# Algorítmica: práctica 2

Mezclando $k$ vectores ordenados

Grupo 2

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

6 de abril de 2017

## Introducción

El objetivo de esta práctica es diseñar un algoritmo *divide y vencerás* que se encargue de combinar $k$ vectores ordenados. Además tenemos que analizar su eficiencia, implementarlo y compararlo con un algoritmo clásico.

## Algoritmo clásico

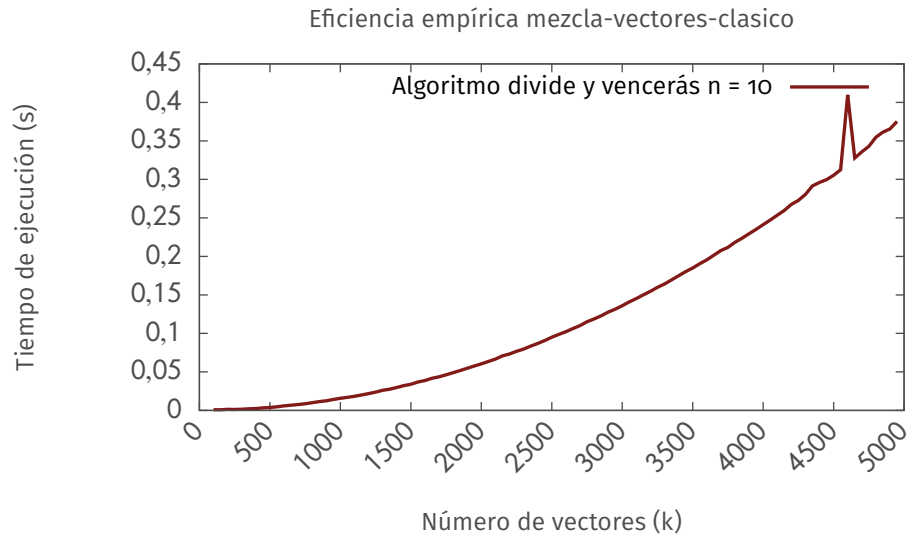
A continuación se proporciona el código de la función `mezcla_vectores`, que utiliza un algoritmo clásico para mezclar $k$ vectores en uno solo. El código del programa completo se puede encontrar en la carpeta *src*.

```
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 teórica

### Eficiencia empírica

En el gráfico que se muestra a continuación se muestran los resultados de la ejecución del algoritmo *clásico* con vectores de 10 elementos.




**Eficiencia *híbrida***

## Algoritmo *divide y vencerás* con vectores dinámicos

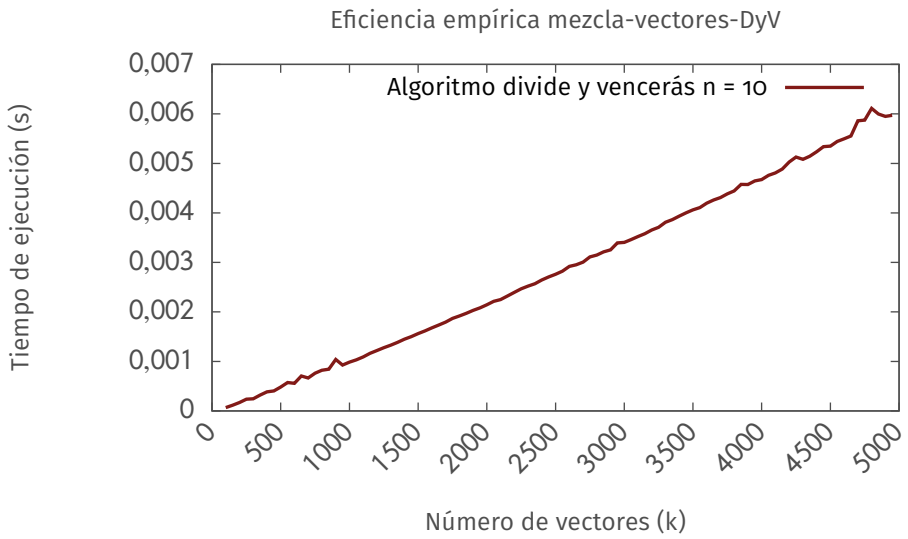
A continuación se proporciona el código de la función `mezclaDV`, que utiliza un algoritmo divide y vencerás (con vectores dinámicos) para mezclar $k$ vectores en uno solo. El código del programa completo se puede encontrar en la carpeta *src*.

```
int* mezclaDV(int** T, int n, int start, int end) {
  int k = end - start + 1;  // Número de vectores

  // Caso base
  if (k == 1) {
    return T[start];
  }

  // Caso general
  else {
    int middle = (start + end) / 2;
    int n1 = middle - start + 1;
    int n2 = end - (middle + 1) + 1;

    // Divide
    int* izqda = mezclaDV(T, n, start, middle);
    int* dcha = mezclaDV(T, n, middle + 1, end);

    // Vencerás
    return merge(izqda, dcha, n * n1, n * n2);
  }
}
```

### Eficiencia teórica

### Eficiencia empírica

En el gráfico que se muestra a continuación se muestran los resultados de la ejecución del algoritmo *divide y vencerás* con vectores dinámicos de 10 elementos.




### Eficiencia *híbrida*

## Algoritmo *divide y vencerás* con vectores de la STL

A continuación se proporciona el código de la función `mezclaDV`, que utiliza un algoritmo divide y vencerás (con vectores de la STL) para mezclar $k$ vectores en uno solo. El código del programa completo se puede encontrar en la carpeta *src*.

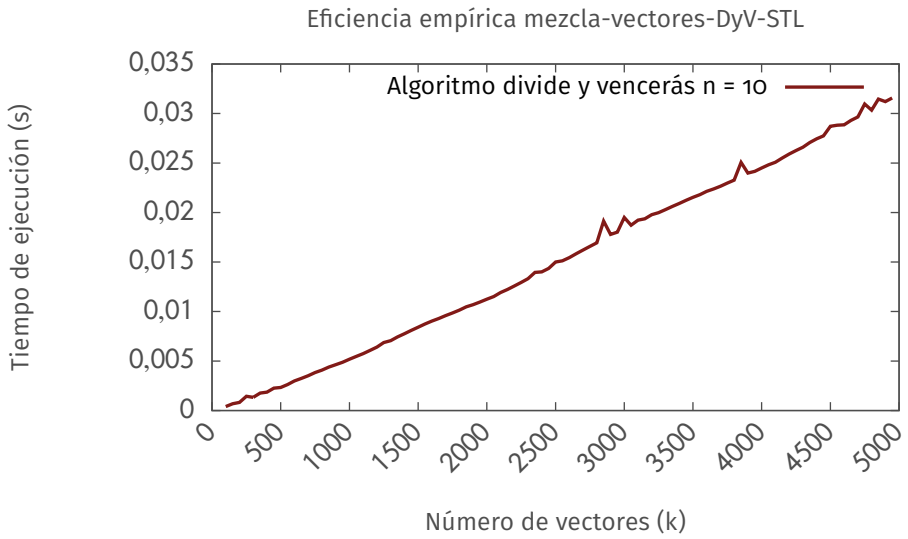
```
vector<int> mezclaDV(vector<vector<int>> vectores) {

    // Casos base
    if (vectores.size() < 1) {
        vector<int> sol;
        return sol;
    } else if (vectores.size() == 1) {
        return vectores[0];
    } else if (vectores.size() == 2) {
        return merge(vectores[0], vectores[1]);
    }

    vector<vector<int>>::iterator half = vectores.begin() + vectores.size() / 2;
    vector<vector<int>> firstHalf(vectores.begin(), half), secondHalf(half + 1, vectores.end());

    // Divide
    vector<int> s1 = mezclaDV(firstHalf);
    vector<int> s2 = mezclaDV(secondHalf);

```

```
20      // Vencerás
21      return merge(s1, s2);
22  }
```

### Eficiencia teórica

### Eficiencia empírica

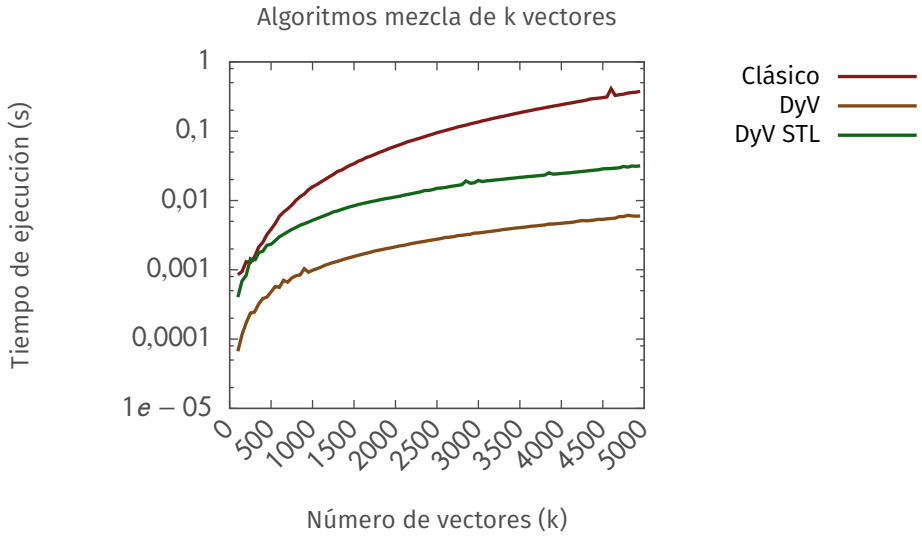
En el gráfico que se muestra a continuación se muestran los resultados de la ejecución del algoritmo *divide y vencerás* con vectores `std::vector` de 10 elementos.




### Eficiencia *híbrida*

## Comparación de la eficiencia

En el siguiente gráfico se puede observar de forma visual qué algoritmo es más eficiente. Como es de esperar, el algoritmo clásico es el más lento de todos. Algo más curioso quizás es que el algoritmo que utiliza vectores *dinámicos* es más rápido que el que usa la clase `vector` de la STL.


Algoritmos mezcla de k vectores
Tiempo de ejecución (s)
1
0,1
0,01
0,001
0,0001
1e − 05
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
Número de vectores (k)
Clásico
DyV
DyV STL

## Anexo

### Características de los ordenadores donde se ha ejecutado

1. Apple MacBook Pro, Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz, 8GB RAM.
   Compilador: clang-800.0.38
   Sistema operativo: macOS Sierra

2. Dell XPS 13, Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, 8GB RAM.
   Compilador: g++ 6.3.1
   Sistema operativo: Arch Linux