ALGORÍTMICA

Práctica 2: Algoritmos Divide y Vencerás

- mezclando k vectores ordenados -

Grupo de prácticas C3

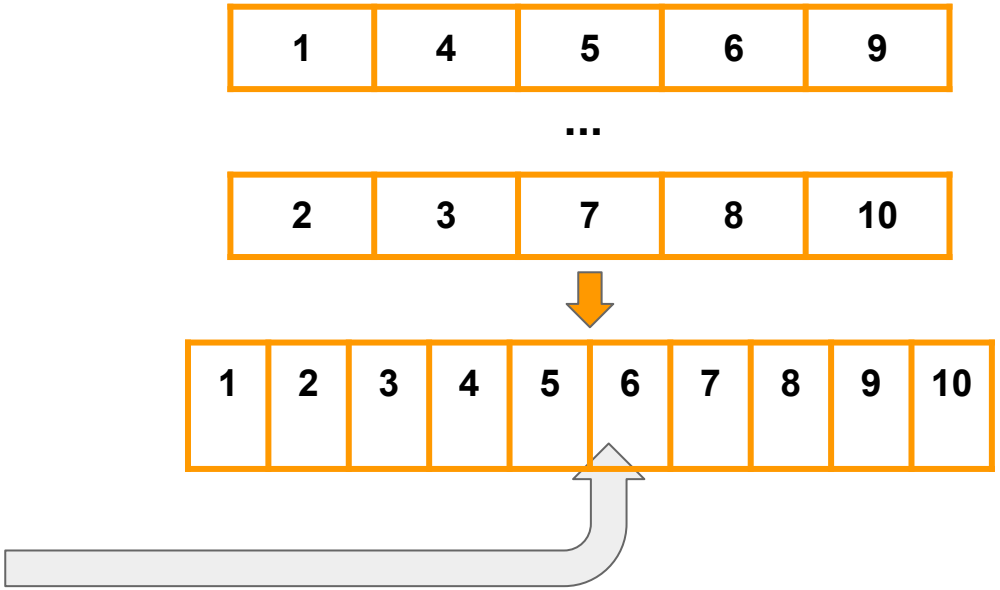
Grupo 2

ÍNDICE

1. Definición del problema

Ejercicio 4: mezclando k vectores ordenados

Tenemos ‘k’ **vectores ordenados crecientes** compuesto por **n enteros**.



Combinarlos en un **único vector k*n ordenado.**

2. Resolución: Fuerza Bruta

Opción 1

Código C++ :

- *Introducimos la matriz en el vector*
- *Ordenamos el vector con el algoritmo Burbuja*

```
for (int i=0; i<k; i++) {
 for (int j=0; j<n; j++){
  numero = T[i][j];
  res.push_back(numero);
  }
}

if (k>=2){
// Burbuja

 for(int i=0;i<res.size()-1;i++)
    for (int j=i+1; j<res.size(); j++)
    if(res.at(i) > res.at(j)){
    numero=res.at(i);
    res.at(i) = res.at(j);
    res.at(j) = numero;
    }
}
```

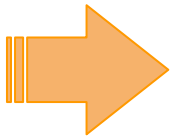
2. Resolución: Fuerza Bruta

Opción 1

1	2	6	9
4	7	8	9
2	5	7	8

K = 3 N = 4

Si K=1, devolvemos el vector ordenado por defecto



K * N

1	2	6	9	4	7	8	9	2	5	7	8

Método de ordenación Burbuja

1	2	2	4	5	6	7	7	8	8	9	9

2. Resolución: Fuerza Bruta

Opción 2

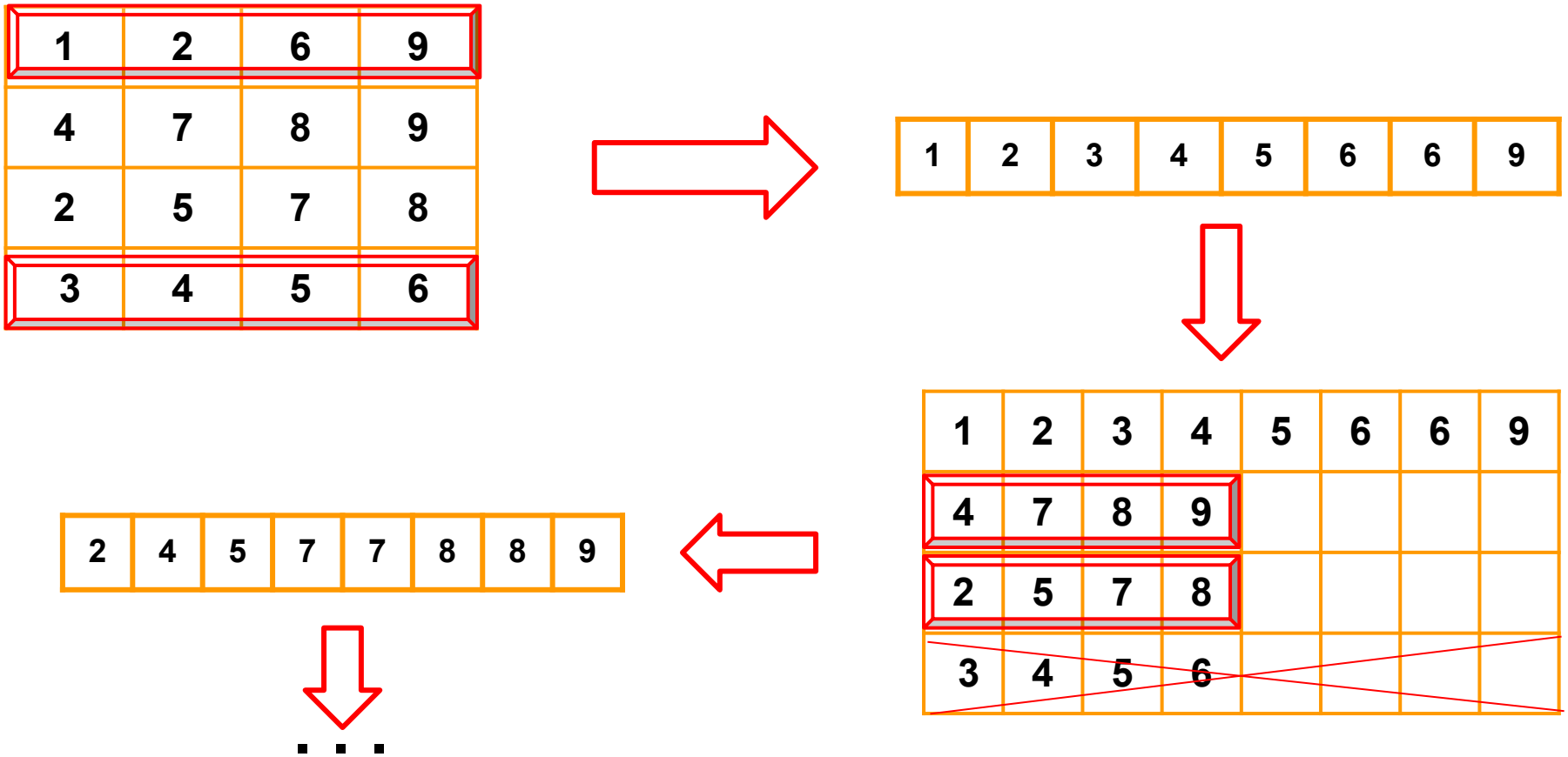
Código C++ :

- Requiere la matriz de vectores
- Mezcla y ordena el primer vector y último
- Termina cuando queda un único vector

```
vector<int>  mergeKVectors(vector<vector<int> > &arr) {
        int last = arr.size()-1;

        // Repetir hasta que nos quede un unico vector
        while (last != 0) {
        int i = 0, j = last;

        // (i, j) forms a pair
        while (i < j) {

        vector<int> aux = SortedMerge(arr.at(i), arr.at(j));

        arr.at(i) = aux;

        // pasamos al siguiente par de vectores
        i++, j--;

        // Si todos los vectores estan fusionados, actualizamos last
        if (i >= j)
        last = j;
        }
        }

        return arr.at(0);
}
```

2. Resolución: Fuerza Bruta

Opción 2



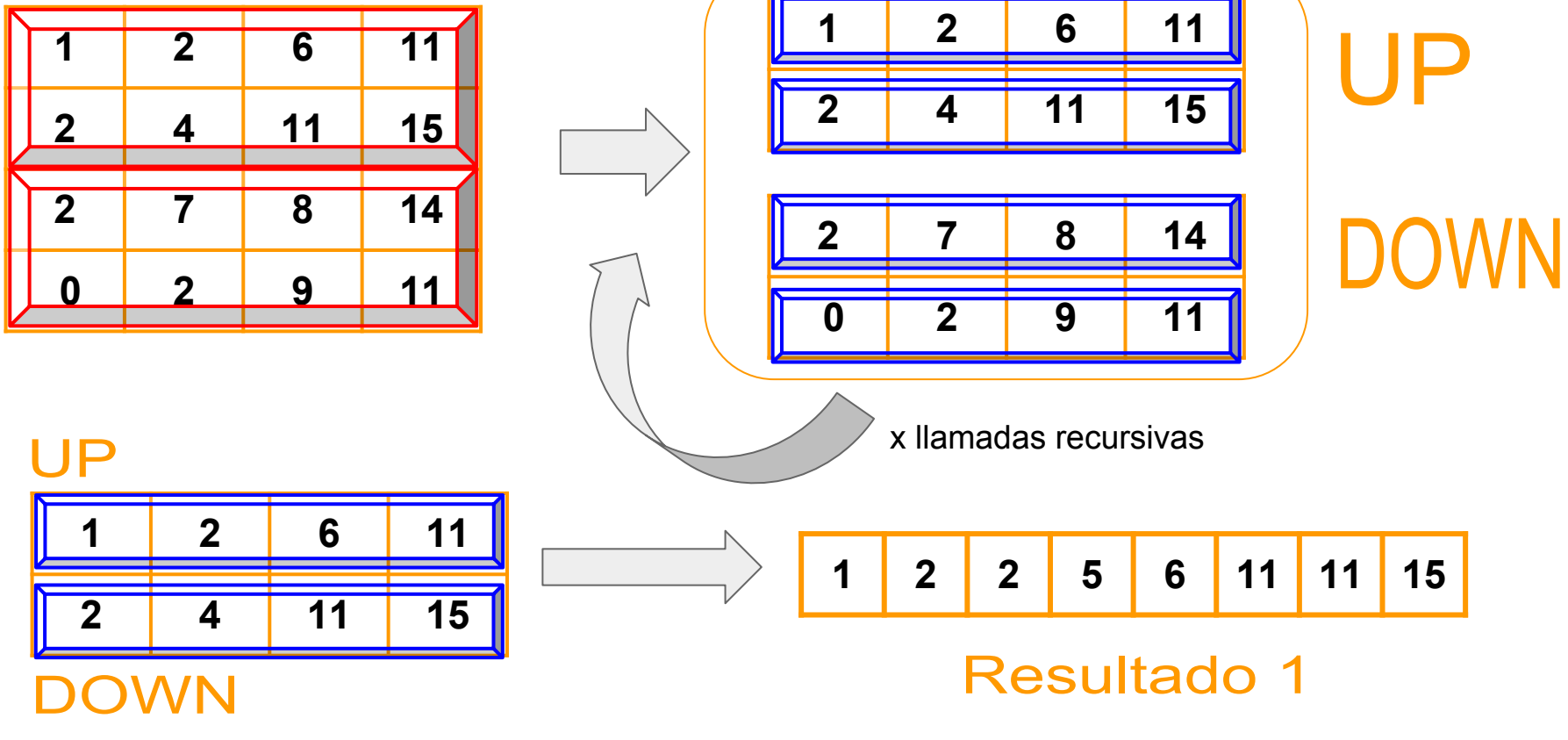
2. Resolución: Divide y Vencerás

Código C++ :

- *Dividimos la matriz de vectores por la mitad*
- *Almacenamos las mitad en dos submatrices*
- *Llamada recursiva hasta obtener un solo vector por mitad*
- *Mezclamos y ordenamos ambos vectores*
- *Mezclamos los resultantes*

```
vector<vector<int>> dyv(vector<vector<int>> &vectores){

    // Caso base, el tamaño de la matriz es 1
    if (vectores.size() <= 1)
    return vectores;

    // En otro caso:
    // Calculamos la mitad de la matriz en filas
    // Creamos las matrices de ambas mitades
    // Rellenamos las nuevas matrices con los elementos de cada mitad
            up[i][j] = vectores[i][j];
            down[i - mitad][j] = vectores[i][j];

    // Llamada recursiva que divide la matrices creadas hasta llegar al caso base (un vector)
    up = dyv(up);
    down = dyv(down);

    // Almacenamos en un vector
    vector<vector<int>>resultado;
    resultado.resize(1);

    resultado = merge(up, down); // Mezclamos y ordenamos los vectores
    return resultado;
}
```

2. Resolución: Divide y Vencerás



3. Comparativa Fuerza Bruta y Divide y Vencerás

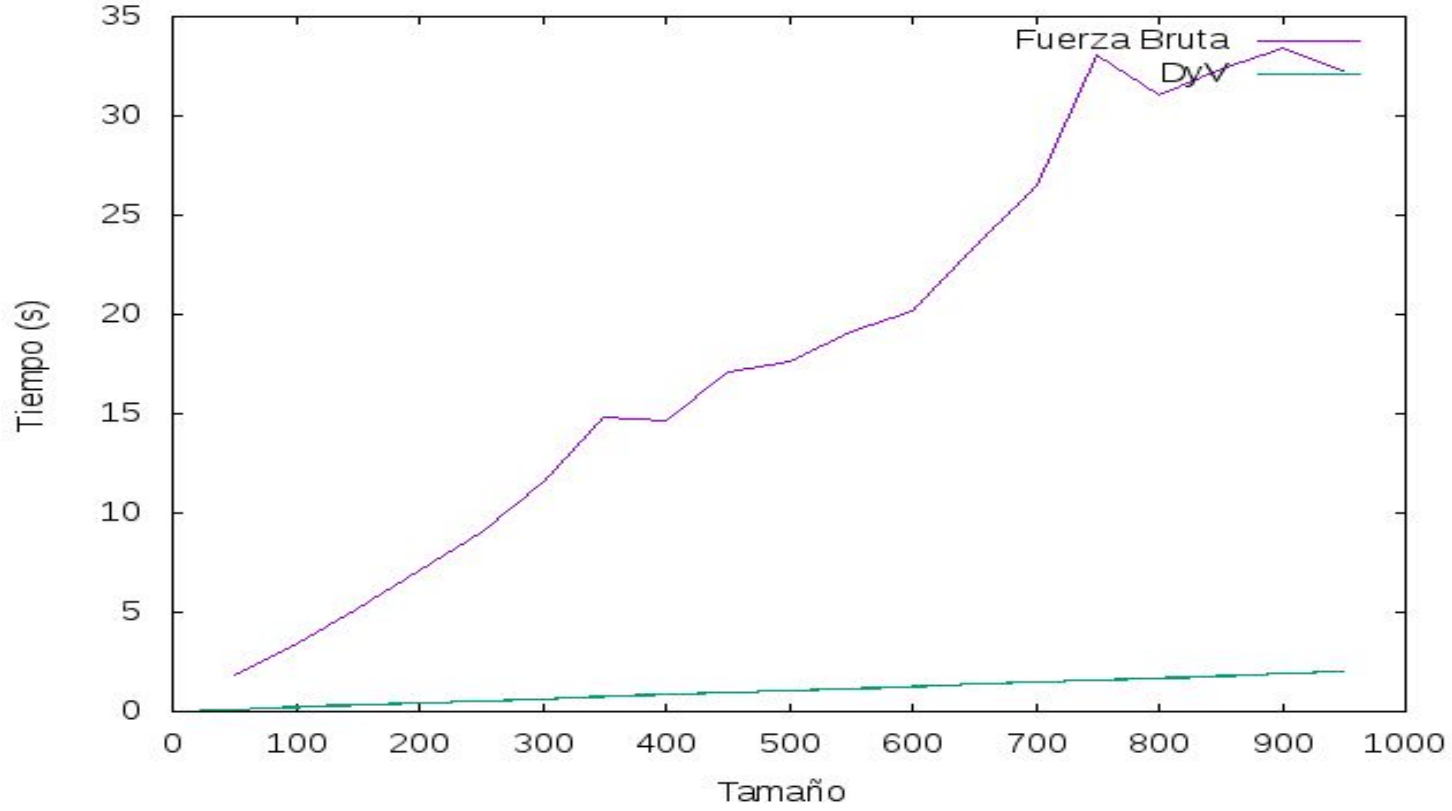
Fuerza bruta:

Tamaño 1000xN	Tiempo segundos	Tamaño Kx1000	Tiempo segundos
1000x50	0.0844241	50x1000	1.84682
1000x100	0.341674	100x1000	3.43329
1000x150	0.758249	150x1000	5.13446
1000x200	1.35689	200x1000	7.07285
1000x250	2.10287	250x1000	9.01873
1000x300	3.26232	300x1000	11.6132
1000x350	4.12584	350x1000	14.8343
1000x400	5.41344	400x1000	14.6106
1000x450	6.71772	450x1000	17.1195
1000x500	8.47446	500x1000	17.6491
1000x550	10.7019	550x1000	19.1016
1000x600	12.0535	600x1000	20.1846
1000x650	15.1226	650x1000	23.3878
1000x700	18.2998	700x1000	26.4876
1000x750	21.0617	750x1000	33.0731
1000x800	22.9302	800x1000	31.0557
1000x850	24.5404	850x1000	32.4107
1000x900	28.625	900x1000	33.4493
1000x950	33.1188	950x1000	32.2736

3. Comparativa Fuerza Bruta y Divide y Vencerás

Tamaño 1000xN	Tiempo segundos	Tamaño Kx1000	Tiempo segundos
1000x50	0.0612993	50x1000	0.115042
1000x100	0.141167	100x1000	0.220447
1000x150	0.229345	150x1000	0.32586
1000x200	0.331457	200x1000	0.451122
1000x250	0.427116	250x1000	0.532114
1000x300	0.524363	300x1000	0.635923
1000x350	0.662011	350x1000	0.750101
1000x400	0.730249	400x1000	0.843959
1000x450	0.926191	450x1000	0.994253
1000x500	0.94314	500x1000	1.05263
1000x550	1.09086	550x1000	1.1599
1000x600	1.17267	600x1000	1.26129
1000x650	1.28985	650x1000	1.36486
1000x700	1.49419	700x1000	1.46648
1000x750	1.54771	750x1000	1.60998
1000x800	1.63927	800x1000	1.67563
1000x850	1.82332	850x1000	1.78031
1000x900	1.9484	900x1000	1.88767
1000x950	2.00995	950x1000	1.99376
1000x1000	2.21E-06	1000x1000	4.99E-06

3. Comparativa Fuerza Bruta y Divide y Vencerás



4. Conclusiones

1. Vemos que el algoritmo divide y vencerás es mucho más rápido para cualquier tamaño, incluso para tamaños pequeños, es de orden de 2 - 3 segundos más rápido. En cuanto el tamaño crece un poco, se dispara el tiempo para el algoritmo de fuerza de bruta.
2. En términos de eficiencia, el algoritmo de fuerza bruta implementado tiene una de n2. Por otra parte tenemos el algoritmo usado de divide y vencerás, en el que conseguimos una eficiencia n*log (n). Como podemos apreciar claramente en la gráfica de comparación, el código usado para fuerza bruta, la gráfica crece de forma cuadrática, mientras que la de divide y vencerás se mantiene por debajo, n * log (n), ya que es un orden de eficiencia menor (más óptimo).