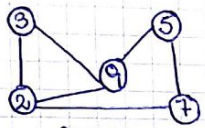
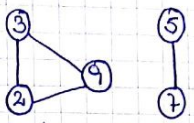
GRAFOS

GRAFO DIRIGIDO, LA RELACIÓN SOBRE V NO ES SIMÉTRICA. ARCOS ≡ PAR ORDENADO (u, v)
GRAFO NO DIRIGIDO, LA RELACIÓN SOBRE V ES SIMÉTRICA. ARISTAS ≡ PAR NO ORDENADO $\{u, v\}$, $u, v \in V$ y $u \neq v$.

UN GRAFO NO DIRIGIDO ES CONEXO SI HAY UN CAMINO ENTRE CADA PAR DE VÉRTICES.

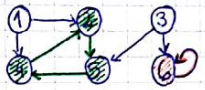


CONEXO



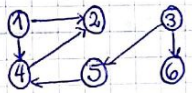
NO CONEXO

CICLO: EJ: $<2,5,4,2>$ ES UN CICLO DE LONGITUD 3
EL CICLO ES SIMPLE SI EL CAMINO ES SIMPLE



BUCLE: CICLO DE LONGITUD 1.

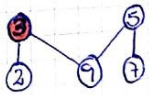
GRAFO ACÍCLICO: GRAFO SIN CICLOS.



UN **BOSQUE**, ES UN GRAFO SIN CICLOS

UN **ÁRBOL LIBRE** ES UN BOSQUE CONEXO

UN **ÁRBOL**, ES UN ÁRBOL LIBRE EN EL QUE UN NODO SE HA DESIGNADO COMO RAÍZ

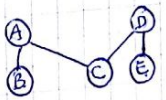


SEA UN GRAFO G NO DIRIGIDO CON m VÉRTICES Y m ARCOS ¿ENTONCES

$$\sum_{v \in G} \text{grado}(v) = 2 * m$$

$$8 = 2 * 4$$

$$8 = 8 \checkmark$$



$g(a) = 2$
$g(b) = 1$
$g(c) = 2$
$g(d) = 2$
$g(e) = 1$
8

Siempre: $m \leq (n*(n-1))/2$

Si G es conexo: $m \geq n-1$
Si G ÁRBOL: $m = n-1$
Si G BOSQUE: $m \leq n-1$

REPRESENTACIONES: MATRIZ DE ADYACENCIAS

$G=(V,E)$: MATRIZ A DE DIMENSIÓN $|V| \times |V|$
COSTO ESPACIAL: $\Theta(|V|^2)$

REPRESENTACIONES: LISTA DE ADYACENCIAS

$G=(V,E)$: VECTOR DE TAMAÑO $|V|$.

Si G es DIRIGIDO, la suma de las longitudes de las listas de adyacencias será $|E|$
Si G es NO DIRIGIDO, la suma de las longitudes de las listas de adyacencias será $2|E|$.
Costo espacial sea dirigido o no: $\Theta(|V|+|E|)$

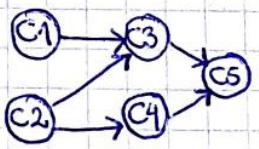
TIEMPO DEL RECORRIDO EN PROFUNDIDAD ES $\Theta(|V|+|E|)$

TIEMPO DEL RECORRIDO POR NIVELES ES $\Theta(|V|+|E|)$
COSTO $T(|V|,|E|)$

ORDENACIÓN TOPOLÓGICA

Solo se puede aplicar a grafos acíclicos

1. Se realiza recorrido DFS
2. Por cada vértice q' se termina de recorrer se apila.
3. Una vez q' terminamos todo desapilamos y ese es el resultado de la ordenación topológica.



