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

27 de abril de 2017

Objetivo

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

Fuerza Bruta

Hemos realizado dos algoritmos mediante fuerza bruta:

- Método clásico
- Método alternativo

Método Clásico - Mezcla de 2 vectores

```
void merge(int T1[], int T2[], int S[], int n1, int n2) {
  int p1 = 0, p2 = 0, p3 = 0;

  while (p1 < n1 && p2 < n2) {
    if (T1[p1] <= T2[p2]) {
      S[p3] = T1[p1];
      p1++;
    }
    else {
      S[p3] = T2[p2];
      p2++;
    }

    p3++;
  }

  while (p1 < n1) {
    S[p3++] = T1[p1++];
  }

  while (p2 < n2) {
    S[p3++] = T2[p2++];
  }
}
```

Método Clásico

```
int* mezcla_vectores(int** T, int k, int n) {
  int* S = new int[k*n];  // Vector mezcla
  assert(S);

  if (k > 1) {
    int* aux = new int [k*n];
    assert(aux);

    // Primera mezcla
    merge(T[0], T[1], S, n, n);

    // Resto de mezclas
    for (int i = 2; i < k; i++) {
      merge(S, T[i], aux, i*n, n);
      swap(S, aux);   // Intercambiamos punteros
    }

    delete [] aux;
  }

  else {
    for (int i = 0; i < n; i++) {
      S[i] = T[0][i];
    }
  }

  return S;
}
```

Eficiencia teórica

Consideramos que n es una constante fija.

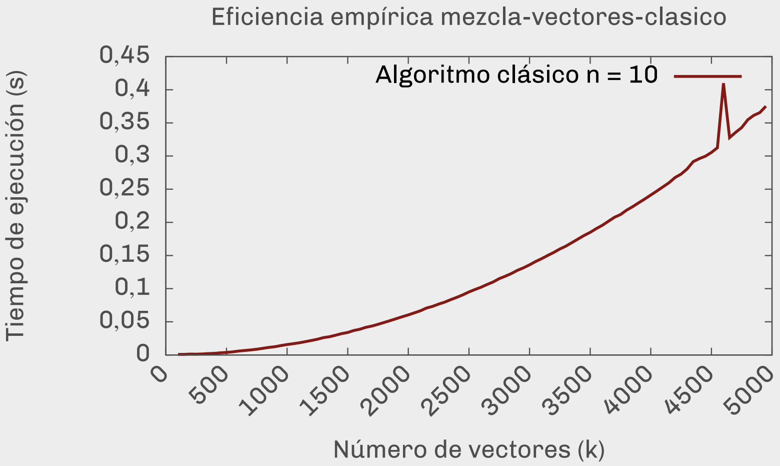
Ecuación general:

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

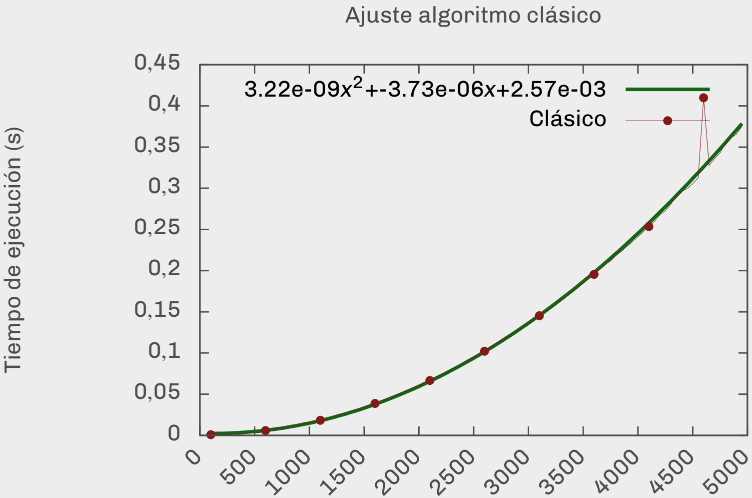
Orden de eficiencia:

