Practica 2: Divide y vencerás

EJERCICIO 1:

En el primero ejercicio debemos calcular el maximo y el minimo valor de un vector usando la técnica de divide y vencerás, para ello he implementado el siguiente algoritmo:

```
pair <int,int> Max_Min(int *v, int n){
  pair<int,int> resultado, p1, p2;
  if(n==1){
    resultado.first = v[0];
    resultado.second = v[0];
  }else if(n==2){
    if(v[0]>v[1]){
      resultado.first=v[0];
      resultado.second=v[1];
    }else{
      resultado.first=v[1];
      resultado.second=v[0];
    }
  }else{
    p1=Max_Min(v,n/2);
    p2=Max_Min(v+n/2,n-n/2);

    if(p1.first>p2.first)
      resultado.first = p1.first;
    else
      resultado.first = p2.first;

    if(p1.second<p2.second)
      resultado.second = p1.second;
    else
      resultado.second = p2.second;
    }

  return resultado;
}
```

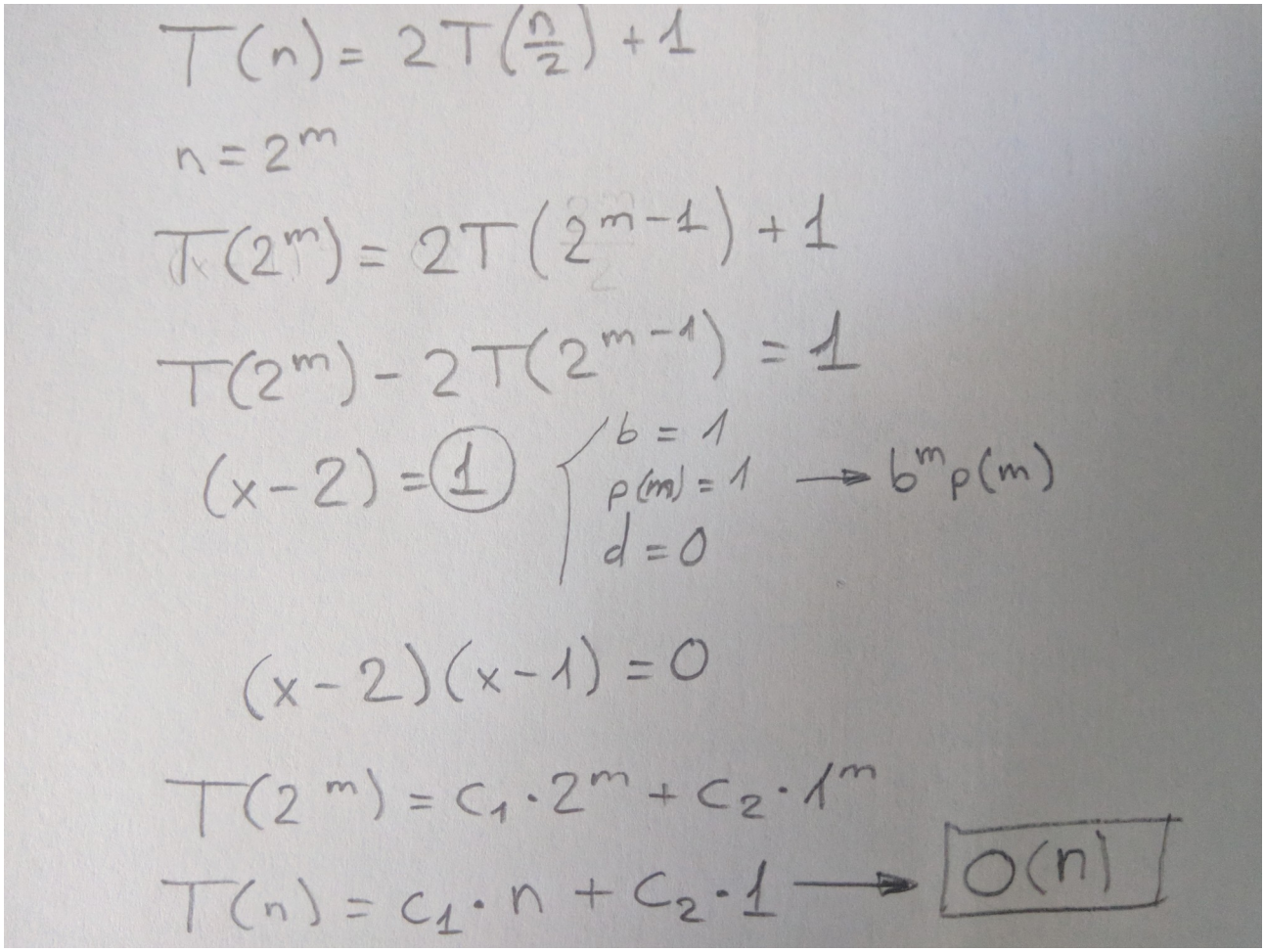
Lo que realizamos es ir dividiendo el vector en mitades y a su vez esas dos mitades en otras dos y así sucesivamente hasta encontrar cual es el máximo y cual es el minimo, para ello si el tamaño del vector es 1, directamente el mismo es el mayor y el menor, si el tamaño es dos uno es el mayor y otro es el menor y en cualquier otro caso se vuelve a llamar recursivamente a Max_Min para dividir de nuevo el vector una vez que ya no se puede dividir más fusionamos los resultados con la parte final del algoritmo.