ordenación topológica:
C2 - C4 - C1 - C3 - C5

Pila

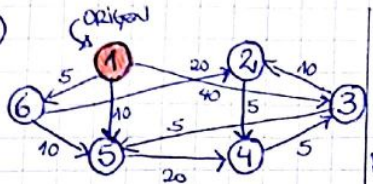
C2
C4
C1
C3
C5

COSTO TOTAL DEL ALGORITMO ES $O(|V|+|E|)$

ALGORITMO DIJKSTRA $(\Theta(E \log V))$

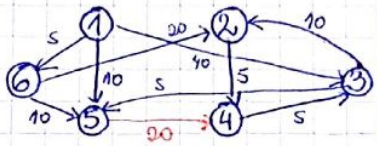
D_v : DISTANCIA MÍNIMA DESDE EL ORIGEN (INICIALMENTE ∞ PARA TODOS LOS VÉRTICES EXCEPTO EL ORIGEN CON VALOR 0)
P_v : VÉRTICE POR DONDE PASÓ PARA LLEGAR.
CONOCIDO : DATO BOOLEANO QUE INDICA SI YA ESTÁ PROCESADO (INICIALMENTE TODOS EN 0).

PASO 1

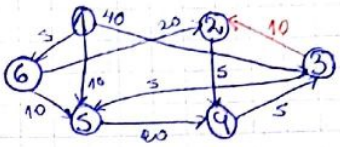


V	D_v	P_v	conoc.
1	0	0	0
2	∞	0	0
3	∞	0	0
4	∞	0	0
5	∞	0	0
6	∞	0	0

PASO 4



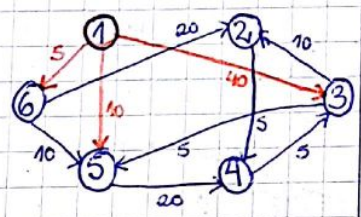
V	D_v	P_v	conoc.
1	0	0	1
2	25	6	0
3	40	1	0
4	30	5	0
5	10	1	1
6	5	1	1



V	D_v	P_v	conoc
1	0	0	1
2	25	6	1
3	35	4	1
4	30	5	1
5	10	1	1
6	5	1	1

* NO SE MODIFICA P_v A 3 PORQUE SUPERA AL D_v.

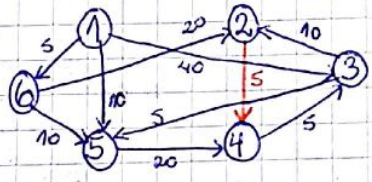
PASO 2



V	D_v	P_v	conoc.
1	0	0	1
2	∞	0	0
3	40	1	0
4	∞	0	0
5	10	1	0
6	5	1	0

PROX →

PASO 5

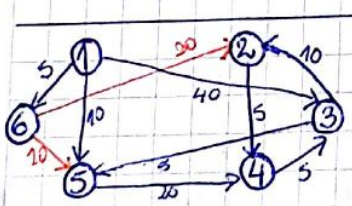


V	D_v	P_v	conoc.
1	0	0	1
2	25	6	1
3	40	1	0
4	30	5	0
5	10	1	1
6	5	1	1

PROX →

* NO SE MODIFICA P_v A 2 PORQUE D_v ES 30 Y QUEDA =

PASO 3

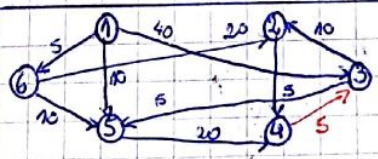


V	D_v	P_v	conoc
1	0	0	1
2	25	6	0
3	40	1	0
4	∞	0	0
5	10	1	0
6	5	1	1

PROX →

* NO SE MODIFICA P_v A 6 PORQUE D_v ES 10 Y QUEDA =

PASO 6



V	D_v	P_v	conoc.
1	0	0	1
2	25	6	1
3	35	4	0
4	30	5	1
5	10	1	1
6	5	1	1

PROX →

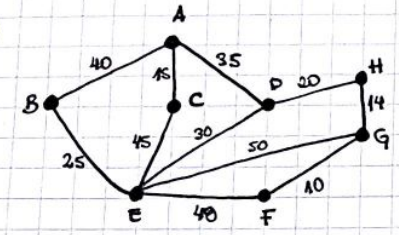
GRAFOS	BFS $O(V+E)$	DIJKSTRA $O((E+V)\log V)$	Algoritmo mas rapido en cualquier caso $O(V+E)$	Optimización de Dijkstra (sort top) $O(V+E)$
No Pesados	ÓPTIMO	CORRECTO	malo	Incorrecto si tiene ciclos
Pesados	INCORRECTO	ÓPTIMO	malo	Incorrecto si tiene ciclos
Pesos Negativos	INCORRECTO	INCORRECTO	ÓPTIMO	Incorrecto si tiene ciclos
Grafos Pesados Acíclicos	INCORRECTO	CORRECTO	malo	ÓPTIMO

CORRECTO → Aceptado pero no es el mejor.
MALO → Una solución muy lenta.