$$O\left(\frac{n}{2}k^2\right) \sim O(k^2)$$

Eficiencia empírica



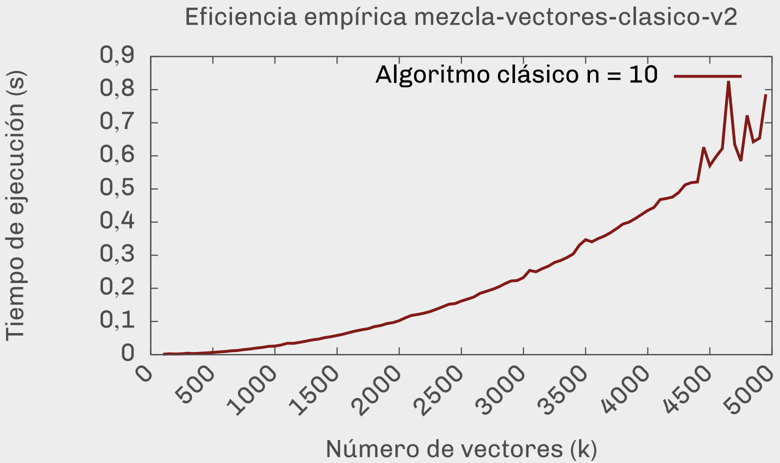
Eficiencia híbrida



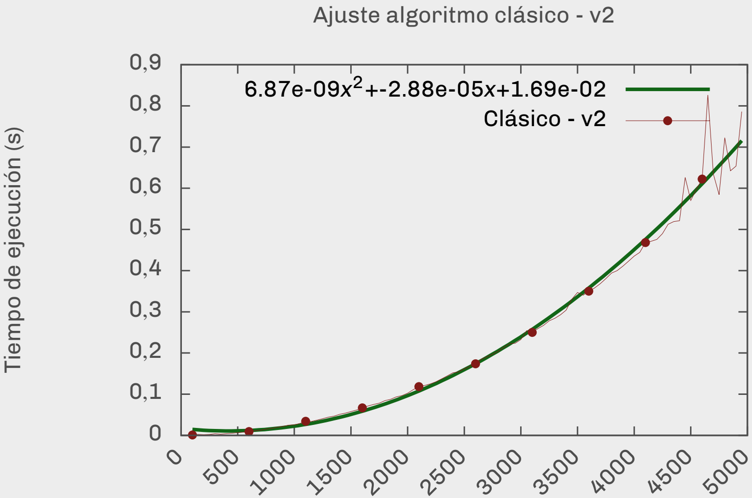
Método Alternativo

```
void mezclarK(int** T, int* res, int n, int k)
{
  // Tamaño total del vector
  int N = n * k;
  // Vector donde vamos a guardar el índice del valor que falta por meter de cada vector
  int* v_indices = new int[k];
  // Inicializamos al último índice de cada vector (los máximos)
  for (int i = 0; i < k; ++i)
    v_indices[i] = n - 1;
  // El bucle dura hasta N = nk colocados; empezamos en N y acabamos en -1
  int indice_colocar = N - 1;
  while (indice_colocar >= 0)
  {
    // Este índice indica que vector es el que contiene el valor max.
    int indice_max = 0;
    // Recorremos buscando el índice que contenga el valor mayor
    for (int i = 0; i < k; ++i)
      if (T[indice_max][v_indices[indice_max]] < T[i][v_indices[i]])
        indice_max = i;
    // Guardamos en res con el valor correspondiente
    res[indice_colocar] = T[indice_max][v_indices[indice_max]];
    // Decrementamos el índice corresponiente al vector que acabamos de usar
    --v_indices[indice_max];
    // Hemos colocado uno
    --indice_colocar;
  }
  delete [] v_indices;
}
```

Eficiencia empírica



Eficiencia híbrida



Divide y Vencerás

Hemos realizado dos algoritmos usando la técnica Divide y Vencerás:

- Vectores dinámicos
- Vectores de la STL

Vectores Dinámicos

```
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

Consideramos que n es una constante fija.

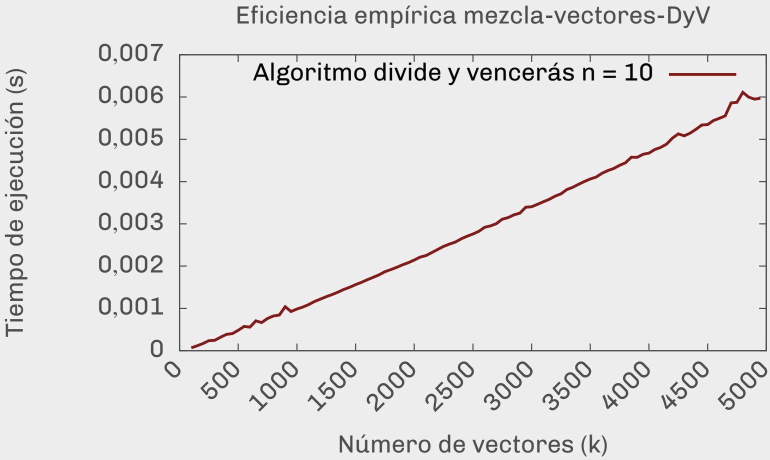
Ecuación general:

$$\begin{cases} T(1) = 1 \\ T(k) = 2T\left(\frac{k}{2}\right) + n\frac{k}{2} \end{cases}$$

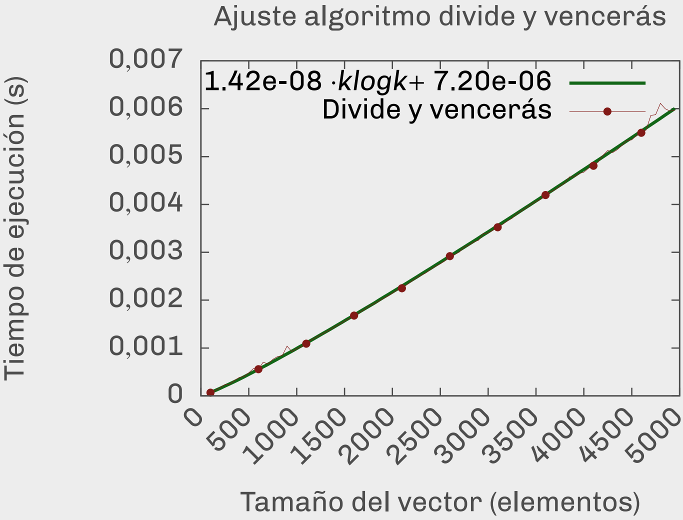
Orden de eficiencia:

$$O\left(\frac{n}{2}k \log k\right) \sim O(k \log k)$$

Eficiencia empírica



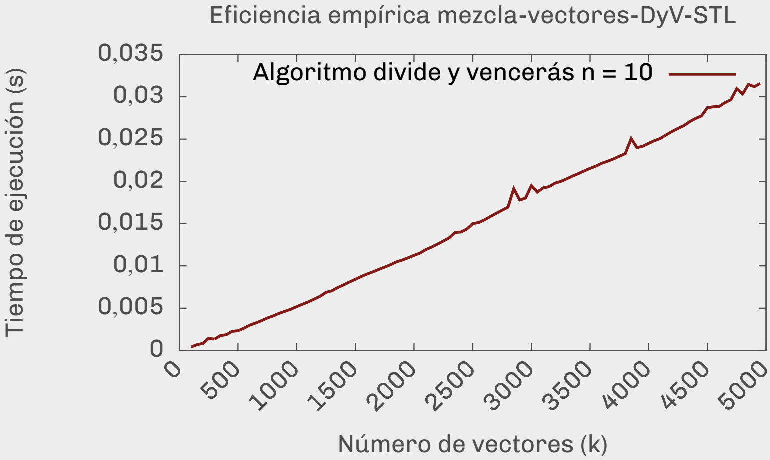
Eficiencia híbrida



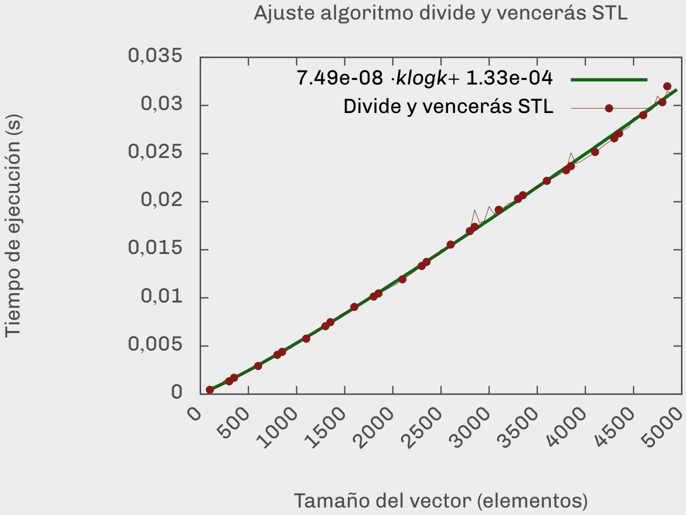
Vectores de la STL

```
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);

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

Eficiencia empírica



Eficiencia híbrida



Comparación eficiencias