Práctica 2

Algoritmos Divide y Vencerás

Algoritmo de trasposición de una matriz

Matriz traspuesta

- Dada una matriz A, su traspuesta queda definida:

$$(A^t)_{ij} = A_{ji}, 1 \leq i \leq n, 1 \leq j \leq m$$

$$\begin{bmatrix} 0 & 0 & 4 \\ 1 & 0 & 4 \\ 0 & 1 & 0 \\ 0 & 3 & 2 \\ 0 & 2 & 3 \\ 0 & 3 & 4 \\ 3 & 3 & 1 \end{bmatrix}^t = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 1 & 3 & 2 & 3 & 3 \\ 4 & 4 & 0 & 2 & 3 & 4 & 1 \end{bmatrix}$$

Algoritmo simple

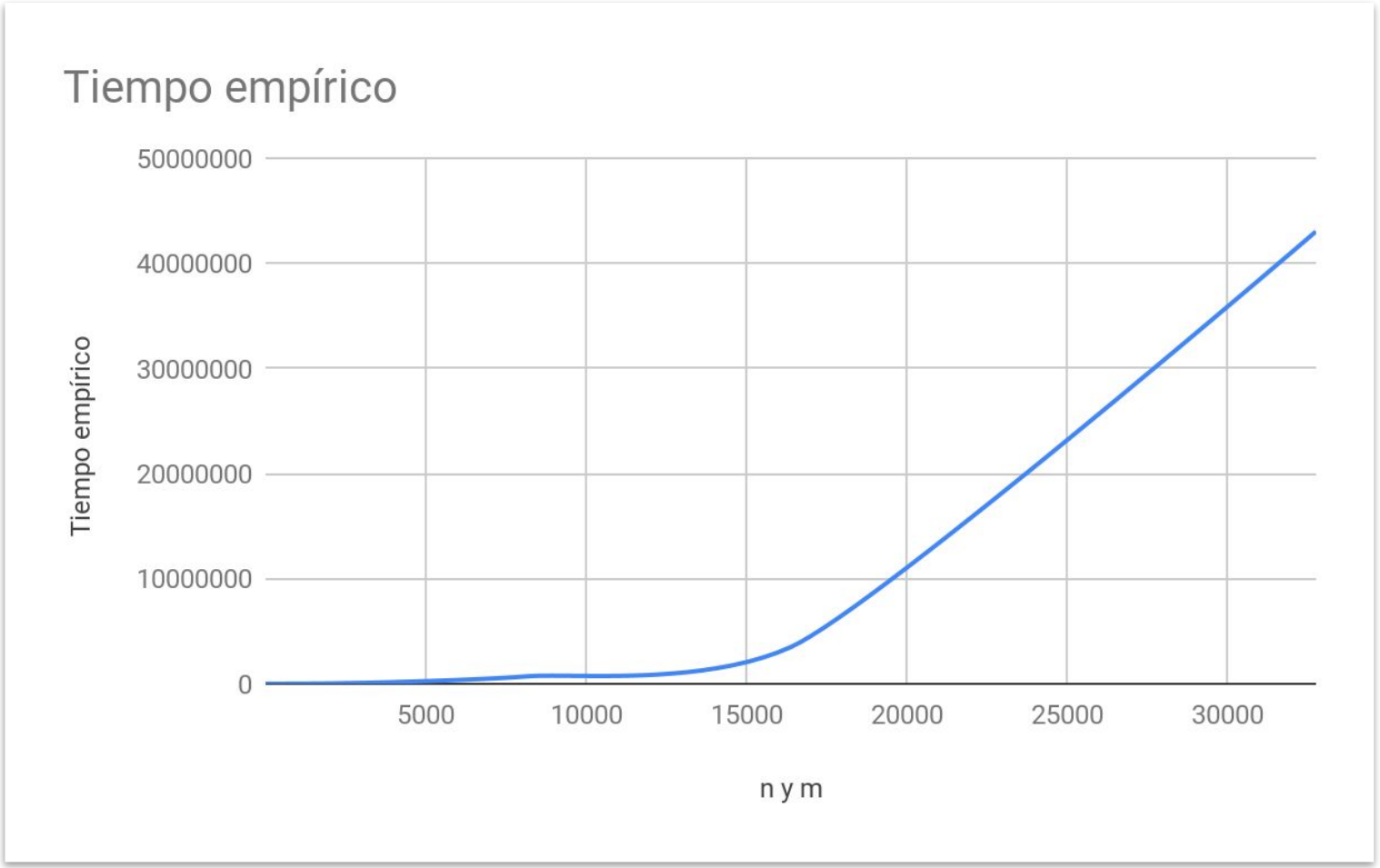
1. Creamos matriz vacía
2. Insertamos cada número de la primera matriz en la nueva en su posición correspondiente.

