UNIVERSIDAD DE GRANADA

Divide y Vencerás

ALGORÍTMICA

Lukas Häring García 2ºD

Tabla de contenidos

Máximo y mínimo en un vector **2**
0.1 Código 2
0.2 Eficiencia (n = Tamaño de la muestra) 3
0.3 Gráfica 3

Mínimo y máximo en una matriz **4**
1.1 Código 4
1.2 Eficiencia (n^2 = Tamaño de la matriz) 5
1.3 Gráfica 5

Zapatos con sus pies **6**
2.1 Código 6
2.2 Eficiencia 8
2.3 Gráfica 8

Moda de un conjunto de enteros **9**
3.1 Código 9
3.2 Eficiencia 10
3.3 Gráfica 10

Especificaciones **11**

Máximo y mínimo en un vector

0.1 Código

```
#include<utility>
#include<limits>
#include<cmath>
// Suponemos que n > 0
// PAR RESULTANTE: (MAX, MIN)
std::pair<int, int> Max_Min(int* v, int n){
  std::pair<int, int> result(*v, *v);
  // Si solo hay un elemento, entonces ese es el minimo y maximo
  if(n == 1){
    return result;
  }
  // Si hay dos elementos, entonces el min y el max son ambos.
  if(n == 2){
    if(*(v + 0) < *(v + 1)){
      result.first = *(v + 1);
    }else{
      result.second = *(v + 1);
    }
    return result;
  }
  // Calculamos las particiones (Segun sea par o impar).
  int bt = floor(n / 2.0), tp = ceil(n / 2.0);
  // Particion izquierda.
  std::pair<int, int> left = Max_Min(v, bt);
  // Particion derecha.
  std::pair<int, int> right = Max_Min(v + bt, tp);

  // max{max0, max1}, min{min0, min1}
  result.first = std::max(left.first, right.first);
  result.second = std::min(left.second, right.second);
  // Devolvemos el resultado.
  return result;
}
```

0.2 Eficiencia (n = Tamaño de la muestra)

$$T(n) = \begin{cases} 1 & n = 1,2 \\ T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + 1 & n \geq 3 \end{cases}$$

Realizando la siguiente sustitución, $n = 2^k$.

$$T(2^k) = T(\lfloor 2^{k-1} \rfloor) + T(\lceil 2^{k-1} \rceil) + 1 = 2 \cdot T(2^{k-1}) + 1$$

1. La parte **homogénea** es: $T(2^k) - 2 \cdot T(2^{k-1})$.
2. La parte **no homogénea** es: $1 = 1 \cdot 1^k$.

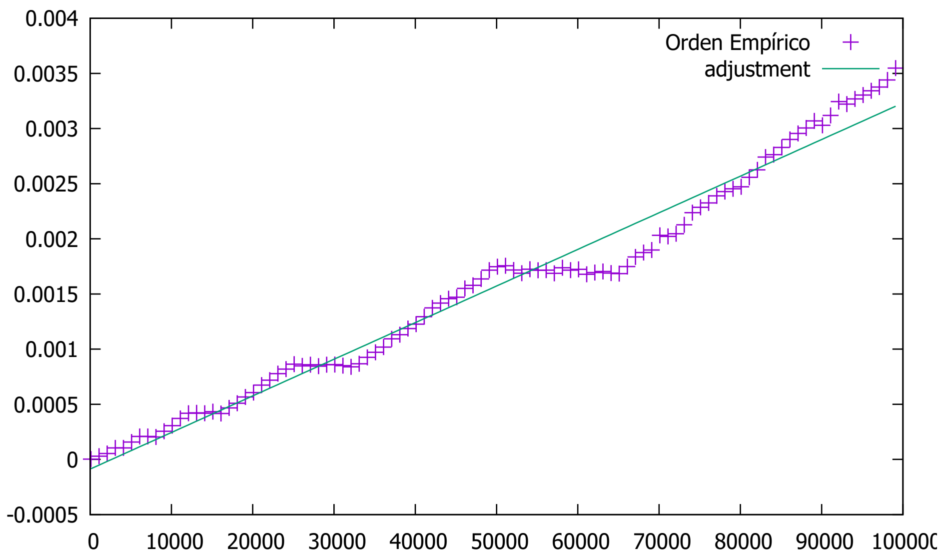
Por lo que resolviendo las ecuaciones, obtenemos que:

1. De la parte **homogénea** es: $k - 2 = 0 \Rightarrow k = 2$.
2. De la parte **no homogénea**: $k = 1$.

Obtenemos la ecuación no recurrente.

$T(2^k) = 2^k \cdot c_1 + 1^k \cdot c_2 = n \cdot c_1 + c_2 = T(n) \Rightarrow O(T(n)) = O(n \cdot c_1 + c_2) \in O(n)$

0.3 Gráfica



Mínimo y máximo en una matriz

1.1 Código

```cpp
#include<utility>
#include<limits>
#include<cmath>
// PAR RESULTANTE: (MAX, MIN)
std::pair<int,int> Max_Min(int** v,int ii,int jj,int n){
  std::pair<int, int> result;
  if(n == 2){ // Matriz 2 x 2
    int max = std::numeric_limits<int>::min();
    int min = std::numeric_limits<int>::max();
    for(int j = jj; j < jj + n; ++j){
      for(int i = ii; i < ii + n; ++i){
        min = std::min(v[j][i], min);
        max = std::max(v[j][i], max);
      }
    }
    result.first = max;
    result.second = min;
    return result;
  }
  // Calculamos la la siguiente particion.
  int k = n % 2, tp = ceil(n / 2.0);
  int rr = tp - k;
  std::pair<int,int> tl = Max_Min(v, ii     , jj     , tp);
  std::pair<int,int> tr = Max_Min(v, ii + rr, jj     , tp);
  std::pair<int,int> bl = Max_Min(v, ii     , jj + rr, tp);
  std::pair<int,int> br = Max_Min(v, ii + rr, jj + rr, tp);
  //max,min{(max0, min0),(max1, min1),(max2, min2),(max3, min3)}
  result.first = std::max(std::max(tl.first, tr.first),
  std::max(bl.first, br.first));
  result.second = std::min(std::min(tl.second, tr.second),
  std::min(bl.second, br.second));
  // Devolvemos el resultado.
  return result;
}
```

1.2 Eficiencia (n^2 = Tamaño de la matriz)

$$T(n^2) = \begin{cases} 1 & n = 2 \\ 4 \cdot T\left(\dfrac{n^2}{4}\right) + 1 & n \geq 3 \end{cases}$$

Realizando la siguiente sustitución, $n^2 = 2^k$.

$$T(2^k) = 4 \cdot T(2^{k-2}) + 1 = 0 \cdot T(2^{k-1}) + 4 \cdot T(2^{k-2}) + 1$$

1. La parte **homogénea** es: $T(2^k) - 4 \cdot T(2^{k-2})$.
2. La parte **no homogénea** es: $1 = 1 \cdot 1^k$.

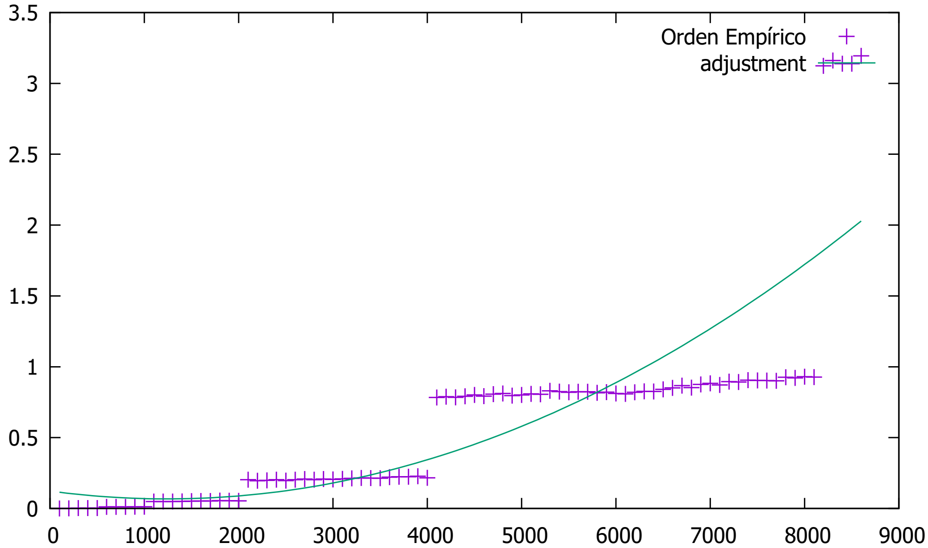
Por lo que resolviendo las ecuaciones, obtenemos que:

1. De la parte **homogénea** es: $k^2 - 4 = 0 \Rightarrow k = \sqrt{4} = 2$.
2. De la parte **no homogénea**: $k = 1$.

Obtenemos la ecuación no recurrente (Conociendo que $2^k = n^2$).

$$T(2^k) = 2^k \cdot c_1 + 1^k \cdot c_2 = T(n^2) \Rightarrow O(T(n^2)) = O(n^2 \cdot c_1 + c_2) \in O(n^2)$$

1.3 Gráfica



Zapatos con sus pies

2.1 Código

```
#include<iostream>
#include<utility>
#include <ctime>

// Realiza el swap de dos valores en un vector.
void swap(int* v, int i, int j){
  int h = *(v + i);
  *(v + i) = *(v + j);
  *(v + j) = h;
}

// Desplaza los elementos desde la posicion i de un vector v
// e introduce en la posicion i, el valor k
void shift(int* v, int i, int n, int k){
  for(int j = n - 1; j > i; --j){
    *(v + j) = *(v + j - 1);
  }
  *(v + i) = k;
}
```

```
void Zapatos_Pies(int* shoes, int* toes, int n){
  if(n <= 1){
    return;
  }
  // Pivote (i) para los pies del ultimo zapato
  int i = 0;
  // Buscamos el pivote en para los pies.
  for(i = 0; *(toes + i) != *(shoes + n - 1); ++i);
  // Hacemos swap del pie pivote con el ultimo
  swap(toes, i, n - 1);

  int m, w;
  /* MOVEMOS LOS PIES */
  m = -1;
  w = 0;
  while(w < n){
    // Comparamos el elemento con el pivote de los pies
    // Si este es menor, intercambiamos el muro con el actual.
    if(*(toes + w) < *(shoes + n - 1)){
      ++m;
      swap(toes, m, w);
    }
    ++w;
  }
  // Movemos los elementos a la derecha.
  shift(toes, ++m, n, *(shoes + n - 1));

  /* MOVEMOS LOS ZAPATOS */
  m = -1;
  w = 0;
  while(w < n){
    // Comparamos el elemento con el pivote de los pies
    // Si este es menor, intercambiamos el muro con el actual.
    if(*(shoes + w) < *(toes + n - 1)){
      ++m;
      swap(shoes, m, w);
    }
    ++w;
  }
  // Movemos los elementos a la derecha.
  shift(shoes, ++m, n, *(toes + n - 1));

  // Ahora los nuevos pivotes son m0, m1
  // Repetimos el algoritmo
  Zapatos_Pies(shoes, toes, m);
  Zapatos_Pies(shoes + m, toes, n - m);
}
```

2.2 Eficiencia

La eficiencia de los métodos *swap* y *shift* son $O(1)$ y $O(n)$, respectivamente. Realizamos un quicksort para cada conjunto, para los pies y para los zapatos, como referencia el pivote del conjunto opuesto. Definimos como n el total de pies/niños y k cantidad de elementos mayores al pivote.

$$T(n) = \begin{cases} 1 & n \leq 1 \\ n + T(n-k-1) + T(k) & n \geq 2 \wedge 0 \leq k \leq n \end{cases}$$

Según la regla de simetría.

$k = 0, n{-}1 \Rightarrow T(n{-}0{-}1){+}T(0) = T(n{-}(n{-}1){-}1){+}T(n{-}1) = T(0){+}T(n{-}1)$

Aplicando lo dicho,

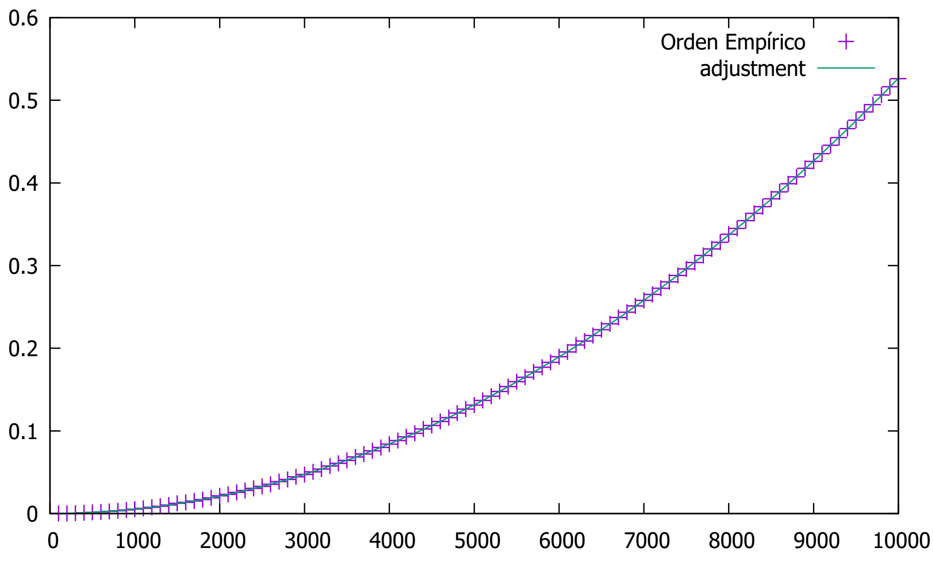
$$T(n) = n + T(0) + T(n-1) = n + 1 + T(n-1)$$

$$T(n) - T(n-1) = 0 \Rightarrow x - 1 = 0, 1^n(n+1)$$

El polinomio resultante y por la regla de simetría:

$$T(n) = 1^n(n^2c_1 + nc_2 + c_3) \Rightarrow \Theta(T(n)) = \Theta(n^2c_1 + nc_2 + c_3) \in \Theta(n^2)$$

2.3 Gráfica



Moda de un conjunto de enteros

3.1 Código

```
std::pair<int, int> Mayor_Frecuencia(int* set, int n){
  int* ht0 = new int[n];
  int* ht1 = new int[n];
  int p = set[0]; // PIVOTE (Elemento inicial al vector).
  int h0 = 0, h1 = 0, m = 1;
  // Si el tamanio es <= 1
  if(n <= 1){
    return std::pair<int, int>(m, p);
  }
  // Metemos segun la homogeneidad de los elementos.
  for(int i = 1; i < n; ++i){
    int df = p - set[i];
    if(df > 0){
      ht0[h0] = set[i];
      ++h0;
    }else if(df < 0){
      ht1[h1] = set[i];
      ++h1;
    }else{
      ++m;
    }
  }
  std::pair<int, int> v0 = Mayor_Frecuencia(ht0, h0);
  std::pair<int, int> v1 = Mayor_Frecuencia(ht1, h1);
  delete[] ht0;
  delete[] ht1;
  // Buscamos en los dos conjuntos heterogeneos (Aquel mayor).
  if(m > v0.first && m > v1.first){
    return std::pair<int, int>(n, p);
  }else if(v0.first > m && v0.first > v1.first){
    return v0;
  }else{
    return v1;
  }
}
```

3.2 Eficiencia

Donde n es el tamaño de la muestra, k las repeticiones al pivote y d los elementos mayores al pivote.

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(n-k-d) + T(d) + n & n \geq 2 \wedge 1 \leq d < k \leq n \end{cases}$$

Diferenciamos dos casos

1. Cuando todos los elementos solo se repiten una vez y no se divide.

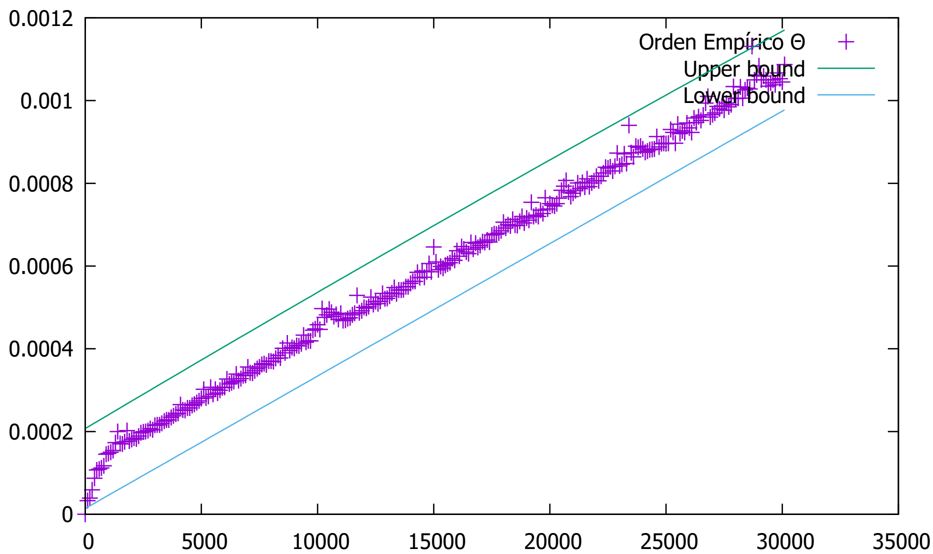
$$k = 1, d = 0 \Rightarrow T(n) = T(n-1) + T(0) + n = T(n-1) + 1 + n$$

   Es parecida al ejercicio anterior, por lo que será $O(n^2)$.

2. $k = n$ Cuando no hay elementos mayores y las repeticiones como n.

$$k = n, d = 0 \Rightarrow T(n) = T(n-n-0) + T(0) + n = 2T(0) + n = n \in \Omega(n)$$

3.3 Gráfica



Especificaciones

1. Windows 10.0.14393
2. Procesador Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz, 3504 Mhz
3. 6 procesadores principales.
4. 12 procesadores lógicos.
5. Memoria física instalada (RAM) 8,00 GB x 2
6. Compilador MinGW.