



Algoritmo divide y vencerás

Algoritmo: Depuración de registros



Algoritmo básico

```
vector<int> depurar_registros(vector<int> v){
    vector <int> salida;
    bool repetido = false;
    int elemento;

    /* Se compara un elemento del vector pasado como parámetro (vector aleatorio con elementos
    repetidos) con los elementos del vector que se está construyendo */
    for(int i = 0; i < v.size(); i++){

        repetido = false;
        elemento = v[i];

        /* Recorremos el vector de salida para comprobar si ya hemos insertado el elemento.
        En ese caso, se sale del bucle for y no lo inserta de nuevo. */
        for(int j = 0; j < salida.size() && !repetido; j++)
            //Comprobamos si en algún momento hay un elemento repetido.
            if(elemento == salida[j])
                repetido = true;

        //Si el elemento no se encuentra ya en salida, no está duplicado por lo que lo añadimos al vector de salida.
        if(!repetido)
            salida.push_back(elemento);

    }
    return salida;
}
```

Algoritmo Divide y vencerás - Caso Base

```
vector<int> depurarRegistrosRekursiva(vector<int> v_inicial, int pos_ini, int pos_fin){  
    vector<int> v_salida, v_dcha, v_izq;  
  
    //Si v_inicial es de tamaño = 1 se devuelve un vector de una sola posicion  
    if (pos_fin == pos_ini){  
        v_salida.push_back(v_inicial.at(pos_ini));  
        return v_salida;  
    }  
  
    //Si v_inicial es de tamaño = 2 se compara si esta dos posiciones son iguales  
    if ((pos_fin - pos_ini) == 1){  
        //Si son iguales se devuelve un vector de una sola posicion  
        if (v_inicial.at(pos_ini) == v_inicial.at(pos_fin))  
            v_salida.push_back(v_inicial.at(pos_ini));  
  
        //Si son distintos se devuelve un vector con las dos posiciones  
        else{  
            v_salida.push_back(v_inicial.at(pos_ini));  
            v_salida.push_back(v_inicial.at(pos_fin));  
        }  
    }  
  
    ...  
}
```

Algoritmo Divide y vencerás - Caso Recursivo

```
...
/* Si v_inicial es de tamaño > 2 se llama recursivamente al algoritmo, dividiendo el
problema en dos */
else{

    int mitad = (pos_fin - pos_ini) / 2 + pos_ini;

    //Llamamos a la funcion depurar registros con la parte izquierda de v_inicial
    v_izq = depurarRegistrosRecursiva(v_inicial, pos_ini, mitad);

    //Llamamos a la funcion depurar registros con la parte derecha de v_inicial
    v_dcha = depurarRegistrosRecursiva(v_inicial, mitad + 1, pos_fin);

    //Combinamos los dos vectores (ambos no tienen elementos repetidos)
    v_salida = combinarSoluciones(v_izq, v_dcha);

}

return v_salida;
}
```

Algoritmo Divide y vencerás - Otras funciones

```
vector<int> combinarSoluciones(vector<int> v_izq, vector<int> v_dcha){  
  
    vector<int> v_salida;  
    int inicio_bucle = 0;  
  
    //Copiamos el vector izq en el vector de salida  
    v_salida = v_izq;  
  
    /* Si el elemento final de v_izq es igual al primer elemento de v_dcha  
    hacemos que el bucle comience en el siguiente elemento */  
    if(v_izq[ v_izq.size() - 1 ] == v_dcha[0])  
        inicio_bucle = 1;  
  
    /* Añadimos al vector de salida todos los elementos de v_dcha */  
    for (int i = inicio_bucle; i < v_dcha.size(); i++)  
        v_salida.push_back(v_dcha[i]);  
  
    return v_salida;  
}
```

```
vector<int> depurarRegistros(vector<int> v){  
    quicksort(v, v.size());  
    return depurarRegistrosRecursiva(v, 0, v.size() - 1);  
}
```

Main para ambos algoritmos

```
int main(int argc, char *argv[]){

    vector<int> v_entrada, v_salida;
    int elemento;

    clock_t t_ini, t_fin;

    fstream fe;
    fe.open(argv[1]);

    while (!fe.eof()){
        fe >> elemento;
        v_entrada.push_back(elemento);
    }

    fe.close();

    int tam_entrada = v_entrada.size();

    t_ini = clock();
    v_salida = depurarRegistros(v_entrada);
    t_fin = clock();

    cout << tam_entrada << " " << ((double)(t_fin - t_ini)) / CLOCKS_PER_SEC << endl;


}
```

Cálculo eficiencia empírica - script

Para facilitar la tarea de ejecutar el algoritmo numerosas veces y poder obtener el tiempo de ejecución de cada tamaño, hacemos uso de un script:

```
#!/bin/csh
@ inicio = 1000
@ fin = 50000
@ incremento = 1000

while ( $i <= $fin )
    `./generar-duplicados $i` > vector.txt
    `./depuracionBasica vector.txt` >> depuracionBasica.dat
    `./depuracionDyV vector.txt` >> depuracionDyV.dat
    @ i += $incremento
end
```



RESULTADOS OBTENIDOS

Algoritmos básico y DyV



Cálculo eficiencia teórica algoritmo básico

```
vector<int> depurar_registros(vector<int> v){  
    vector<int> salida;  $O(1)$   
    bool repetido = false;  $O(1)$   
    int elemento;  $O(1)$   
  
    /* Se compara un elemento del vector pasado como parámetro (vector aleatori  
    o con elementos  
    repetidos) con los elementos del vector que se está construyendo */  
    for(int i = 0; i < v.size(); i++){  
  
        repetido = false;  $O(1)$   
        elemento = v[i];  $O(1)$   
  
        /* Recorremos el vector de salida para comprobar si ya hemos insertado  
        el elemento.  
        En ese caso, se sale del bucle for y no lo inserta de nuevo. */  
        for(int j = 0; j < salida.size() && !repetido; j++)  
            //Comprobamos si en algun momento hay un elemento repetido.  
            if(elemento == salida[j])  $O(1)$   $O(1)$   $O(n)$   
                repetido = true;  $O(1)$   
  
        //Si el elemento no se encuentra ya en salida, no está duplicado por lo  
        que lo añadimos al vector de salida.  
        if(!repetido)  $O(1)$   
            salida.push_back(elemento);  $O(1)$   $O(1)$   
    }  
    return salida;  
}
```

$O(n^2)$

$O(n^2)$

Cálculo eficiencia teórica algoritmo DyV

$$T(n) = \begin{cases} 1 & \text{si } n=1 \\ 2T(\frac{n}{2}) + n & \text{si } n \geq 2 \end{cases}$$

Cambio variable $\rightarrow n = 2^m \Rightarrow m = \log_2 n$

$$T(n) = T(2^m) = 2T(2^{m-1}) + 2^m$$

$$T_m = 2T_{m-1} + 2^m \Rightarrow T_m - 2T_{m-1} = \textcircled{2^m}$$

$\nearrow b=2$
 $\searrow p(n)=1$

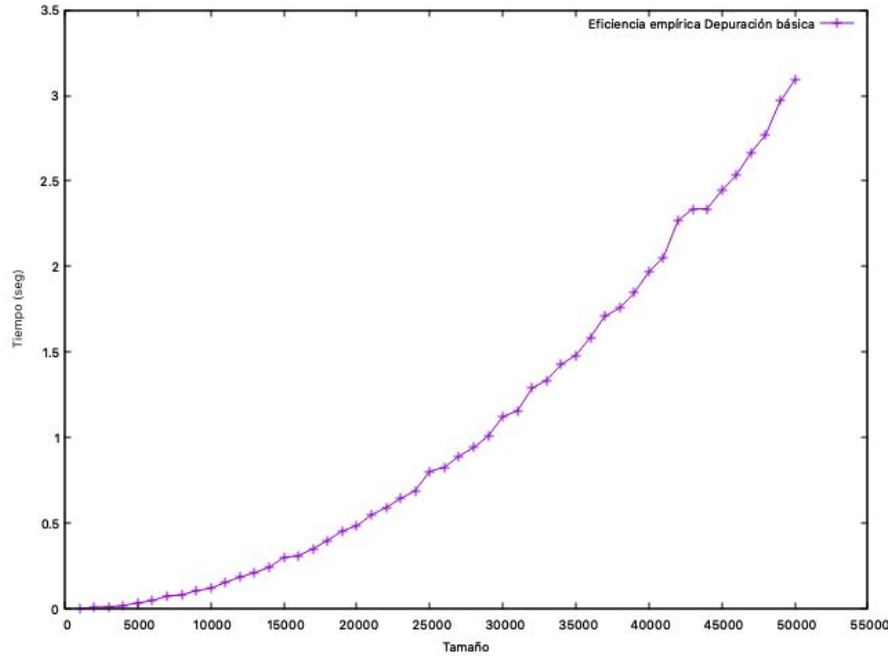
Ecuación característica: $(x-2)^2 = 0$

Solución: $T_m = (c_1 m + c_2) 2^m$

Deshacemos el cambio: $T_n = (A \log_2 n + B) 2^{\log_2 n} = (A \log_2 n + B) n = A \cdot n \log_2 n + Bn \Rightarrow O(n \log_2 n + n)$

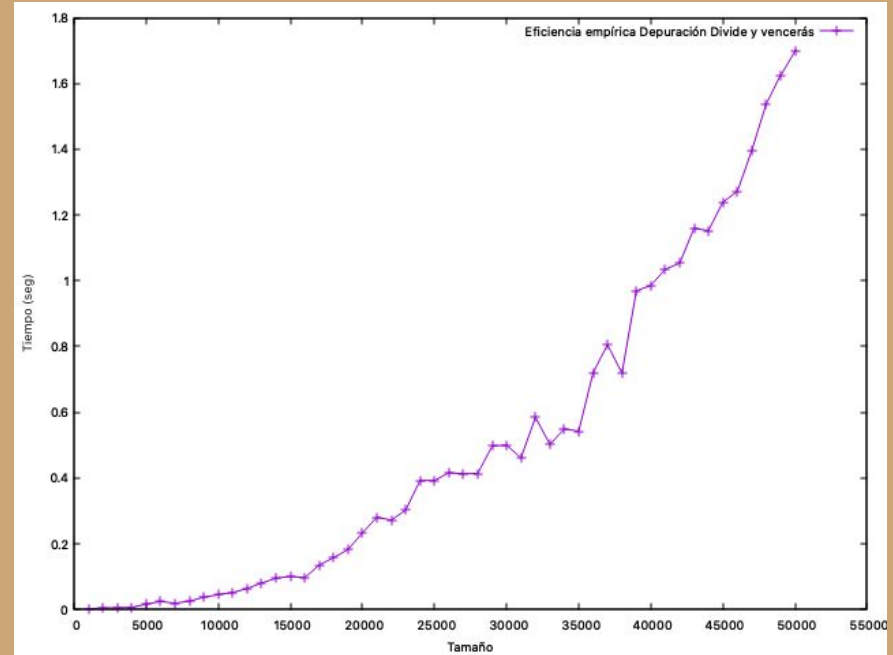
EFICIENCIA

DEPURACIÓN BÁSICA



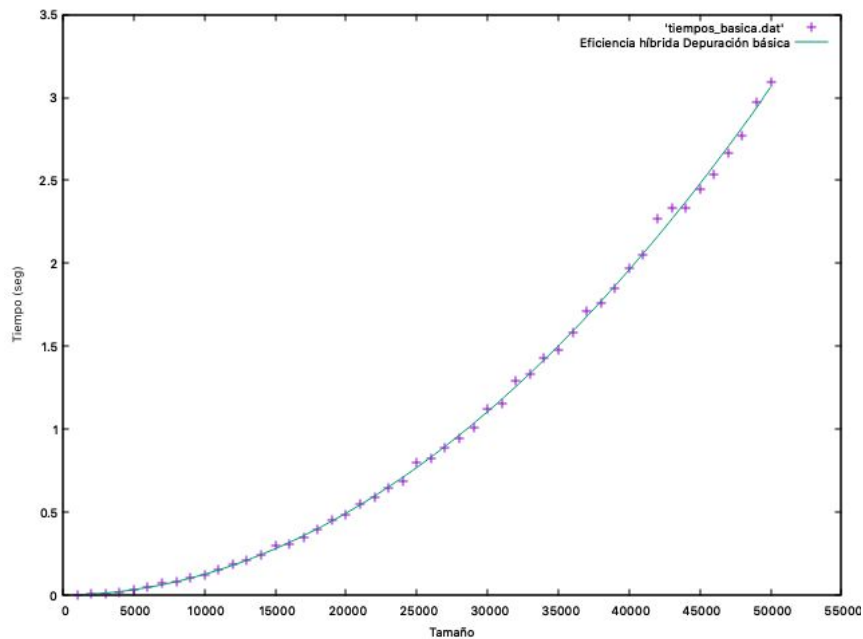
EMPÍRICA

DEPURACIÓN DIVIDE Y VENCERÁS



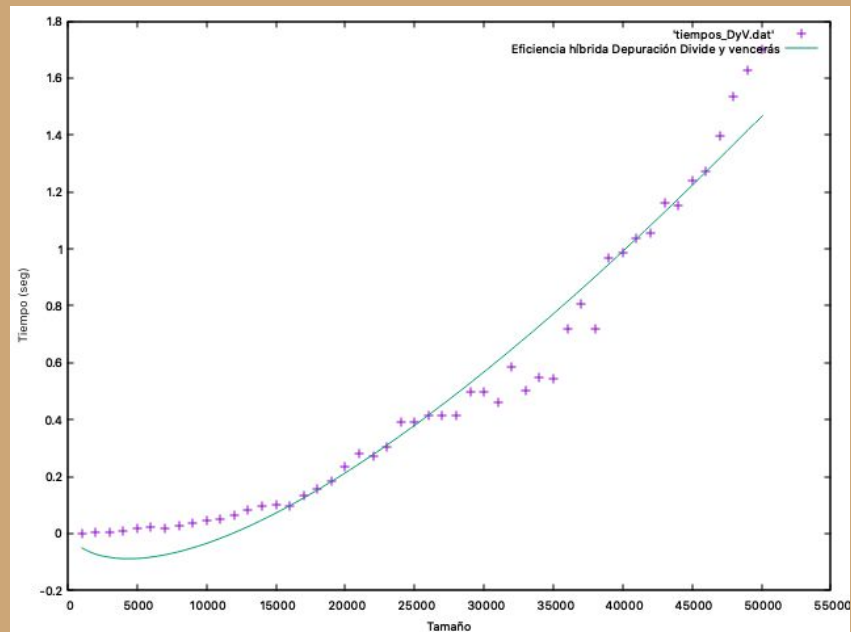
EFICIENCIA

DEPURACIÓN BÁSICA



HÍBRIDA

DEPURACIÓN DIVIDE Y VENCERÁS



Comparación ef. empírica de los algoritmos

