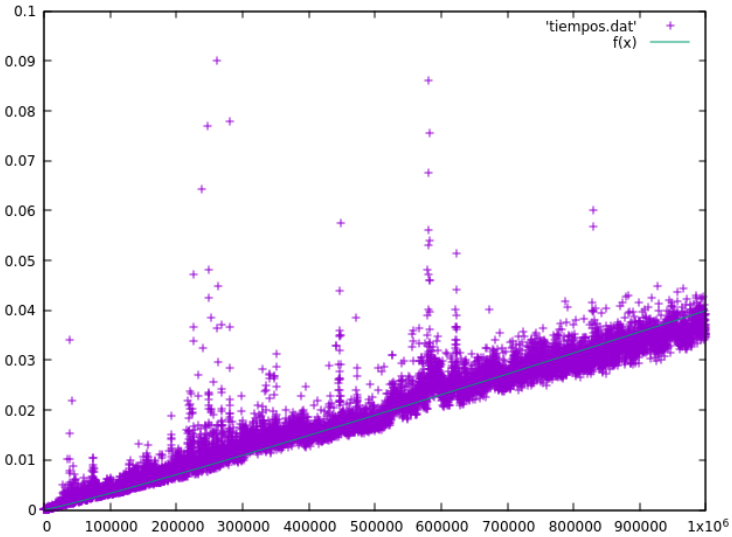
Que sigue un crecimiento cuadrático al introducirle la secuencia de datos ordenados.



El crecimiento ha sido muy alto y podemos ver que para crecia muy rapidamente comparado con las pruebas realizadas anteriormente con el apo. Podemos ver que también cuadra perfectamente con una función general de crecimiento n^2.

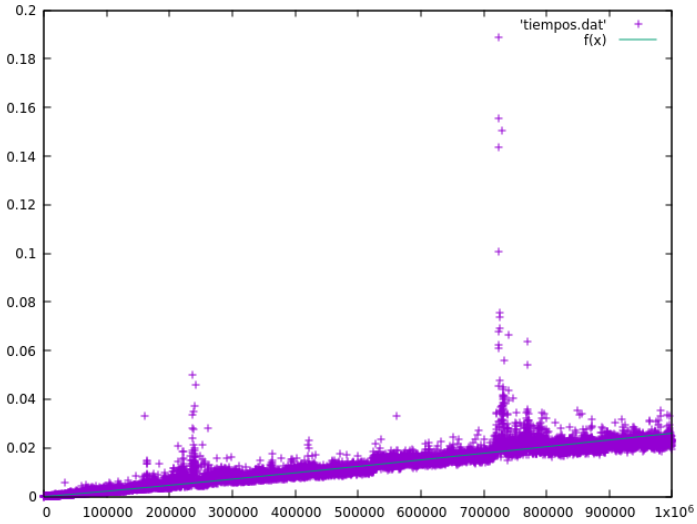
Caso promedio, caso mejor: Tm(n) = n*log(n)

Cuando le metemos los datos de manera aleatoria el abb procederá a ir ordenándolos.



Se puede ver perfectamente la diferencia entre el caso promedio y el caso superior ya que en el caso promedio conseguimos una eficiencia nlog(n) y al ajustarla con gnuplot con la función general la mayoria de los datos se encuentran alrededor de la linea trazada por la función.

El caso mejor que será expuesto a continuación, en el cual le pasamos un vector con los valores en orden inverso. Con esto conseguiremos un crecimiento parecido al caso promedio.



Se ha ajusta de manera perfecta en una función del tipo nlog(n) aunque ajustando los valores con el caso promedio se puede observar que tienen un crecimiento mas lento por lo que en el caso mejor hay mejoría con respecto a los anteriores.