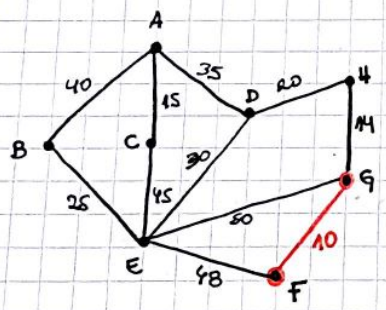
ALGORITMO DE KRUSKAL

1. Seleccionar arco de menor longitud
2. En cada iteración agregar el siguiente arco de menor longitud del conjunto de arcos disponibles
3. El algoritmo finalizará cuando todos los arcos estén conectados. (Si N = número de nodos entonces la solución es óptima debe incluir N-1 arcos).

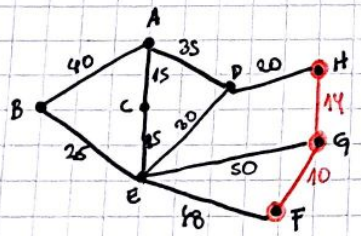
(1)



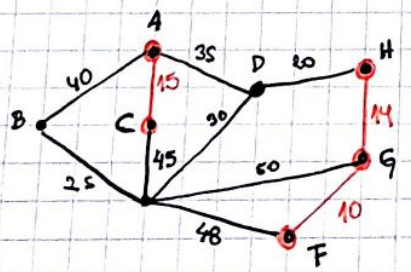
(2)



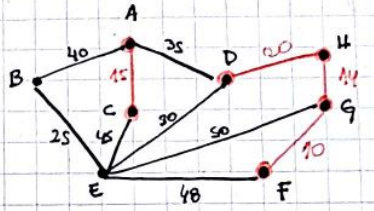
(3)



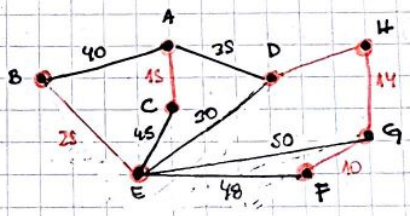
(4)



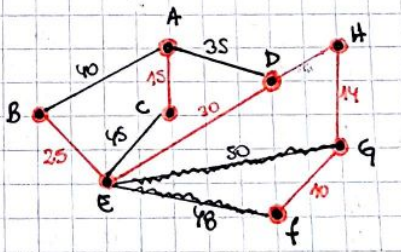
(5)



(6)

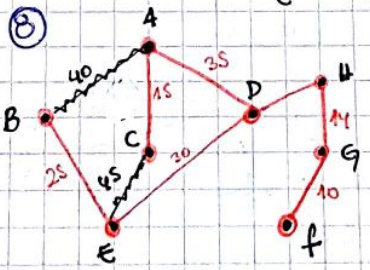


(7)



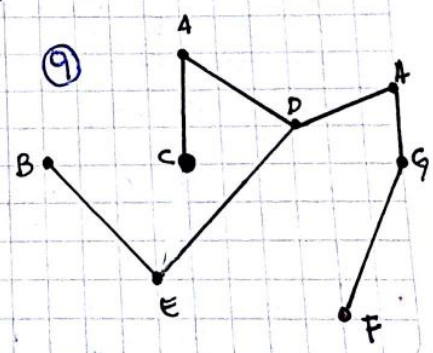
Eliminamos G,E y F,E porq' hacen formar ciclo

(8)



Eliminamos B,A y G,E porq' no se forme ciclo

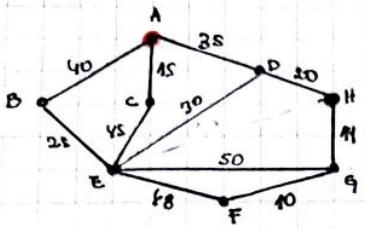
(9)

