5.3. Algoritmo de búsqueda por fuerza bruta

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.

Para la función factorial se ha de realizar f(x)=a*x*gamma(x)+b para que podamos realizar el fit de manera correcta.

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

template <class T>
ostream & operator<<(ostream &os, const vector<T> & d){
   for (int i=0;i<d.size();i++)
      os<<d[i]<<" ";
   os<<endl;
   return os;
}

void MuestraPermutaciones(const Permutacion & P){
  Permutacion::const_iterator it;
  int cnt=1;
  for (it=P.begin();it!=P.end();++it,++cnt)
      cout<<cnt<<"->"<<*it<<endl;
}

bool EsOrdenado(const int *v, const int n){
  bool es = true;
  for (int i =0;i<n-1;i++){
    if(v[i]>v[i+1]){
      es = false;
    }
  }
  return es;
}

void ModificaVector(int *v, int *v_final, const int *v_base, const Permutacion &P){
  const vector<unsigned int> s= (*(P.begin()));
```

```
  for(unsigned int i=0; i<s.size(); i++){
    v_final[i] = v_base[v[s[i]-1]];
  }
}
void ImprimeCadena(const string &c,const Permutacion &P){
  const vector<unsigned int> s= (*(P.begin()));

  for (unsigned int i=0;i<s.size();i++)
     cout<<c[s[i]-1];
  cout<<endl;
}

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]);
  int vmax=10;

          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

  if ( tam<=0){
    sintaxis();
}
```

```
  int *v= new int[tam];
  int *v_base = new int[tam];
  int *v_final = new int[tam];

  for ( int i = 0 ; i<tam ; i++){
    v[i]=i;
  }

  srand(time(0));

    for ( int i = 0 ; i<tam ; i++){
      v_base[i] = rand()%vmax;    // para aleatorio
    }

  Permutacion permu(tam, -1);
  do{
    ModificaVector(v, v_final, v_base, Otra);

  }while(!EsOrdenado(v_final,tam) && permu.GeneraSiguiente()  );

    delete[] v;
    delete[] v_base;
    delete[] v_final;

  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;
}
```

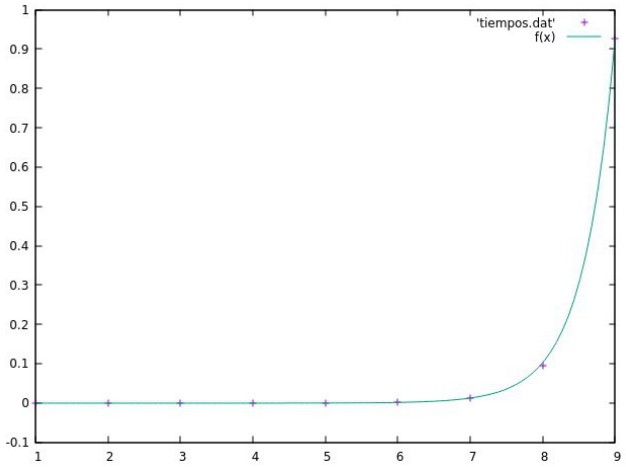
El algoritmo analizado es el de ordenación por fuerza bruta que consiste en realizar permutaciónes en un vector de numeros hasta que estos se queden ordenados de manera ascendente. Esto nos sugiere que la eficiencia en el caso peor es n*n! Ya que ordena pero tambien realiza la comprobacion de que este ordenado.

Caso mejor:

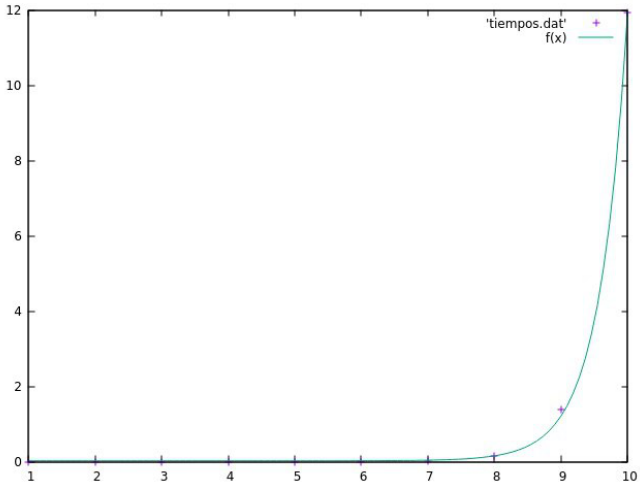
El caso mejor unciamente requiere de la comprobación de saber si esta ordenado ya que la primera permutación ya se encuentra ordenada por lo que la eficiencia sería lineal.

Caso promedio:

La siguiente gráfica corresponde al caso promedio realizado con números de 9 dígitos donde podremos comprobar una alta diferencia cuando es una cantidad de dígitos menor.



Podemos ver para una cantidad de 8 dígitos el tiempo llega casi a 0.1 segundos pero para 9 aumenta hasta casi 1 segundo completo. Para seguir analizando este comportamiento se ha realizado otra prueba con hasta diez dígitos.



Se puede apreciar una gran diferencia cuando pasamos a 10 dígitos ya que pasa de poco menos de un segundo a doce Se puede comprobar que existe un gran salto conforme se añaden dígitos. Esto verifica el crecimiento factorial de la eficiencia teórica del algoritmo.

Caso peor:

Tiene un comportamiento similar al promedio ya que crece de la misma manera, n*n!, por lo que veremos que tiene un comportamiento similar solo que con tiempos ligeramente menores ya que para el caso promedio se ha utilizado un vector aleatorio y para este caso, el peor, hemos usado el vector completamente a la inversa y como consecuencia obtenemos los siguientes resultados.