LECCION 6.- TÉCNICA de DISEÑO DE ALGORITHOS: DIVIDE Y VENCERÁS

DIVIDE Y VENCERAS → resuelve un problema a partir de la solución de subproblemas del mismo tipo pero de menor tamaño → (RECURSIÓN)

PASOS FUNDAMENTALES

1) Plantear el problema como K-subproblemas de menor tamaño.
Dividimos el problema en K-subproblemas cada uno de nk de forma que $0 \leq nk < n \Rightarrow$ DIVISIÓN

2) Se resuelve de manera independiente los K-subproblemas:
- De forma directa si es un caso base
- De forma recursiva.

3) Combinar las K soluciones de los subproblemas para obtener la solución al problema original

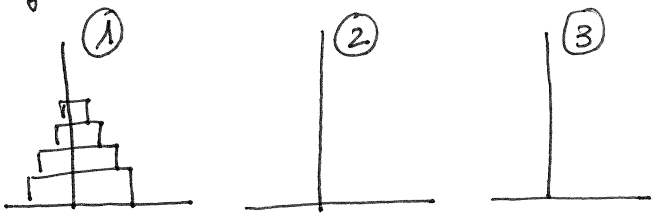
ESQUEMA GENERAL

```
Divide Venceras (p: problema)
    para i = 1,2,... K
        si ← RESOLVER(pi)
    solucion ← COMBINAR(s1, s2 ..., sk).
```

Ej1: Fibonacci
1, 1, 2, 3, 5, 8, ..., (n-1) + (n-2)

```
int Fibonacci(int n){
  if (n == 0 || n == 1)
      return 1
  else
      return Fibonacci(n-1) + Fibonacci(n-2);
}
```

Ej2: Torres de Hanoi



OBJETIVO- mover los n discos de la torre ① a la torre ③ de manera que un disco siempre tiene encima otro disco de menor diámetro.

```
void Mover (int n, int T1, int T2){
  if (n == 1)
    cout << "Moviendo de" << T1 << "a"
         << T2;
  else{
    Mover(n-1, T1, 6-T1-T2);
    Mover (1, T1, T2);
    Mover (n-1) 6-T1-T2, T2);
  }
}
```

LECCION 6.- Divide y Vencerás

ESQUEMA RECURSIVO:

Con división en 2 subproblemas y datos almacenados entre las posiciones p y q.

```
DivideVenceras(p, q : indice)
    si Pequeño(p,q) entonces
        solucion ← SolucionDirecta(p,q)
    else m ← Dividir(p,q)
         Solucion ← Combinar(DivideVenceras(p,m)
                             DivideVencerás(m+1, q))
```

Ej 1: Buscar el maximo en un vector entre las posiciones p y q.

```
const int Th = 3;
int Maximo (const int *v, int p, int q){
    if (q-p < Th) {
        int maximo = v[p];
        for (int i=p+1; i<=q; i++)
            if (max < v[i])
                max = v[i];
        return max;
    }
    else
        int m = (p+q)/2;
        int d1 = Maximo(v, p, m);
        int d2 = Maximo(v, m, q);
        return (d1>d2 ? d1 : d2);
}
```

REQUISITOS PARA APLICAR DyV

1) Tener un método de resolver los problemas de tamaño pequeño

2) División del problema original en un conjunto de subproblemas con soluciones más sencillas.

3) Problemas deben ser disjuntos

4) Método para combinar los resultados.

Si solamente tenemos un subproblem
↳ Hablamos de reducción.

Ejemplo

```
int BusquedaBinaria (int *v,
                     int inicio, int fin, int x)
{
    if (inicio <= fin){
        int mitad = (fin+inicio)/2
        if (v[mitad] == x)
            return mitad
        else if (x > v[mitad])
            return BusquedaBinaria(v, mitad+1,
                                   fin, x);
        else return BusquedaBinario(v, inicio,
                                    mitad-1, x);
    }
    else return -1;
}
```

$T(n) = T\left(\frac{n}{2}\right) + 1 \qquad n \geq 1$

$n = 2^m$

$T(2^m) - T(2^{m-1}) = 1 \quad \begin{cases} b=1 \\ p(m)=1 \\ d=0 \end{cases}$

$x = T(2^m)$

$(x-1)(x-1) = 0$

$T(2^m) = c_1 \cdot 1^m + c_2 \, m \, 1^m$

$T(n) = c_1 + c_2 \log n \in \Theta(\log n).$

LECCION 6: DyV

Ejemplo: Diseñar un algoritmo de busqueda Temaria en un vector.
La idea es partir el vector en 3 partes y activar recursivamente la funcion de busqueda en cada un de las partes