En cuanto a la eficiencia teoria deducimos que las dos promeras codiciones tienen un coste de 1, solo nos centramos en el final, en el ultimo else. En este caso, en la fusión el tiempo o coste es de 1 ya que se realizan operaciones simples de comparación y asignación. El problema está en la llamada

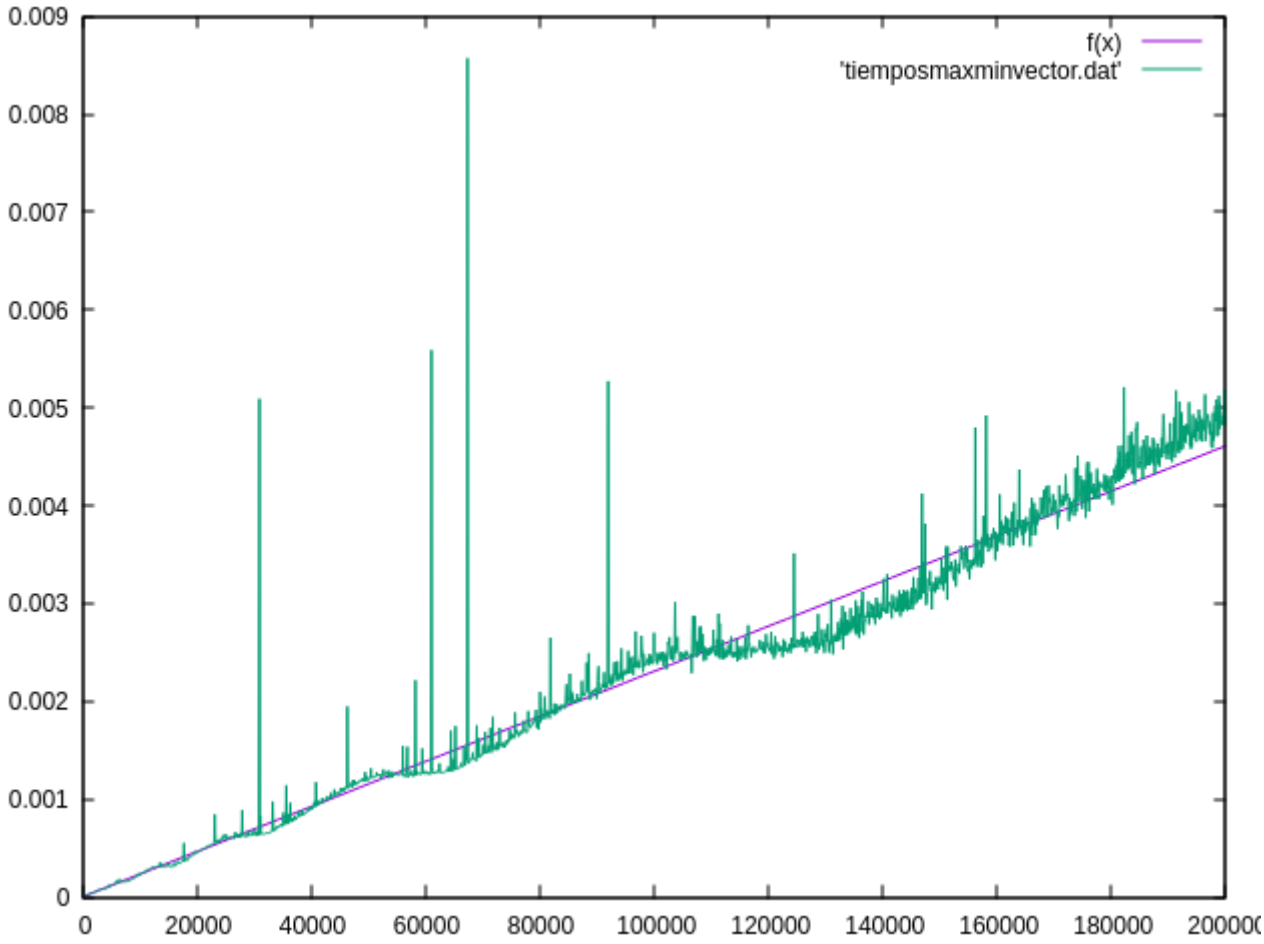
recursiva a Max_Min, en este caso al realizar dos llamadas a este método, es decir, lo llamamos con n/2 elementos y los otros n/2 se los damos a la otra llamada, luego deducimos de esto que el orden final de este método esta alrededor del **O(N)**. Concretamente podría ser de:

T(n) = 2T(n/2)+1

Adjunto foto donde realizo el procedimiento del calculo de la eficiencia teórica:



En cuanto a la eficiencia empírica, tiene sentido con la eficiencia teoria calculada, puesto que al realizar el ajuste a la curva de O(N) me lo realiza correctamente:



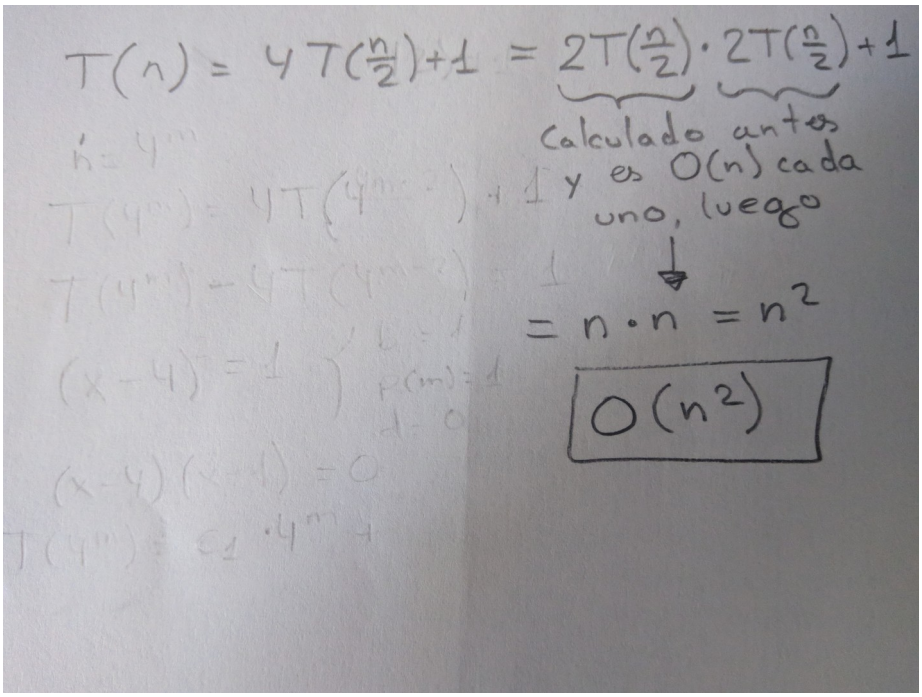
EJERCICIO 2:

En el segundo ejercicio lo que debemos realizar es calcular el maximo y minimo valor que podemos encontrar en una matriz, aplicando de nuevo un algoritmo usando la técnica divide y vencerás, para ello he implementado el siguiente algoritmo:

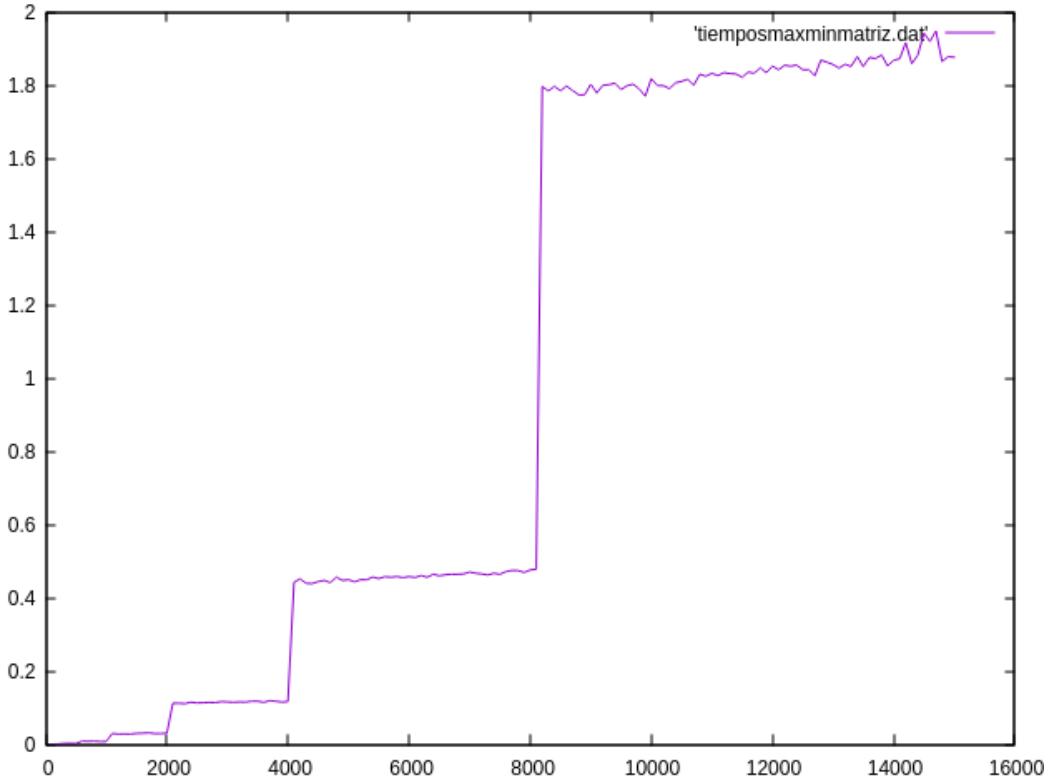
```
pair<int ,int> Max_Min(int ** M,int inicioi,int inicioj,int n){
  pair<int,int> resultado, p1, p2, p3, p4;
  if(n==1){
    resultado.first = M[inicioi][inicioj];
    resultado.second = M[inicioi][inicioj];

    return resultado;
  }else{
    p1=Max_Min(M,inicioi,inicioj,n/2);
    p2=Max_Min(M,inicioi,inicioj+n/2,n/2);
    p3=Max_Min(M,inicioi+n/2,inicioj,n/2);
    p4=Max_Min(M,inicioi+n/2,inicioj+n/2,n/2);

    resultado.first = p1.first;
    resultado.second = p1.second;

    if (p2.first > resultado.first)
        resultado.first = p2.first;

    if (p2.second < resultado.second)
        resultado.second = p2.second;


    if (p3.first > resultado.first)
        resultado.first = p3.first;

    if (p3.second < resultado.second)
        resultado.second = p3.second;


    if (p4.first > resultado.first)
        resultado.first = p4.first;

    if (p4.second < resultado.second)
        resultado.second = p4.second;

    return resultado;
  }
}
```

El algoritmo lo que realiza es algo similar al algoritmo del vector, pero en este caso dividimos la matriz en cuatro trozos, dividimos fila por la mitad y columna por la mitad, dando lugar a una cuadricula de cuatro secciones. Una vez tenemos esta división, estas serian las diferentes llamadas recursivas del algoritmo, lo que nos quedaría luego es fusionar todos los máximos y minimos que hemos encontrado, es decir, tendremos cuatro valores minimos y cuatro valores máximos, luego tendriamos que elegir cual de esos cuatro valores de cada uno es el máximo y cual el minimo.

