

# Algorítmica: práctica 2

## Mezclando $k$ vectores ordenados

Sofía Almeida Bruno  
Antonio Coín Castro  
María Victoria Granados Pozo  
Miguel Lentisco Ballesteros  
José María Martín Luque

Grupo 2

6 de abril de 2017

# Objetivo

- Diseño de un algoritmo *divide y vencerás* que se encargue de combinar  $k$  vectores ordenados.
- Análisis de su eficiencia teórica, empírica e híbrida.
- Comparación con un algoritmo clásico.

# Fuerza Bruta

Hemos realizado dos algoritmos mediante la fuerza bruta:

- Método clásico
- Método alternativo

## Eficiencia teórica

Ecuación general:

$$T(k) = \sum_{i=2}^{k-1} ni \sim \frac{n}{2}k^2$$

Orden de eficiencia:

$$O(nk^2) \sim O(k^2)$$

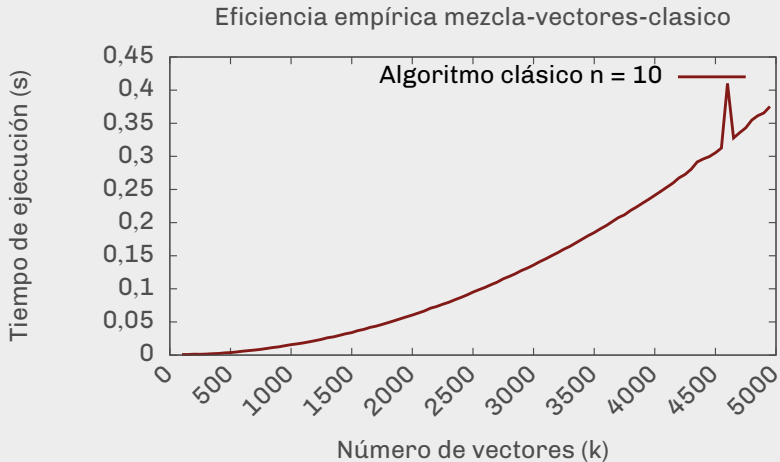
## Método Clásico - Mezcla de 2 vectores

```
1 void merge(int T1[], int T2[], int S[], int n1, int n2) {
2     int p1 = 0, p2 = 0, p3 = 0;
3
4     while (p1 < n1 && p2 < n2) {
5         if (T1[p1] <= T2[p2]) {
6             S[p3] = T1[p1];
7             p1++;
8         }
9         else {
10            S[p3] = T2[p2];
11            p2++;
12        }
13
14        p3++;
15    }
16
17    while (p1 < n1) {
18        S[p3++] = T1[p1++];
19    }
20
21    while (p2 < n2) {
22        S[p3++] = T2[p2++];
23    }
24 }
```

# Método Clásico

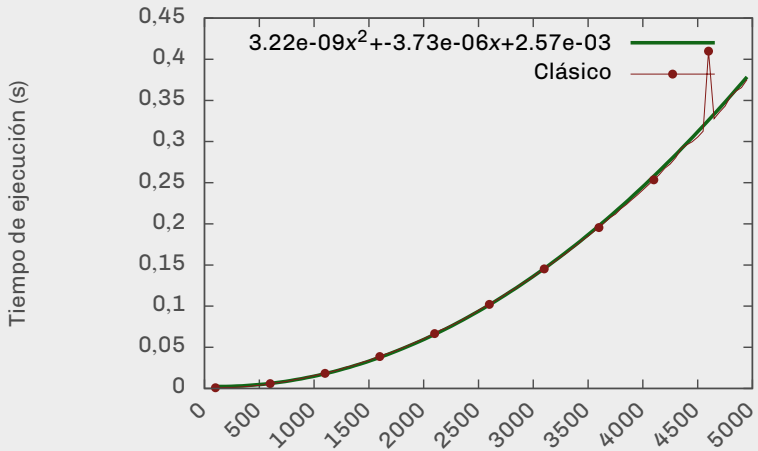
```
1  int* mezcla_vectores(int** T, int k, int n) {
2      int* S = new int[k*n]; // Vector mezcla
3      assert(S);
4
5      if (k > 1) {
6          int* aux = new int [k*n];
7          assert(aux);
8
9          // Primera mezcla
10         merge(T[0], T[1], S, n, n);
11
12         // Resto de mezclas
13         for (int i = 2; i < k; i++) {
14             merge(S, T[i], aux, i*n, n);
15             swap(S, aux); // Intercambiamos punteros
16         }
17
18         delete [] aux;
19     }
20
21     else {
22         for (int i = 0; i < n; i++) {
23             S[i] = T[0][i];
24         }
25     }
26
27     return S;
28 }
```