Solucion 1

```
bool B_Temaria(const int *v, int n, int x)
{
    if (n==0)
        return false

    else if (n==1)
        return v[0]==x;

    else{
        int mitad = n/2;
        if (v[mitad]==x)
            return true
        int tercio = n/3;
        if( B_Temaria(v, tercio, x))
            return true;
        else
            if(B_Temaria(v+tercio, tercio, x))
                return true;

            else
                if (B_temaria(v+2tercio, n-tercio, x))
                    return true;

                else
                    return false;
    }
}
```

$$T(n) = \begin{cases} 1 & n==0 \ || \ n==1 \\ 3T(\frac{n}{3}) + 1 & n \geq 2 \end{cases}$$

$T(n) = 3T(\frac{n}{3}) + 1 \qquad n = 3^m$

$T(3^m) = 3T(3^{m-1}) + 1 \qquad \begin{cases} b=1 \\ p(m)=1 \\ d=0 \end{cases}$

$x = T(3^m)$

$(x-3)(x-1)=0$

$T(3^m) = c_1 \cdot 3^m + c_2 1^m$

$T(n) = c_1 n + c_2 \in O(n)$

SOLUCION 2

```
bool B_Temaria(const int *v, int n, int x
{
    if (n==0)
        return false

    else
        if (n==1)
            return v[0]==x;

        else{
            int tercio = n/3;
            if (v[tercio]==x)
                return true;
            else if (v[2*tercio ==x)
                return true;

            if( v[tercio] > x)
            return B_Temaria(v, tercio, x);
            if (v[2*tercio < x)
                return B_Temaria(v+2*tercio+1,
                                 n-2*tercio-1, x)

            return B_Temaria(v+tercio,
                             tercio, x)
        }
}
```

$$T(n) = \begin{cases} 1 & n==0 \ || \ n==1 \\ T(\frac{n}{3}) + 1 & n \geq 2 \end{cases}$$