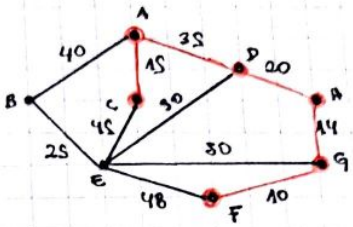


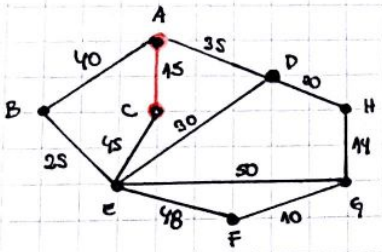
FINAL

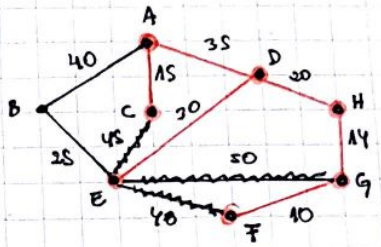
ALGORITMO DE PRIM

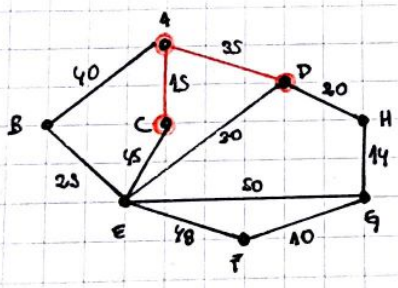
1. Se selecciona un comienzo.
2. Se selecciona el camino a un vértice no incluido con menor peso.

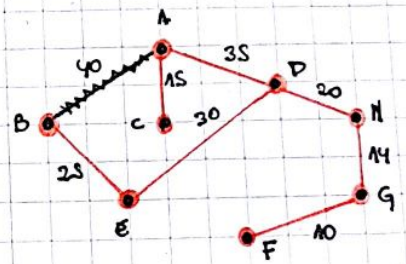


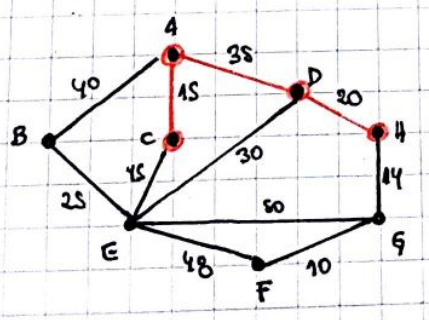


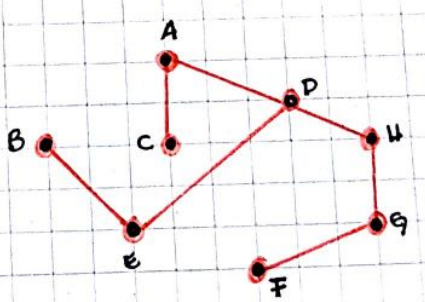


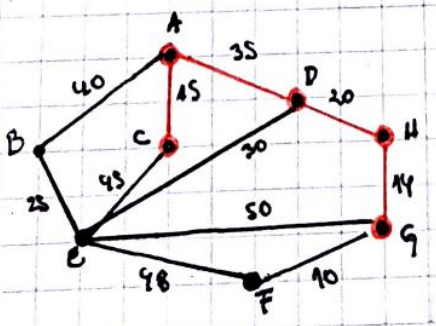












GRAFOS – RECORRIDOS : DFS (con opción lista q' devuelve los vértices recorridos)

```
Public class Recorridos <T> {

    Public ListaEnlazadaGenerica <Vertices<T>> dfs (Grafo <T> grafo) {
        Boolean [ ] marcas = New Boolean [grafo.listaDeVertices( ).tamanio()] ;
        ListaEnlazadaGenerica <Vertices <T> > lis = New listaEnlazadaGenerica <Vertices <T>>;
        For ( int i = 0 ; i < grafo.listaDeVertices() . tamanio () ; i++ ) {
            If (!marcas [i]) {
                This.dfs (i, grafo, lis , marcas) ;
        }
    -> Return lis;

    Private void dfs ( int i, Grafo<T> grafo , listaEnlazadaGenerica <Vertices<T>> lis, Boolean[ ] marcas) {
        Marcas [i] = true ;
        Vertice <T> v = grafo . listaDeVertices ( ). elemento (i) ;
    -> lis . Agregar (v, lis. tamanio()) ;
        listaGenerica <Aristas <T>> ady = grafo . listaAdyacentes (v) ;
        ady . comenzar ( ) .
        While (! ady . Fin()) {
            int j = ady. proximo() . getVerticeDestino(). GetPosición ( ) ;
            If (! marcas [j] )
                This.dfs ( j , grafo , lis , marcas) ;
        }
    }
}
```

+BFS

```
Public void BFS (Vertice <T> v, Boolean [ ] marcas)
    Cola Generica <Vertice <T>> C = New ColaGenerica <Vertice <T>> ;
    C . encolar (v) ,
    While ( ! C . Vacia ( ) ) {
        Vertice <T> w = C . Desencolar ( ) ;
        listaGenerica <Aristas <T>> ady = w . obtenerAdyacentes ( ) ,
        ady . comenzar ( ) .
        While (! ady . Fin ( ) ) {
            Aristas <T> a = ady . elemento ( ) ;
            If (! marcas [a . getVerticeDestino ( ) . posición ( ) )
                C . encolar ( a . getVerticeDestino ( ) ) ;
        }
    }
```

PRÁCTICA — Tiempos de Ejecución

1) Digo q' $T_1(n) < T_2(n)$ si existen constantes $c > 0$ y n_0 tal q':

$$T_1(n) < T_2(n) \cdot c \quad \text{con } n > n_0$$

entonces: $T_1(n) < T_2(n) \cdot 1$

$$T_1(n) < T_2(n)$$

Como $T_1(n) = 10000\,n$ y $T_2(n) = n^2$

$$10000\,n < n^2$$

$$10000 < \frac{n^2}{n}$$

$$10000 < n$$

Podemos decir q' con una cantidad de personas menor a 10000 el algoritmo 1, es mejor q' el algoritmo 2, y con una cantidad de personas q' supere este valor, comienza a ser mejor el algoritmo 2.

3) a- 3^n es de $O(2^n)$?

Decimos q' $T(n) = O(f(n))$ si existen constantes $c > 0$ y n_0 tales q'

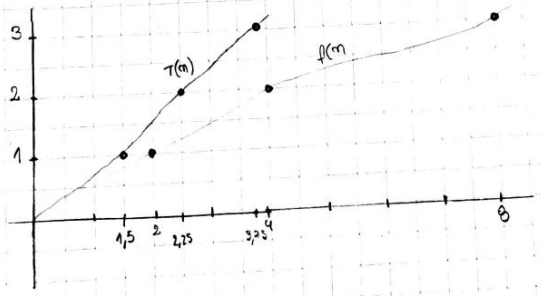
$T(n) \leq f(n) \cdot c$ con $n \geq n_0$ => Si $T(n) = 3^n$ y $f(n) = 2^n$

$$3^n \leq 2^n \cdot c$$

$$\frac{3^n}{2^n} \leq c$$

$$\left(\frac{3}{2}\right)^n \leq c \rightarrow \text{Falso}$$

La función $\left(\frac{3}{2}\right)^n$ crece de forma exponencial a medida q' n crece de manera q' no voy a poder determinar una constante q' acote la función.
La tasa de crecimiento de $f(n) = 2^n$ es menor a la tasa de crecimiento de $T(n) = \left(\frac{3}{2}\right)^n$.



⑥ $\frac{m}{\log_2(m)}$ es de $O(\log_2(m))$?

Para q' $\frac{m}{\log_2(m)} = O(\log_2(m))$ deben existir constantes $C>0$ y m_0 tal que $T(m) = f(m) \cdot C$

siendo $T(m) = \frac{m}{\log_2(m)}$ y $f(m) = \log_2(m)$ $\Rightarrow$

$$\frac{m}{\log_2(m)} \leq \log_2(m) \cdot C$$

$$\frac{m}{\log_2(m)} \cdot \frac{1}{\log_2(m)} \leq C$$

$$\frac{m}{(\log_2(m))^2} \leq C \Rightarrow$$ la función $\frac{m}{(\log_2(m))^2}$ crece así linealmente pero q' $(\log_2(m))^2$ crece mucho más lento q' m entonces no es de $O(\log_2(m))$ y no voy a poder determinar una constante q' acote esta función.

⑦ $m^{1/2} + 10^{20}$ es de $O(m^{1/2})$?

Para q' $m^{1/2}+10^{20} = O(m^{1/2})$ deben existir constantes $C>0$ y m_0 tal q' $T(m) = f(m) \cdot C$

siendo $T(m) = m^{1/2}+10^{20}$ y $f(m) = m^{1/2}$ $\Rightarrow$

$$m^{1/2}+10^{20} \leq C \cdot m^{1/2}$$

$$\frac{m^{1/2}}{m^{1/2}} + \frac{10^{20}}{m^{1/2}} \leq C$$

$\frac{10^{20}}{m^{1/2}} \to 0$ (porque $m^{1/2} \to \infty$)

$1 + 0 \leq C$

TEORÍA — Tiempos de ejecución

Notación Big-Oh

Definición:

Decimos q': $T(n) = \mathcal{O}(f(n))$ si existen constantes $C > 0$ y n_0 tales q': $T(n) \leq C \cdot f(n)$ para cada $n \geq n_0$

Regla de la suma y regla del producto:

Si $T_1(n) = \mathcal{O}(f(n))$ y $T_2(n) = \mathcal{O}(g(n)) \Rightarrow$

1. $T_1(n) + T_2(n) = \max\left(\mathcal{O}(f(n)), \mathcal{O}(g(n))\right)$
2. $T_1(n) \cdot T_2(n) = \mathcal{O}\left(f(n) \cdot g(n)\right)$

Otras reglas:

- $T(n)$ es un polinomio de grado $K \Rightarrow T(n) = \mathcal{O}(n^K)$
- $T(n) = \log^K(n) \Rightarrow \mathcal{O}(n)$ para cualquier K (n siempre crece más rápido q' cualquier potencia de $\log n$)
- $T(n) = CTE \Rightarrow \mathcal{O}(1)$
- $T(n) = CTE * f(n) \Rightarrow T(n) = \mathcal{O}(f(n))$

Ejemplos:

1- $T(n) = 3n^3 + 2n^2$ es $\mathcal{O}(n^3)$? VERDADERO.

Por definición de Big-Oh decimos q': $T(n) = \mathcal{O}(f(n))$ si $\exists$ constantes $C > 0$ y n_0 : .

$T(n) \leq C \cdot f(n) \ \forall \ n \geq n_0 \Rightarrow$

$3n^3 + 2^n \leq C \cdot n^3$

$\frac{3n^3}{n^3} + \frac{2^n}{n^3} \leq C$

$3 + \frac{2^n}{n^3} \leq C \Rightarrow n > 0 \ y \ C \geq 5$

$3 + \frac{2^1}{1^3} \leq 5$

$5 \leq 5$

② $T(m) = 3m^3 + 2m^2$ ES $O(m^4)$? VERDADERO

POR DEFINICIÓN DE BIG-Oh DECIMOS Q': $T(m) \leq O(f(m))$ SI $\exists$ CONSTANTES $C > 0$ Y m_0 ∴

$T(m) \leq C \cdot f(m) \quad \forall \; m \geq m_0 \Rightarrow$

$3m^3 + 2m \leq C \cdot m^4$

$\frac{3m^3}{m^4} + \frac{2m}{m^4} \leq C$

$\frac{3}{m} + \frac{2}{m^3} \leq C \quad \Rightarrow m > 0 \;\; Y \;\; C \geq 5$

$\frac{3}{1} + \frac{2}{1^3} \leq 5$

$3 + 5 \leq 5$

$5 \leq 5$

③ $T(m) = 1000$ ES $O(1)$? VERDADERO

POR DEFINICIÓN DE BIG-Oh DECIMOS Q': $T(m) \leq O(f(m))$ SI $\exists$ CONSTANTES $C > 0$ Y m_0 ∴

$T(m) \leq C \cdot f(m) \quad \forall \; m \geq m_0 \Rightarrow$

$1000 \leq C \cdot 1$

$\frac{1000}{1} \leq C$

$1000 \leq C$

$T(m)$ ES UNA CONSTANTE Y EL O DE UNA CONSTANTE ES 1.

④ $T(m) = 3m$ ES $O(2^m)$? FALSO

POR DEFINICIÓN DE BIG-Oh DECIMOS Q': $T(m) \leq O(f(m))$ SI $\exists$ CONSTANTES $C > 0$ Y m_0 ∴ $T(m) \leq C \cdot f(m) \; \forall \; m \geq m_0 \Rightarrow$

$3m \leq C \cdot 2^m$

$\frac{3m}{2^m} \leq C$

$\frac{3m}{2^m}$ SIEMPRE VA A TENDER A SER UN Nº MÁS CHICO Y NO SE CUMPLE Q' EXISTAN CONSTANTES C MAYORES O IGUALES.

TEORÍA — TIEMPOS DE EJECUCIÓN

Cálculo del tiempo de ejecución: Algoritmos recursivos

```
/**
    Calcula el Factorial
*/
PUBLIC STATIC INT FACTORIAL (INT n) {
    IF (n==1)
        RETURN 1;
    ELSE
        RETURN n* Factorial (n-1);
}
```

Factorial (n)

$$T(n) = \begin{cases} cte_1 & n = 1 \\ T(n-1) + cte_2 & n > 1 \end{cases}$$

DESARROLLO DE LA RECURRENCIA

$T(n) = T(n-1) + cte_2 = T((n-1)-1) + cte_2) + cte_2 \Rightarrow$

$= T(n-2) + 2cte_2 = T(n-2-1) + cte_2) + 2cte_2 \Rightarrow$

$= T(n-3) + 3cte_2 = \dots$

Paso i:

$T(n) = T(n-i) + i \cdot cte_2$

La recurrencia termina cuando $n-i=1$, despejamos y queda $n-1=i$ y dejamos todo en función de n.

$T(n) = \underbrace{T(n-n-1)}_{T(1)=cte_1} + (n-1) \cdot cte_2 \Rightarrow cte_1 + n-1 \cdot cte_2 \Rightarrow O(n)$

MAXIMO EN UN ARREGLO

$$T(m) = \begin{cases} CTE_1 & m = 1 \\ 2 \cdot T(m/2) + CTE_2 & m > 1 \end{cases}$$

$T(m) = 2 \cdot T(m/2) + CTE_2 \Rightarrow 2\left[2 \cdot T\left(\frac{m}{2}/2\right) + CTE_2\right] + CTE_2 \Rightarrow$

$= 4T(m/4) + 3\,CTE_2 \Rightarrow 4\left[2 \cdot T\,\frac{m}{4}/2) + CTE_2\right] + 3\,CTE_2 \Rightarrow$

$= 8T(m/8 + 7\,CTE_2) \Rightarrow 8\left[2 \cdot T\left(\frac{m}{8}/2\right) + CTE_2\right] + 7\,CTE_2 \Rightarrow$

$= 16T(m/16) + 15\,CTE_2 = \ldots.$

Paso i:

$2^i T\left(\frac{m}{2^i}\right) + \left(2^i - 1\right) \cdot CTE_2$ EL DESARROLLO SE TERMINA CUANDO $m = 1$ ENTONCES $\frac{m}{2^i} = 1 \Rightarrow$

$\Rightarrow m = 1 \cdot 2^i \Rightarrow m = 2^i \Rightarrow \log_2 m = \log_2(2^i) \Rightarrow \boxed{\log_2 m = i}$

$2^{\log_2 m} \cdot T\left(\frac{m}{2^{\log_2 m}}\right) + \left(2^{\log_2 m} - 1\right) \cdot CTE_2 \Rightarrow m \cdot \underbrace{T\left(\frac{m}{m}\right)}_{CTE_1} + (m-1) \cdot CTE_2 \Rightarrow m \cdot CTE_1 + (m-1) \cdot CTE_2 \Rightarrow O(n)$