Eficiencia teórica

En una matriz m por n, insertaremos siempre n*m elementos en la nueva matriz, de lo que deducimos:

- Caso peor: **O(n*m)**
- Caso medio: **Θ(n*m)**
- Caso mejor: **Ω(n*m)**

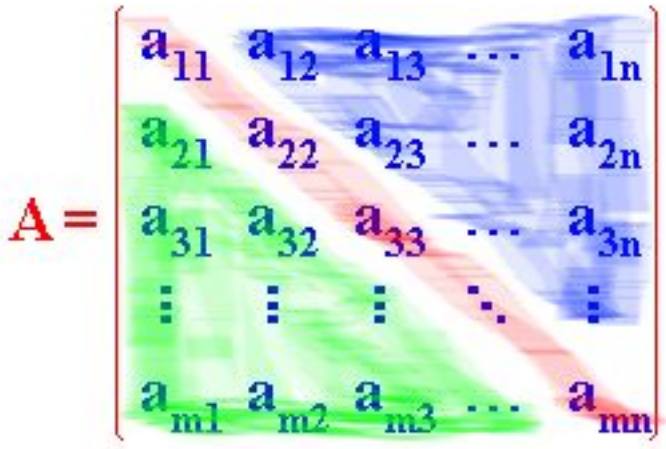
Eficiencia Empírica



Eficiencia híbrida

Algoritmo Divide y Vencerás

Dividimos la matriz en tres partes:



Algoritmo Divide y Vencerás

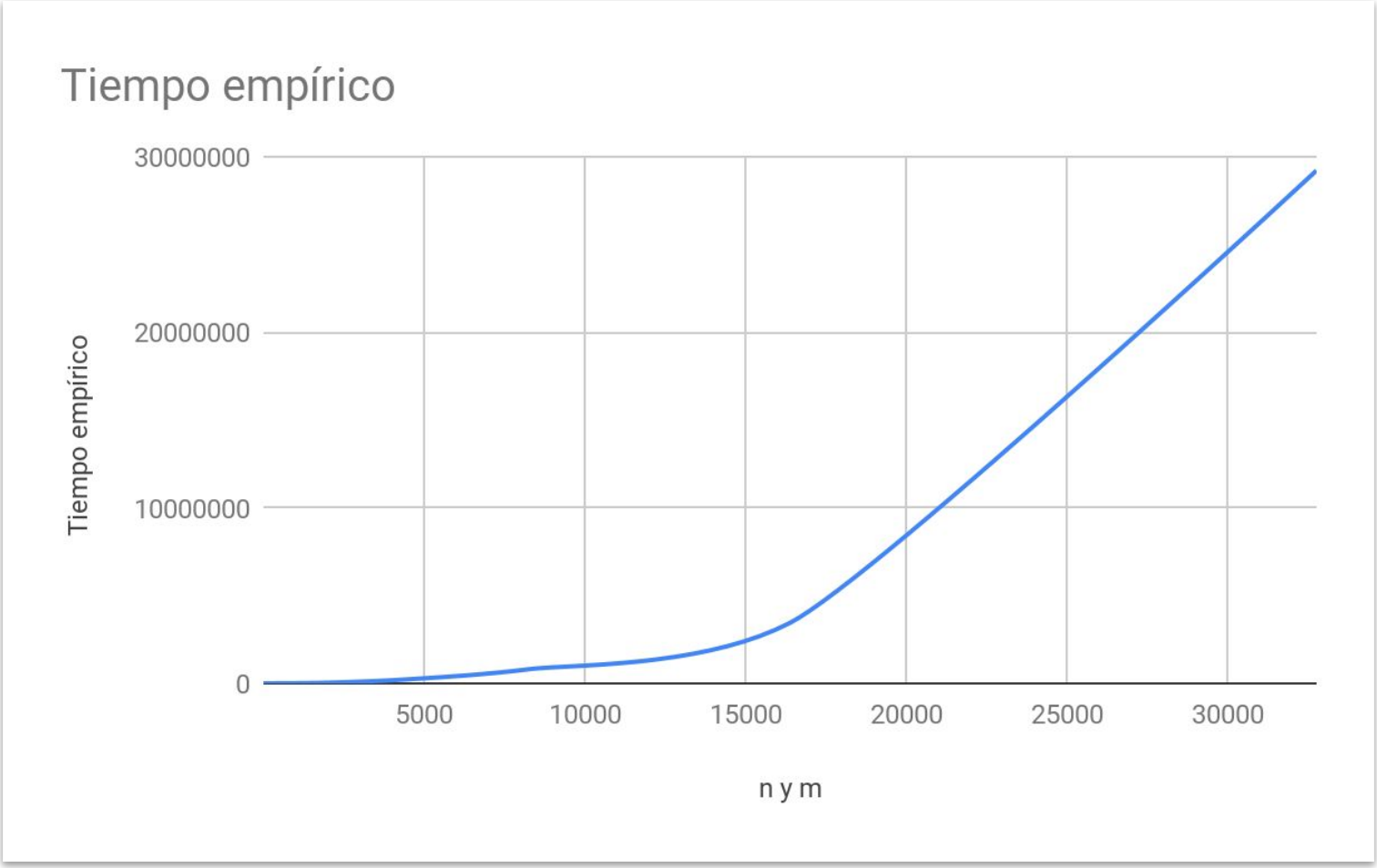
1. Crear matriz vacía
2. Copiar diagonal
3. Insertar triángulo superior en la matriz traspuesta
4. Insertar triángulo inferior en la matriz traspuesta

Eficiencia teórica

- Paso 1: **O(1)**
- Paso 2: **O(n)**
- Paso 3: **O((n² –n) /2)**

Utilizando la regla del máximo,
el algoritmo es **O(n²)**

Eficiencia empírica



Eficiencia híbrida

Comparación

Ambas son $O(n^2)$

Caso de ejecución

```
migue@migue:~/Escritorio/ALG/practica2$ ./bin/traspuesta2 4 4
MATRIZ ORIGINAL
52526   56524   1185   83958
45044   23651   83644   37343
54522   28131   47929   14791
50944   53488   54249   87134

1. Se copia la diagonal
52526   0   0   0
0   23651   0   0
0   0   47929   0
0   0   0   87134

2. Se traspone el triangulo superior
52526   0   0   0
56524   23651   0   0
1185   83644   47929   0
83958   37343   14791   87134

3. Se traspone el triangulo inferior
52526   45044   54522   50944
56524   23651   28131   53488
1185   83644   47929   54249
83958   37343   14791   87134

TRASPUESTA
52526   45044   54522   50944
56524   23651   28131   53488
1185   83644   47929   54249
83958   37343   14791   87134
```

Algoritmo de búsqueda en una serie unimodal

Serie unimodal

Enunciado:

Vector v de tamaño n
Índice p tal que $0<p<n-1$
Ordenado crecientemente de los índices 0 a p
Ordenado decrecientemente de los índices p a $n-1$
Objetivo: encontrar p

Algoritmo simple

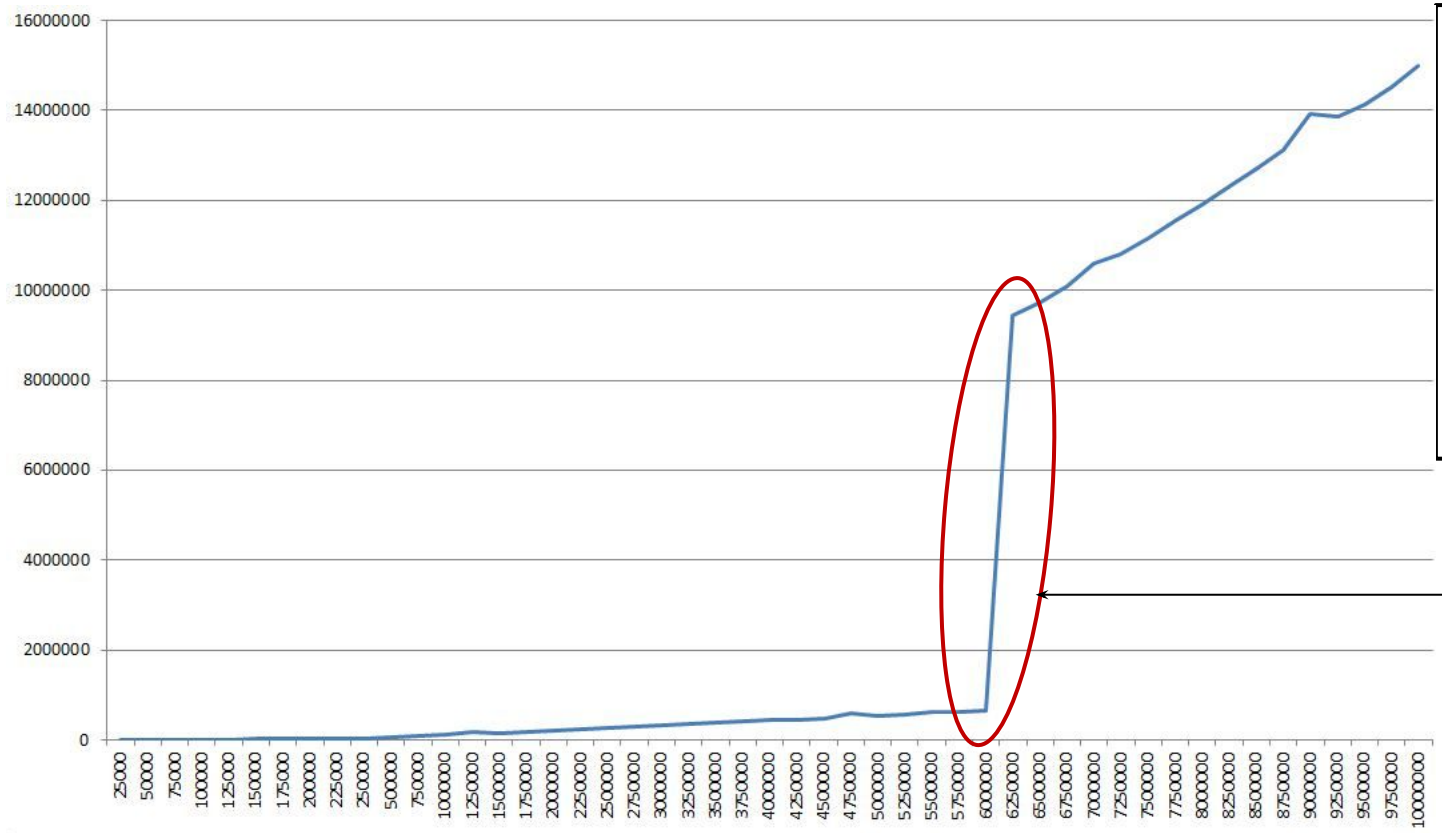
1. Recorremos los índices del vector en orden usual
2. Si $v[i]>v[i+1]$, solución encontrada: $p=i$

Eficiencia teórica

Nos encontramos ante una búsqueda lineal:

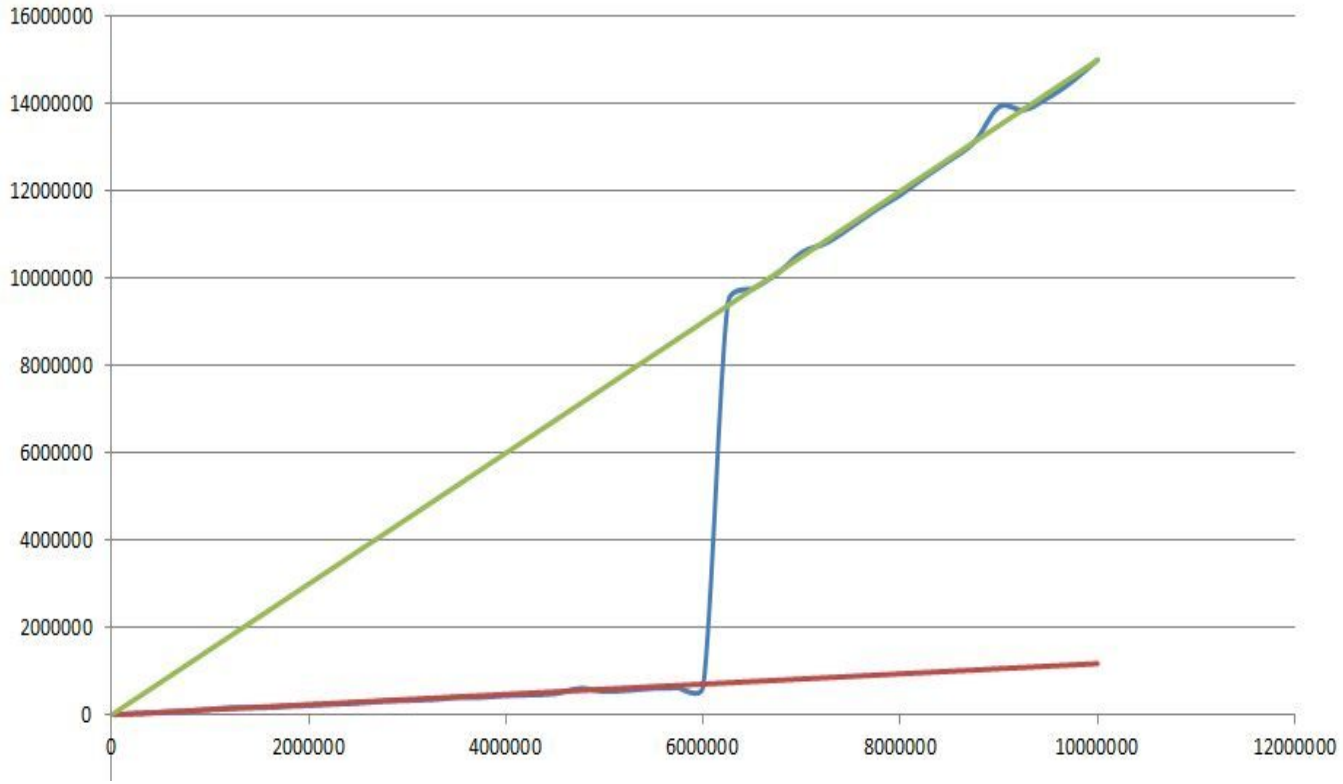
- Caso peor: **O(n)**
- Caso medio: **Θ(n/2)**
- Caso mejor: **Ω(1)**

Eficiencia empírica



Discontinuidad producida por un cambio en la localización del almacenamiento en memoria del vector

Eficiencia híbrida



Dos ajustes lineales para las dos partes de la gráfica empírica

Algoritmo divide y vencerás

Dado un índice k:

- Si v tiene máximo en k: $p=k$
- Si v creciente en k: $p>k$
- Si v decreciente en k: $p<k$

Algoritmo divide y vencerás

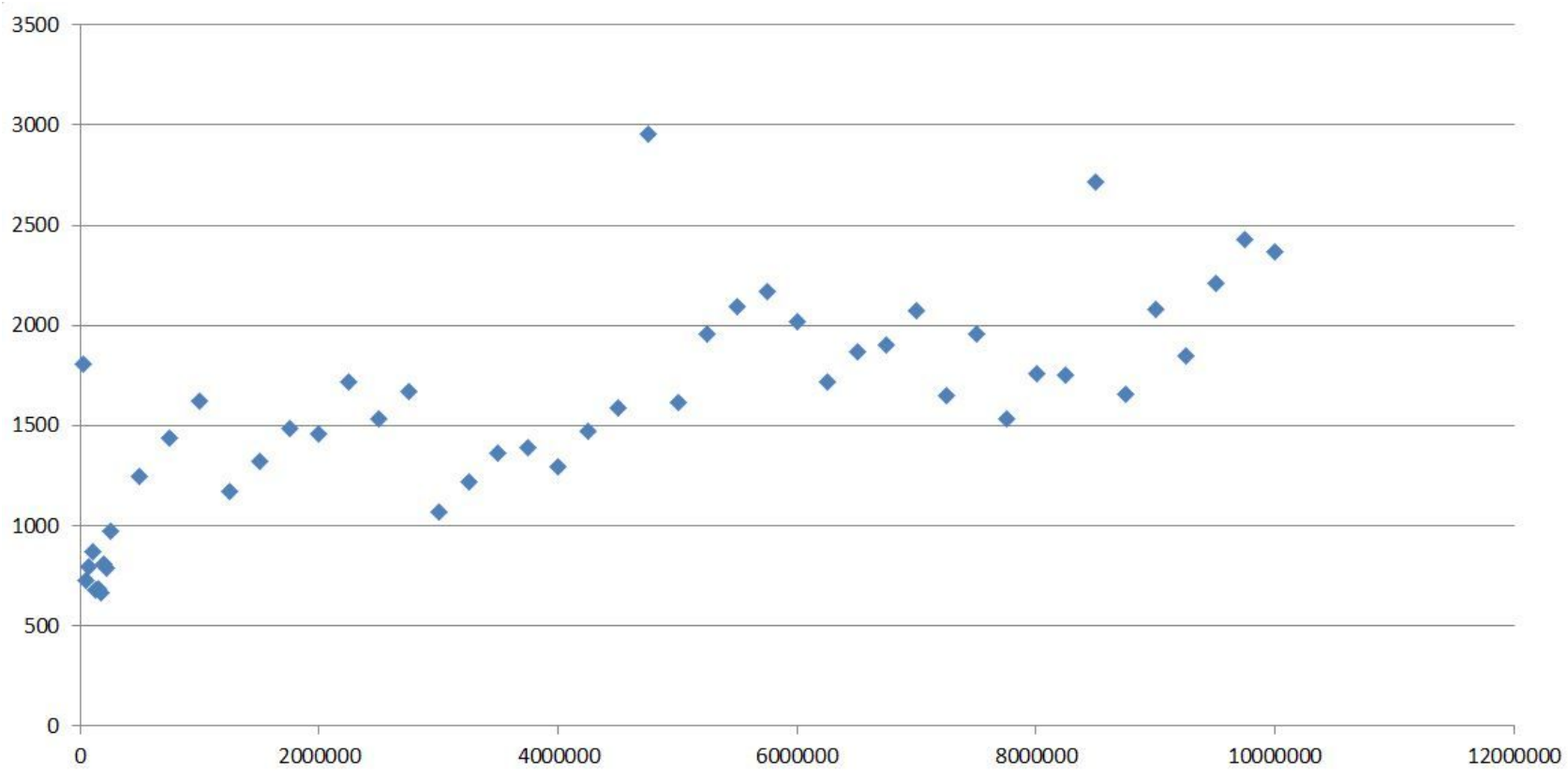
1. *k=n/2*
2. *Estudiamos k:*
   1. *Si p=k, solución encontrada*
   2. *Si p!=k,* volvemos al paso 1 con la mitad del vector en la que sabemos que está *p*

Es un algoritmo de BÚSQUEDA BINARIA

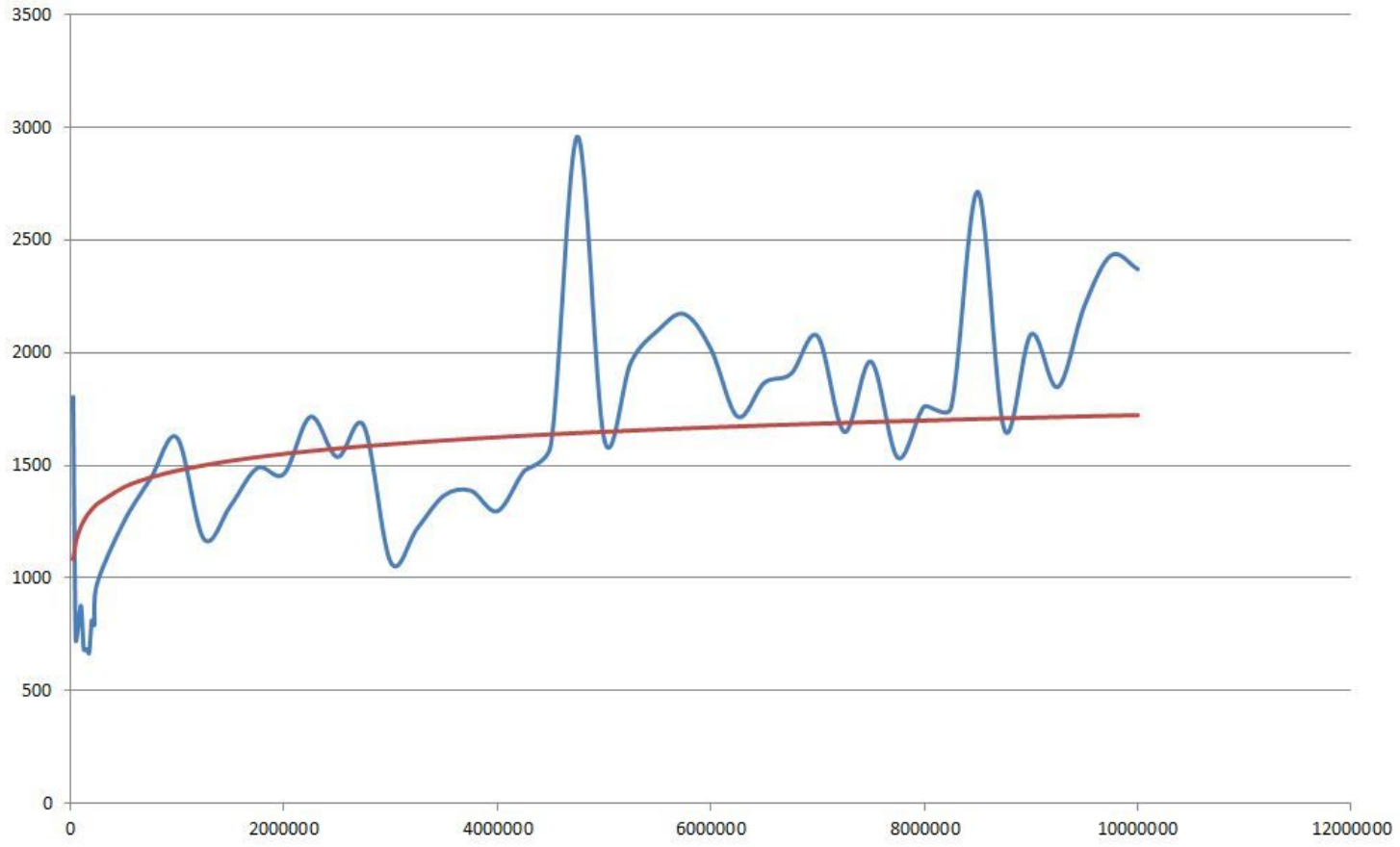
Eficiencia teórica

- Caso peor: $\mathbf{O(\log_2(n))}$
- Caso medio: $\mathbf{\Theta(\log_2(n)-2)}$
- Caso mejor: $\mathbf{\Omega(1)}$

Eficiencia empírica

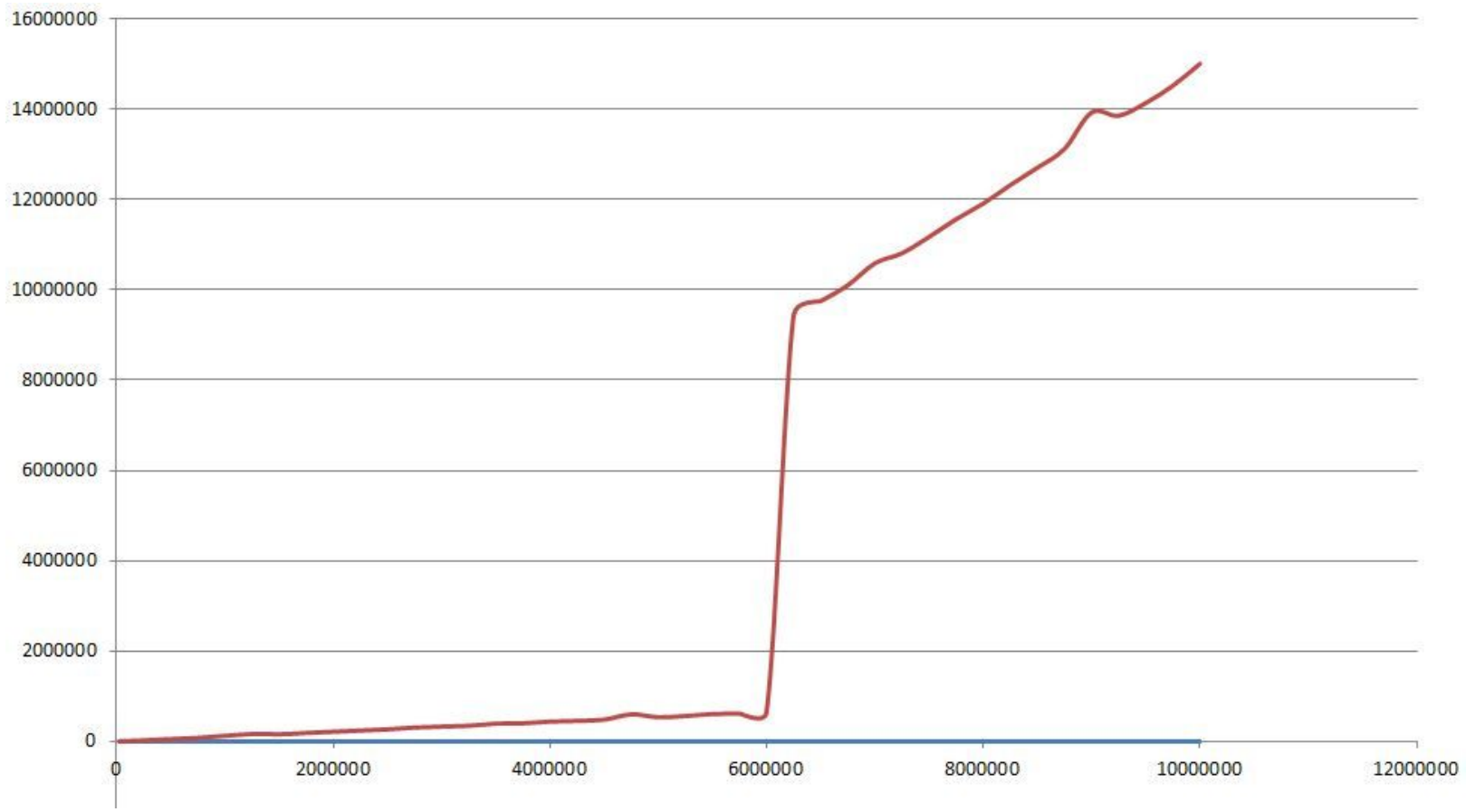


Eficiencia híbrida



Comparación

DyV claramente mejor



Caso de ejecución

```
migue@migue:~/Escritorio/ALG/practica2$ ./bin/serie-unimodal 30
GENERAMOS EL VECTOR Y P
P = 19
UMBRAL = 5

Vector actual: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 29 28 27 26 25 24 23 22 21 20 19
P EN MITAD DERECHA DEL VECTOR

Vector actual: 15 16 17 18 29 28 27 26 25 24 23 22 21 20 19
P EN MITAD IZQUIERDA DEL VECTOR

Vector actual: 15 16 17 18 29 28
P EN MITAD DERECHA DEL VECTOR

Vector actual: 18 29 28
N < UMBRAL -> FUERZA BRUTA
P ENCONTRADO, P = 19

TAM = 30        t = 98136
```