En cuanto a la eficiencia teórica, adjunto una foto donde la realizo:



Como observamos la eficiencia es de O(n²), el problema que he tenido es que cuando he realizado la eficiencia empírica no se corresponde con la eficiencia calculada teóricamente, y no se si puede ser debido algún problema con mi ordenador, pero dejo la gráfica de los tiempos sacados de ejecutar el algoritmo de busqueda del máximo y mínimo de una matriz:



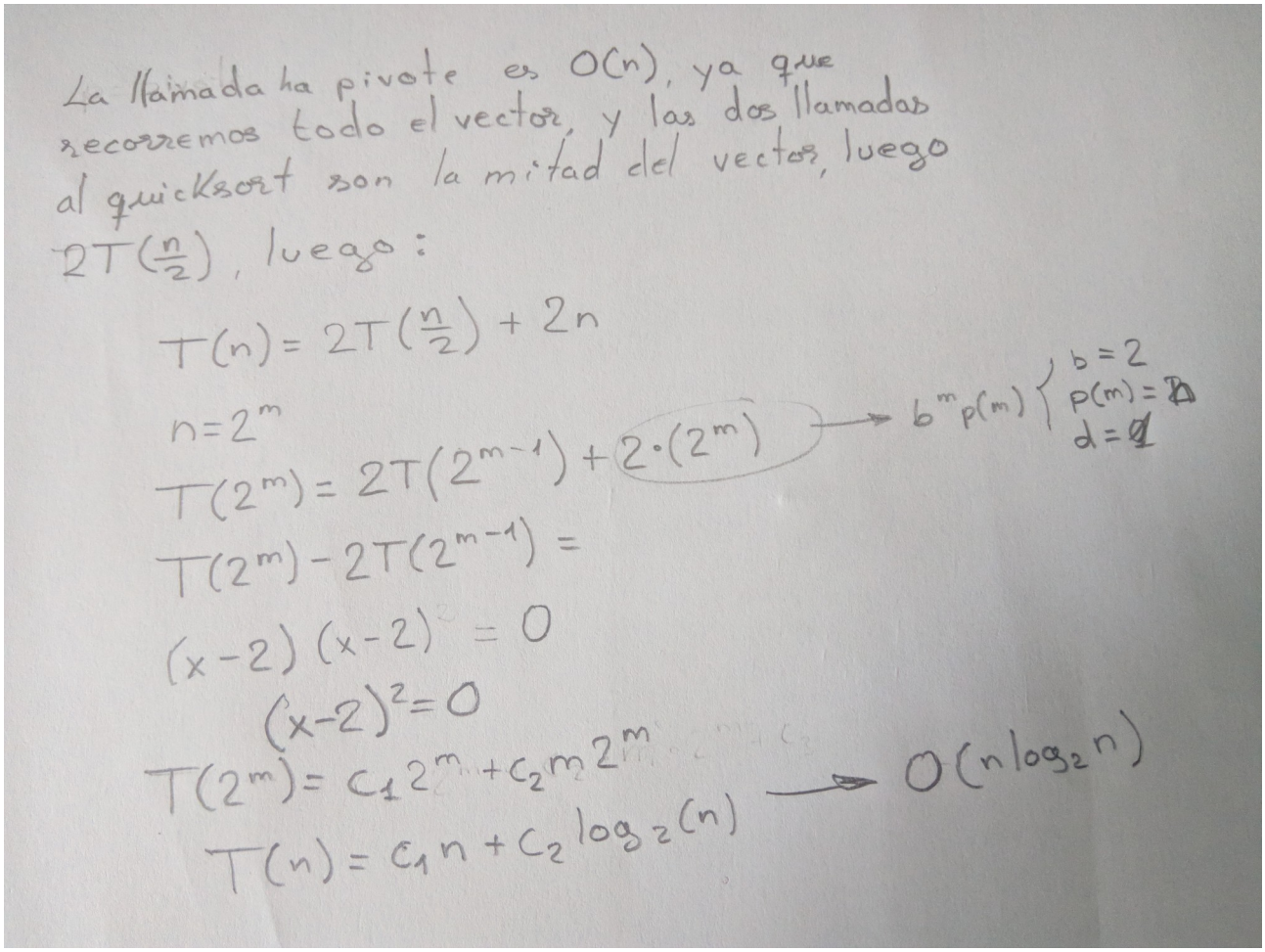
EJERCICIO 3:

En este ejercicio lo que nos pedían es aplicar un algoritmo que teniendo dos vectores con los mismo valores, pero uno de ellos con los valores permutados, que aplicando el algoritmo de ordenación por quicksort usando como pivote un valor del otro vector que al final ambos vectores queden acorde, es decir, que el valor del primer vector tiene que estar en la misma posición luego en el segundo vector. Esto se hace para asociar una serie de zapatos a una serie de tamaños de pies, ambos introducidos en los dos vectores que he dicho anteriormente. Para ello he usado el siguiente algoritmo:

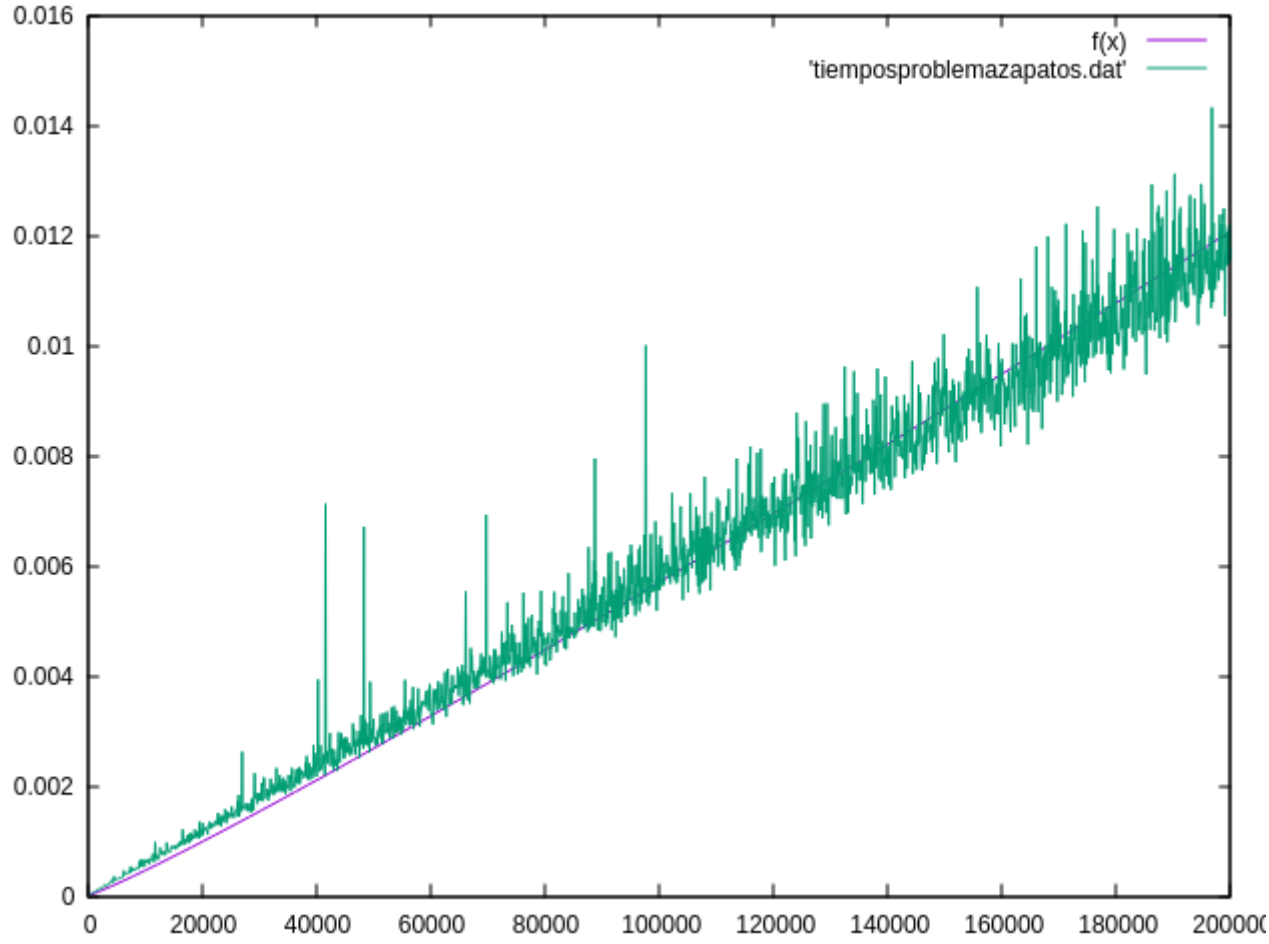
```
pair<int,int> pivote(int *v, int pivote, int inicio, int final){
  pair<int,int>p;
  p.first=inicio;
  p.second=final;
  int i=inicio;

  while(i<=p.second){
    if(v[i]<pivote){
      swap(v,p.first++,i++);
    }else if(v[i]>pivote){
      swap(v,i,p.second--);
    }else{
      i++;
    }
  }
  return p;
}

// Función recursiva para hacer el ordenamiento
void quicksort(int *zapatos, int* pies, int inicio, int final){
  pair<int,int> p;

  if (inicio < final) {
      p=pivote(pies,zapatos[inicio],inicio,final);
      p=pivote(zapatos,pies[p.second],inicio,final);

      // Ordeno la lista de los menores
      quicksort(zapatos, pies, inicio, p.first-1);

      // Ordeno la lista de los mayores
      quicksort(zapatos, pies, p.second+1, final);
  }
}
```