# Eficiencia empírica



# Eficiencia híbrida

Ajuste algoritmo clásico

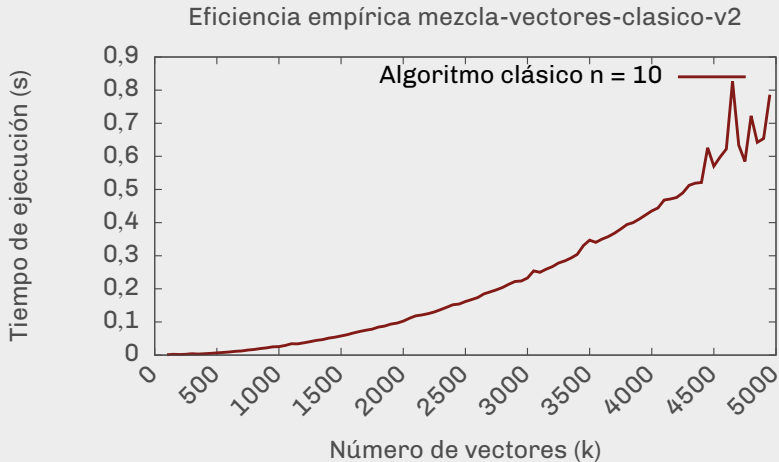




# Método Alternativo

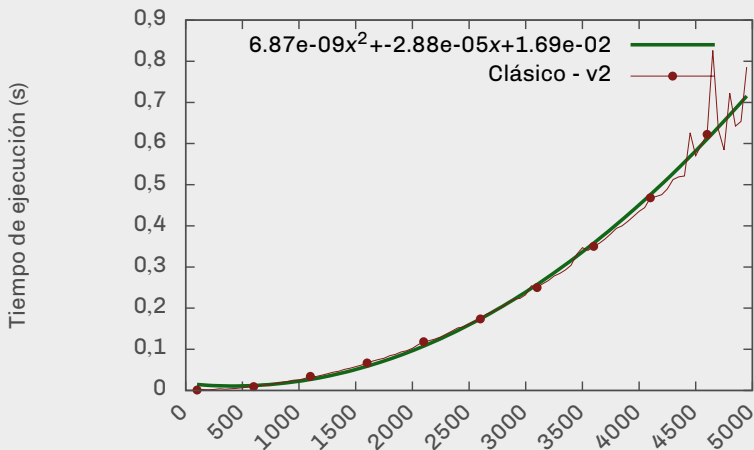
```
1 void mezclarK(int** T, int* res, int n, int k)
2 {
3     // Tamaño total del vector
4     int N = n * k;
5     // Vector donde vamos a guardar el indice del valor que falta por meter de cada vector
6     int* v_indices = new int[k];
7     // Inicializamos al último indice de cada vector (los máximos)
8     for (int i = 0; i < k; ++i)
9         v_indices[i] = n - 1;
10    // El bucle dura hasta N = nk colocados; empezamos en N y acabamos en -1
11    int indice_colocar = N - 1;
12    while (indice_colocar >= 0)
13    {
14        // Este indice indica que vector es el que contiene el valor max.
15        int indice_max = 0;
16        // Recorremos buscando el indice que contenga el valor mayor
17        for (int i = 0; i < k; ++i)
18            if (T[indice_max][v_indices[indice_max]] < T[i][v_indices[i]])
19                indice_max = i;
20        // Guardamos en res con el valor correspondiente
21        res[indice_colocar] = T[indice_max][v_indices[indice_max]];
22        // Decrementamos el indice correspondiente al vector que acabamos de usar
23        --v_indices[indice_max];
24        // Hemos colocado uno
25        --indice_colocar;
26    }
27    delete [] v_indices;
28 }
```

# Eficiencia empírica



# Eficiencia híbrida

Ajuste algoritmo clásico - v2



# Divide y Vencerás

Hemos realizado dos algoritmos usando la técnica Divide y Vencerás:

- Vectores dinámicos
- Vectores de la STL

# Eficiencia teórica

Ecuación general:

$$T(k) = 2T\left(\frac{k}{2}\right) + n\frac{k}{2}$$

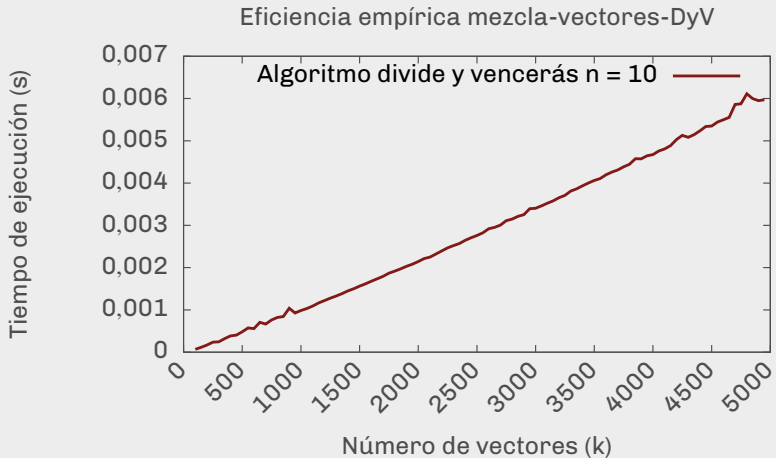
Orden de eficiencia:

$$O(nk \log k) \sim O(k \log k)$$

# Vectores Dinámicos

```
1  int* mezclaDV(int** T, int n, int start, int end) {
2      int k = end - start + 1; // Número de vectores
3
4      // Caso base
5      if (k == 1) {
6          return T[start];
7      }
8
9      // Caso general
10     else {
11         int middle = (start + end) / 2;
12         int n1 = middle - start + 1;
13         int n2 = end - (middle + 1) + 1;
14
15         // Divide
16         int* izqda = mezclaDV(T, n, start, middle);
17         int* dcha = mezclaDV(T, n, middle + 1, end);
18
19         // Vencerás
20         return merge(izqda, dcha, n * n1, n * n2);
21     }
22 }
```

# Eficiencia empírica



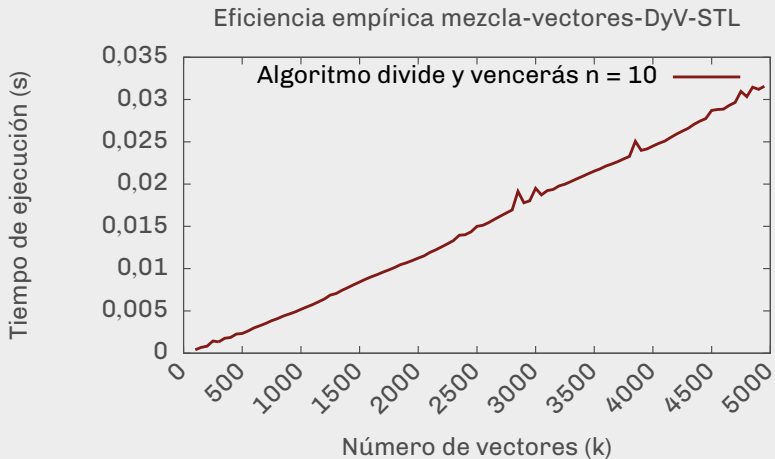
## Eficiencia híbrida



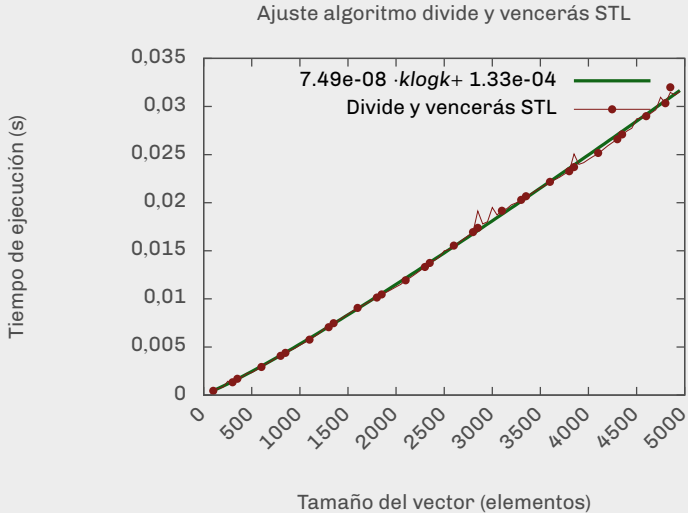
# Vectores de la STL

```
1  vector<int> mezclaDV(vector<vector<int>> vectores) {
2
3      // Casos base
4      if (vectores.size() < 1) {
5          vector<int> sol;
6          return sol;
7      } else if (vectores.size() == 1) {
8          return vectores[0];
9      } else if (vectores.size() == 2) {
10         return merge(vectores[0], vectores[1]);
11     }
12
13     vector<vector<int>>::iterator half = vectores.begin() + vectores.size() / 2;
14     vector<vector<int>> firstHalf(vectores.begin(), half), secondHalf(half + 1, vectores.end());
15
16     // Divide
17     vector<int> s1 = mezclaDV(firstHalf);
18     vector<int> s2 = mezclaDV(secondHalf);
19
20     // Vencerás
21     return merge(s1, s2);
22 }
```

# Eficiencia empírica



# Eficiencia híbrida



# Comparación eficiencias