$T(n) = T(\frac{n}{3}) + 1 \qquad n = 3^m$

$T(3^m) = T(3^{m-1}) + 1 \qquad \begin{cases} b=1 \\ p(m)=1 \\ d=0 \end{cases}$

$(x-1)^2 = 0$

$T(3^m) = c_1 1^m + c_2 m \cdot 1^m$

$T(n) = c_1 + c_2 \log_3(n) \in O(\log(n))$

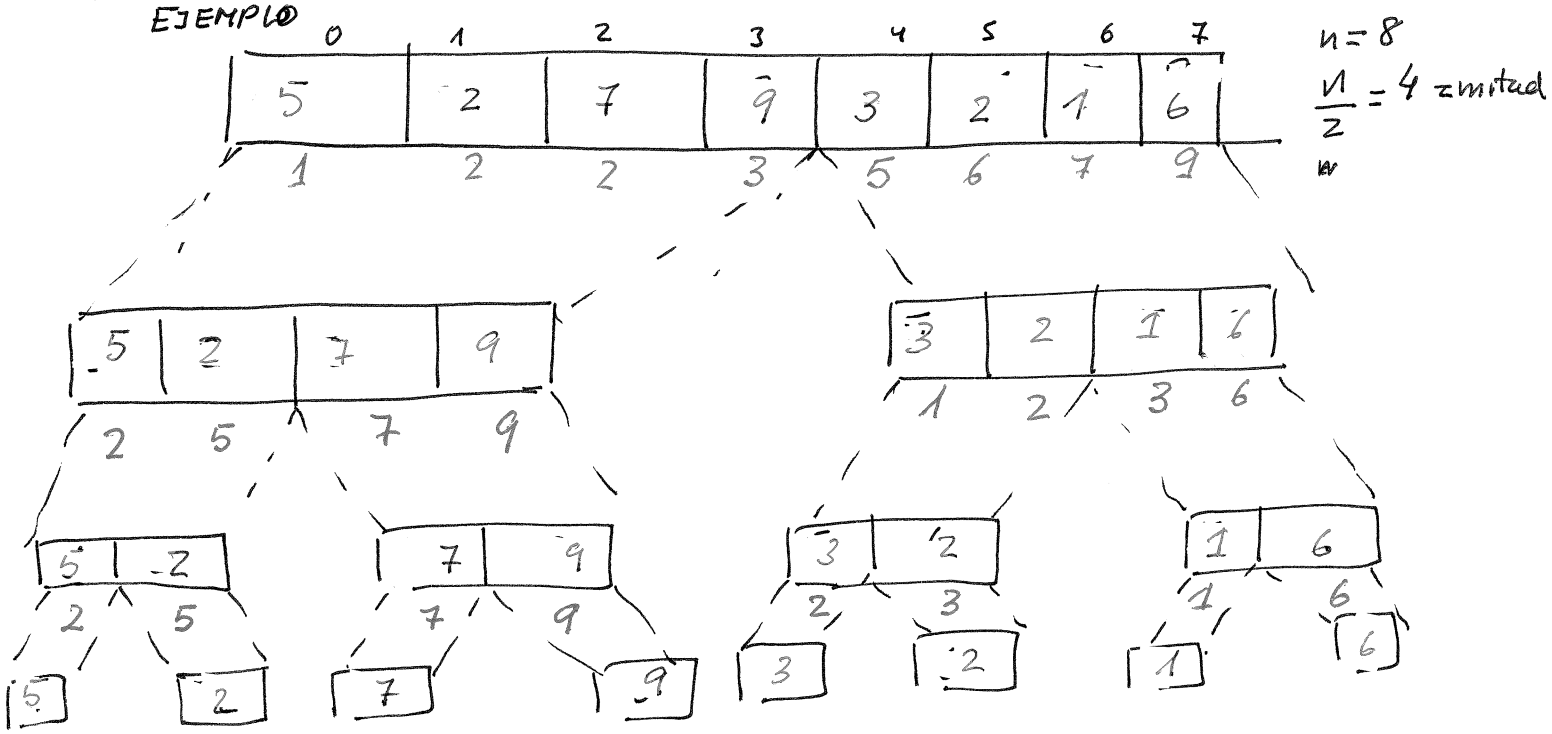
LECCION 6: DyV

Ordenación por mezcla: MERGESORT

OBJETIVO: Realizar la ordenación de un vector mediante DyV.

- FUNCIONES GENÉRICAS
  - DIVIDIR: El vector se divide en dos subvectores de igual tamaño de $\lceil \frac{n}{2} \rceil$ y $\lfloor \frac{n}{2} \rfloor$ elementos
  - RESOLVER recursivamente los dos subproblemas
  - PEQUEÑO: Si tenemos un solo elemento ya está ordenado
  - COMBINAR: Combinar los dos subvectores ordenados en el vector original de forma ordenada.

EJEMPLO



LECCION 6.- DyV

MergeSort (continuación)

```
void Mergesort(int *v, int inicio, int fin){
    if (fin-inicio < Th)
        Ordenacion_clasica(v, inicio, fin);
    else{
        int mitad = (fin+inicio)/2;
        MergeSort(v, inicio, mitad)
        MergeSort(v, mitad+1, fin)
        Combinar(v, inicio, mitad, fin)
    }
}
void Combinar(int *v, int inicio, int mitad, int fin){
    int *aux = new int[fin-inicio+1];
    int izq = inicio; int drch = mitad+1;
    int pos = 0;
    while(izq ≤ mitad && drch ≤ fin){
        if(v[izq] < v[drch]){
            aux[pos] = v[izq];
            izq++;
        }
        else{
            aux[pos] = v[drch];
            drch++;
        }
        pos++;
    }
    while(izq ≤ mitad){
        aux[pos] = v[izq];
        izq++; pos++;
    }
    while(drch ≤ fin){
        aux[pos] = v[drch];
        pos++; drch++;
    }
    for(int i=0; i < pos; i++)
        v[i+inicio] = aux[i];
    delete []aux;
}
```

LECCION 6.- DyV

Ordenación QUICKSORT

- Funciones genéricas

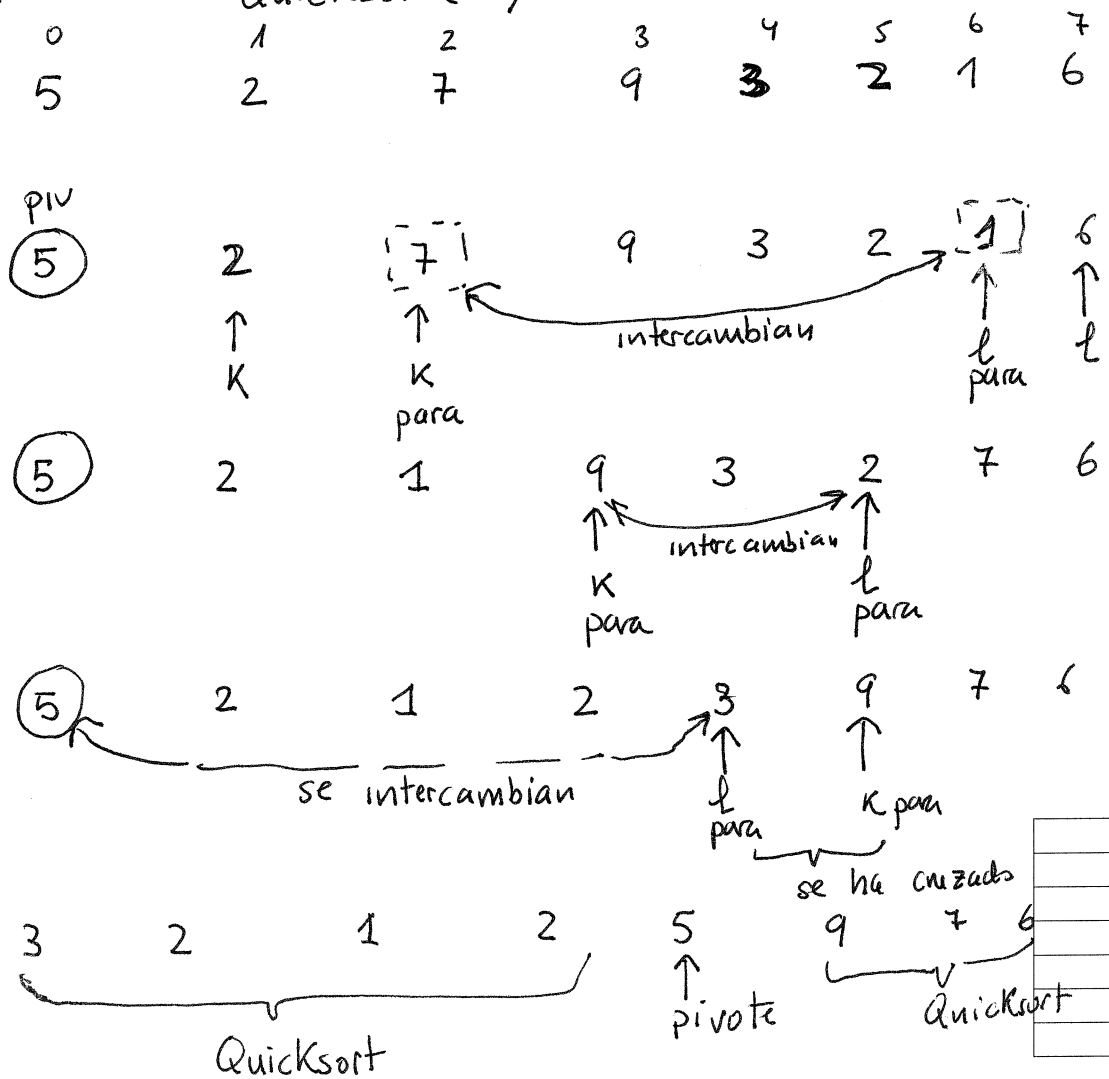
. DIVIDIR.- el vector se divide usando un procedimiento Pivote que devuelve un entero l entre (i,j) tal que

$$v[ia] \leq v[l] < v[ja] \qquad ia = 1 \ldots l-1 \quad ja = l+1 \ldots j$$

. ORDENAR.- Ordena recursivamente los trozos $(i \ldots l-1)$ y $(l+1 \ldots j)$

. COMBINAR.- No es necesario realizar ninguna combinación.

Ej.- Quicksort(v, 0, n-1)



LECCION 6.- Dy V

```
void Quicksort(int *v, int i, int j){
    if (i<j){
        int p = Pivota(v, i, j)
        Quicksort(v, i, p-1)
        Quicksort(v, p+1, j)

    }
}
int Pivota(int *v, int i, int j){
    int piv, k, l;
    k = i;
    piv = v[i];
    l = j+1;
    do{
        k = k+1;
    } while (v[k] <= piv && k<j);
    do{
        l = l-1;
    } while (v[l] > piv);
    while (k<l){
        Intercambiar(v[k], v[l]);
        do{
            k = k+1;
        } while (v[k] <= piv);
        do{
            l = l-1;
        } while (v[l] > piv;
    }
    Intercambiar(v[i], v[l]);
    return l;
}
```