En este algoritmo lo que hacemos es una variación del quicksort normal, para que nos devuelva los indices donde se encuentra los valores menores, iguales y mayores al pivote, es decir divide al vector en tres trozos, dejando finalmente el vector como hemos dicho. La clave de este algoritmo está en que nunca usa como pivote un valor propio, si no que usa como pivote un valor del segundo vector y el segundo vector del primero.

En cuanto a la eficiencia teórica, la dejo en forma de ejercicio:



En cuanto a la eficiencia empírica hemos verificado que es correcto el calculo de la eficiencia teórica que hemos realizado, para ello he ejecutado el algoritmo con diferentes tamaños y he realizado su gráfica, ajustandola al orden de eficiencia teórico calculado:



EJERCICIO 4:

En este ejercicio lo que nos pedian era calcular la moda de los valores de un vector usando la técnica divide y vencerás, para ello el algoritmo que he realizado es el siguiente:

```
pair<int, int> Moda(int v[], int inicio, int final) {
  pair<int, int> moda;
  int *homogeneo=new int[final];
  int *heterogeneo=new int[final];
  int tamanoHo = 0, tamanoHe = 0;
  int j = 0;

  moda.first = 0;
  moda.second = 0;

  if (!ordenado){
    quicksort(v, inicio, final);
    ordenado = true;
  }

  do{
    homogeneo[tamanoHo] = v[j];
    ++j;
    ++tamanoHo;
  }while (v[j] == v[j-1]);

  while (tamanoHe < (final - tamanoHo)){
    heterogeneo[tamanoHe] = v[j];
    ++j;
    ++tamanoHe;
  }

  if (tamanoHo > tamanoHe && tamanoHo > 1){
    moda.first = homogeneo[0];
    moda.second = tamanoHo;
  }else{
    pair<int, int> modaheterogeneo =  Moda(heterogeneo, tamanoHo, tamanoHe);
    if (modaheterogeneo.second > moda.second){
      moda.first = modaheterogeneo.first;
      moda.second = modaheterogeneo.second;
    }
  }
  return moda;
}
```

Como podemos observar lo que realizo es, primero ordenar el vector inicial, una vez ordenado, lo que realizo es contar y guardar los valores iguales al pivote, y por otro lado guardo y cuento los valores menores y mayores al pivote. Si el tamaño del vector de homogeneos (valores iguales al pivote) es mayor al de heterogéneos (valores menores y mayores al pivote) directamente devuelvo la moda, que es el valor almacenado en homogéneos, en cambio, si esto no es así, vuelvo a llamar recursivamente a moda con el vector de heterogéneos.

En cuanto a la eficiencia teórica, podemos ver en el código que lo más relevante de todo es la llamada a QuickSort para ordenar el vector, la llamada recurrente a Moda y los dos bucle centrales. Como el vector esta ordenado y la llamada a QuickSort tiene una eficiencia de O(nlog(n)), la eficiencia teórica total del algoritmo es de esta, O(nlog(n)).

Finalmente, para calcula la eficiencia empírica, he lanzado el algoritmo con diferentes tamaños de vector y he calculado su tiempo de ejecución, para que al final lo pueda representar con gnuplot en una gráfica, ajustandolo a la eficiencia teórica calculada anteriormente: