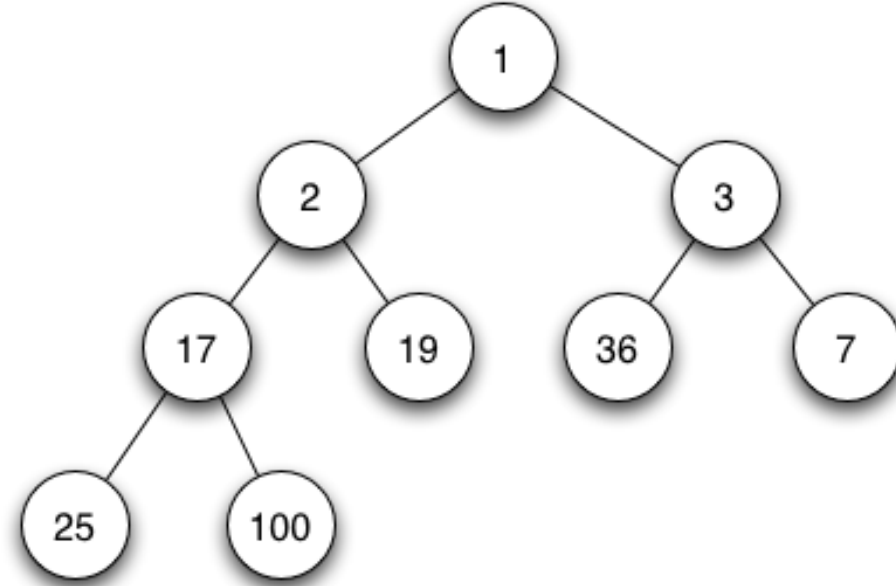


TRABAJO FINAL

COMPLEJIDAD TEMPORAL, ESTRUCTURA DE DATOS Y ALGORITMOS



Índice:

01 **Introducción**

02 **Cómo se construye la Heap**

03 **ShortestJobFirst**

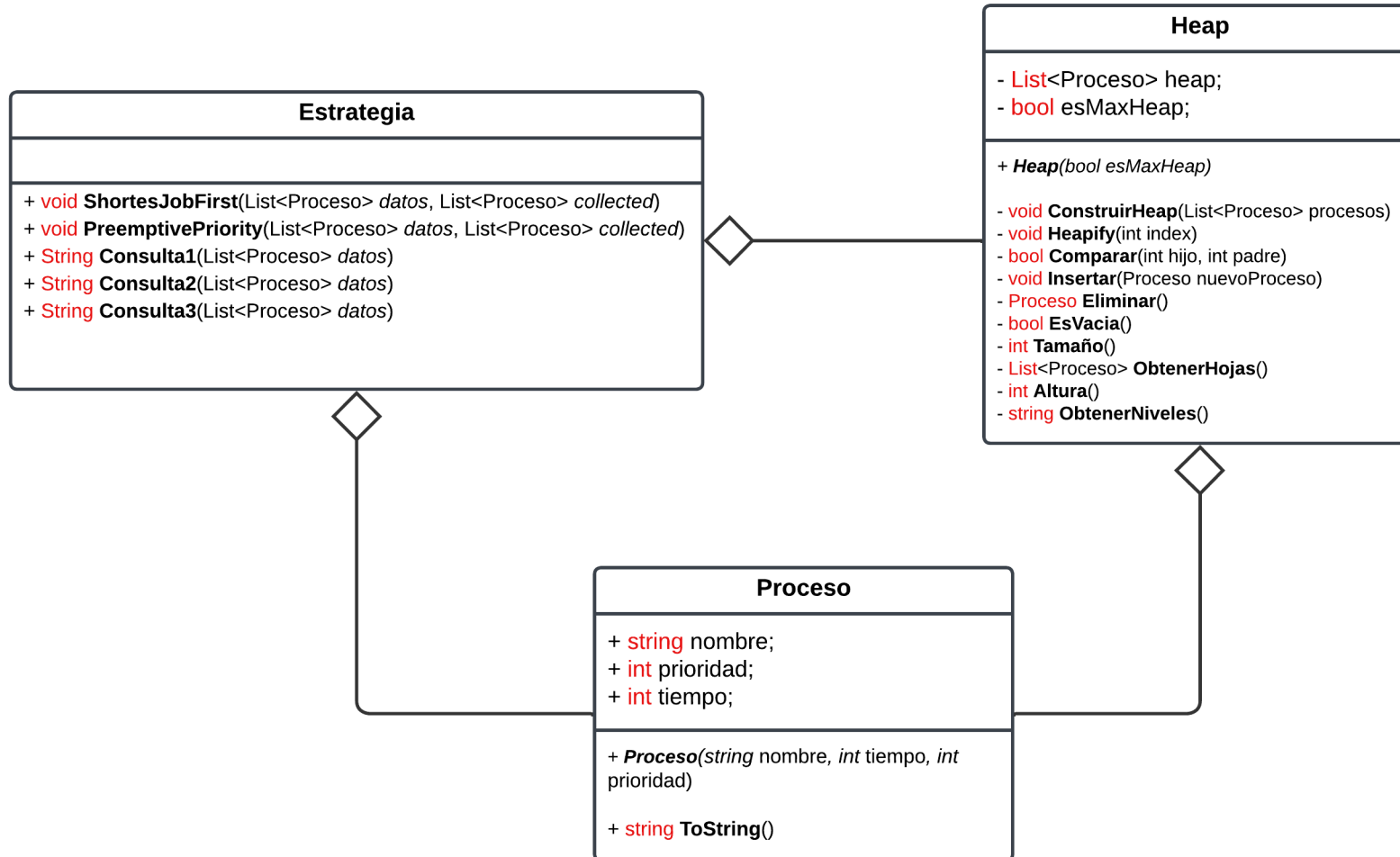
04 **PreemptivePriority**

05 **Consulta 1**

06 **Consulta 2**

07 **Consulta 3**

1. Introducción



2. Heap

```
public Heap(bool esMaxHeap) {  
    this.esMaxHeap = esMaxHeap;  
    heap = new List<Proceso> { };  
}
```

Booleano en el constructor para
definir si es MaxHeap o MinHeap



```
public void ConstruirHeap(List<Proceso> procesos) {  
    heap.AddRange(procesos);  
    for (int i = (heap.Count - 1) / 2; i >= 1; i--) {  
        Heapify(i);  
    }  
}
```

2. Heap

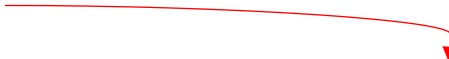
```
private void Heapify(int index) {  
    int izq = 2 * index;  
    int der = 2 * index + 1;  
    int extremo = index;  
  
    if (izq < heap.Count && Comparar(izq, extremo))  
        extremo = izq;  
  
    if (der < heap.Count && Comparar(der, extremo))  
        extremo = der;  
  
    if (extremo != index) {  
        var temp = heap[index];  
        heap[index] = heap[extremo];  
        heap[extremo] = temp;  
        Heapify(extremo);  
    }  
}
```

Considera que ese nodo es el "**extremo**" (es decir, el más pequeño en una **MinHeap** o el más grande en una **MaxHeap**).

```
private bool Comparar(int hijo, int padre) {  
    if (esMaxHeap)  
        return heap[hijo].prioridad > heap[padre].prioridad;  
  
    else  
        return heap[hijo].tiempo < heap[padre].tiempo;  
}
```

3. ShortestJobFirst

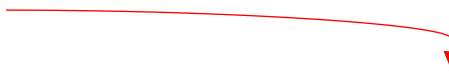
```
public void ShortesJobFirst(List<Proceso> datos, List<Proceso> collected) {  
    Heap minHeap = new Heap(false);  
    minHeap.ConstruirHeap(datos);  
  
    while (!minHeap.EsVacia()) {  
        Proceso tempo = minHeap.Eliminar();  
        collected.Add(proceso);  
    }  
}
```



```
public Proceso Eliminar() {  
    if (heap.Count <= 1) {  
        throw new InvalidOperationException("El heap está vacío.");  
    }  
  
    Proceso raiz = heap[1]; // almacenar la raíz  
    heap[1] = heap[heap.Count - 1];  
    heap.RemoveAt(heap.Count - 1);  
    Heapify(1); // restaurar la propiedad de la Heap  
    return raiz;  
}
```

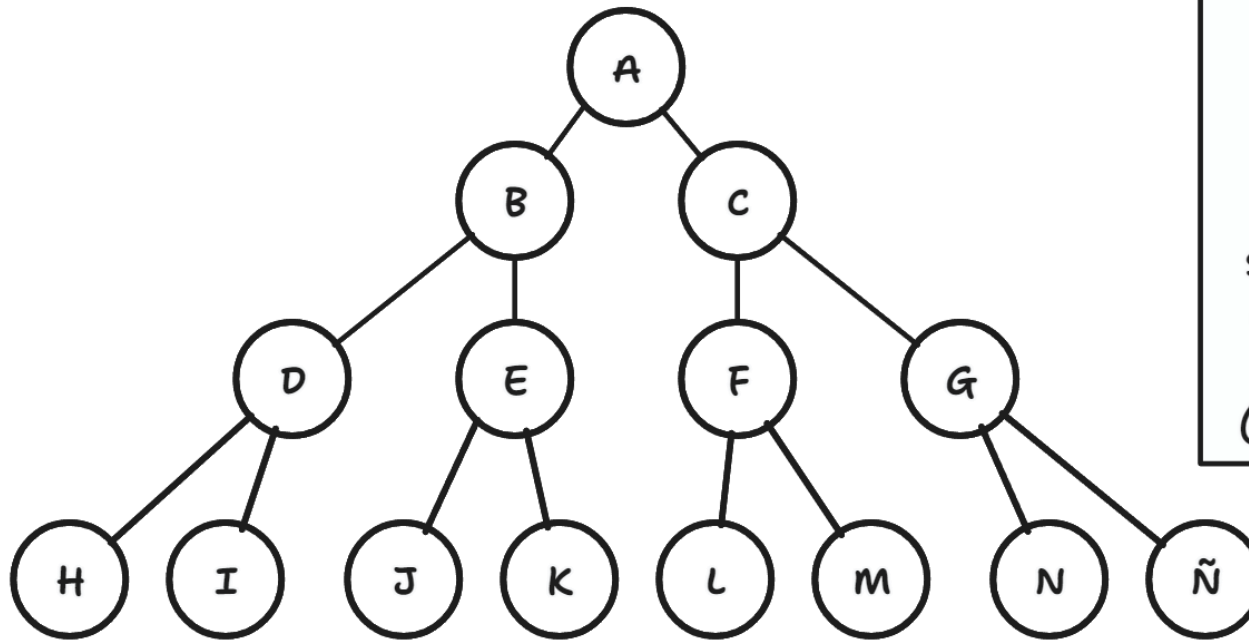
4. PreemptivePriority

```
public void PreemptivePriority(List<Proceso> datos, List<Proceso> collected) {  
    Heap maxHeap = new Heap(true);  
    maxHeap.ConstruirHeap(datos);  
  
    while (!maxHeap.EsVacia()){  
        Proceso tempo = maxHeap.Eliminar();  
        collected.Add(tempo);  
    }  
}
```



```
public Proceso Eliminar() {  
    if (heap.Count <= 1) {  
        throw new InvalidOperationException("El heap está vacío.");  
    }  
  
    Proceso raiz = heap[1]; // almacenar la raíz  
    heap[1] = heap[heap.Count - 1];  
    heap.RemoveAt(heap.Count - 1);  
    Heapify(1); // restaurar la propiedad de la Heap  
    return raiz;  
}
```

5. Consulta 1 (hojas de la Heap)



Tamaño $n = 15$
Hijo izq = $i * 2$
Hijo der = $(i * 2) + 1$
Padre = $i / 2$

Si calculamos el padre
del último elemento y
le sumamos un 1
 $(n / 2) + 1 = \text{POS } 8 \rightarrow H$

-	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

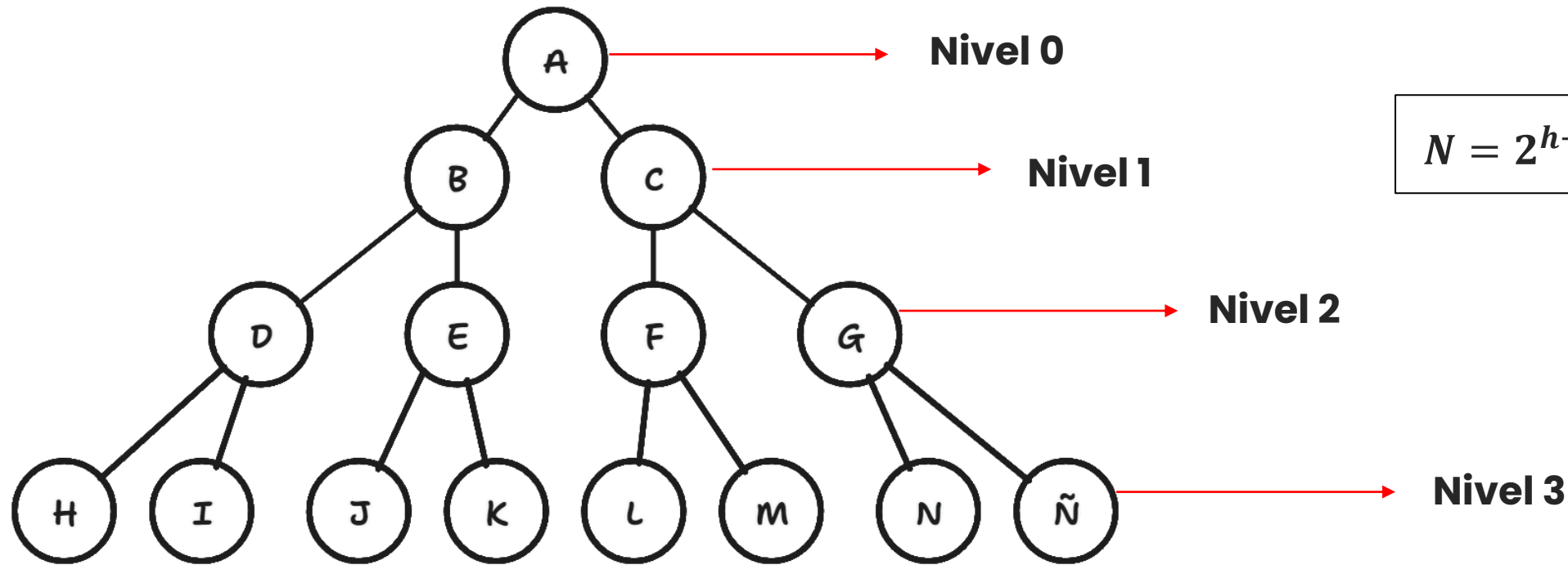
5. Consulta 1 (hojas de la Heap)

```
public List<Proceso> ObtenerHojas() {  
    List<Proceso> hojas = new List<Proceso>();  
    int n = Tamaño();  
  
    for (int i = n / 2 + 1; i <= n; i++) {  
        hojas.Add(heap[i]); // Acceder a las hojas  
    }  
    return hojas;  
}
```

**Ejemplo de uso
con minHeap:**

```
Heap minHeap = new Heap(false);  
minHeap.ConstruirHeap(datos);  
List<Proceso> hojasMinHeap = minHeap.ObtenerHojas();  
foreach (var hoja in hojasMinHeap) {  
    // Imprimir los datos deseados (nombre, tiempo, prioridad)  
}
```

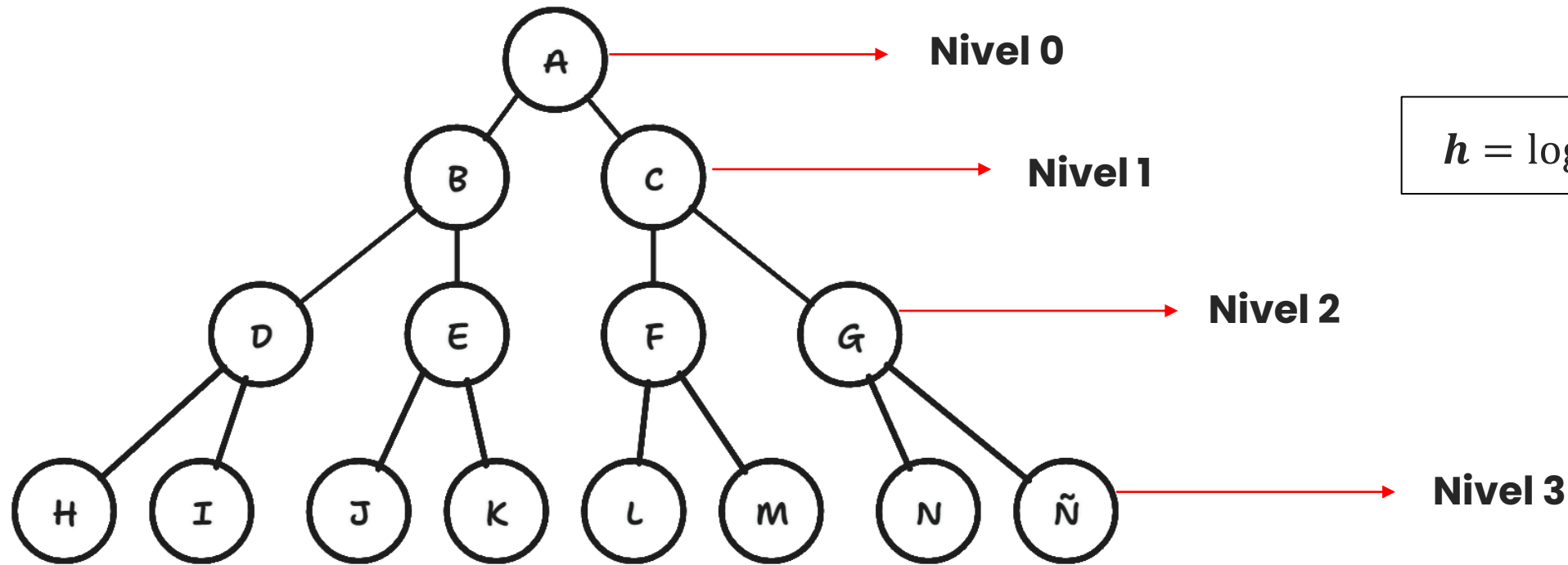
6. Consulta 2 (altura de la Heap)



$$N = 2^{h+1} - 1$$

$$N = 2^{3+1} - 1 = 15 \text{ nodos}$$

6. Consulta 2 (altura de la Heap)



$$h = \log_2(N)$$

$$h = \log_2(15) = 3$$

6. Consulta 2 (altura de la Heap)

$$h = \log_2(N)$$

$$\log_2(N) = h \rightarrow \text{Sí } 2^h = N$$



Si n = 16

$$2^4 = 16 \rightarrow \log_2(16) = 4$$

¿Cómo lo implemento a nivel código?

- ☐ Usando `Math.log2(Double)` y luego redondeando con un `Math.Floor()`.
- ☐ Divisiones sucesivas y un Contador.

6. Consulta 2 (altura de la Heap)

□ Divisiones sucesivas y un Contador.

Padre = $i / 2$

Contador = 0 y n = 16

$16 / 2 = 8 \rightarrow$ Contador = 1

$8 / 2 = 4 \rightarrow$ Contador = 2

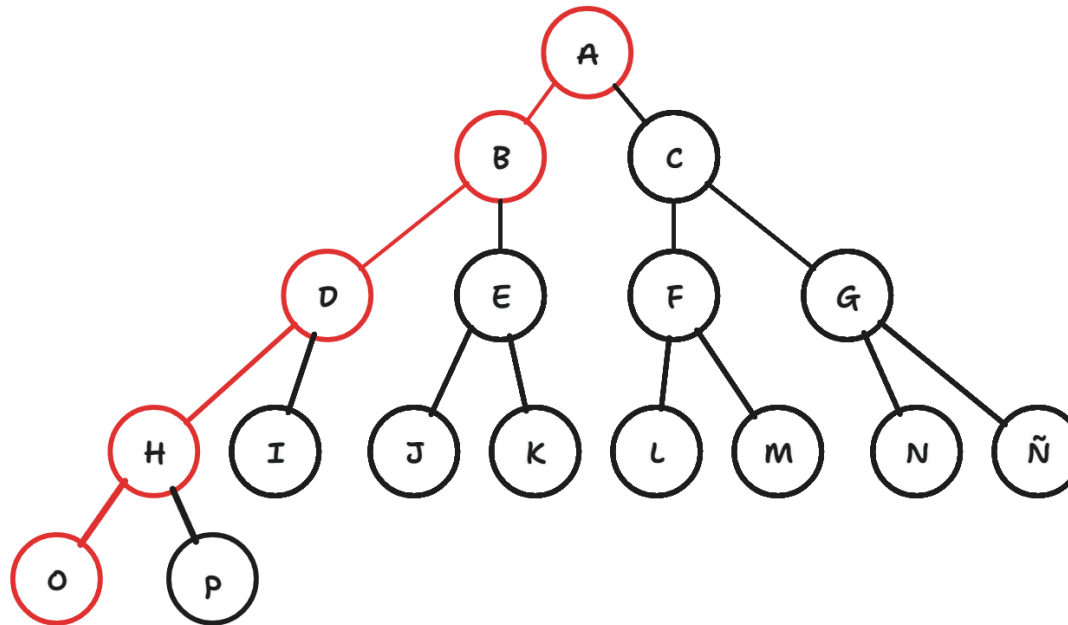
$4 / 2 = 2 \rightarrow$ Contador = 3

$2 / 2 = 1 \rightarrow$ Contador = 4



Si n = 16

$2^4 = 16 \rightarrow \log_2(16) = 4$

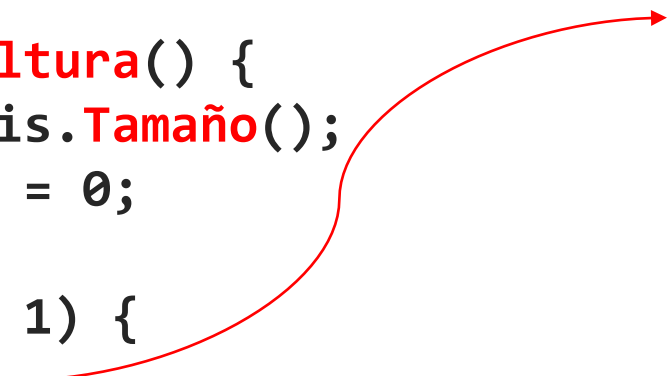


-	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

6. Consulta 2 (altura de la Heap)

□ Divisiones sucesivas y un Contador.

```
public int Altura() {  
    int n = this.Tamaño();  
    int altura = 0;  
  
    while (n > 1) {  
        n >>= 1;  
        altura++;  
    }  
    return altura;  
}
```



Desplazamiento hacia la derecha
n = 16. Contador = 0.

16 = 0001 0000

Desplazamiento, Contador = 1.

8 = 0000 1000

Desplazamiento, Contador = 2.

4 = 0000 0100

Desplazamiento, Contador = 3.

2 = 0000 0010

Desplazamiento, Contador = 4.

1 = 0000 0001

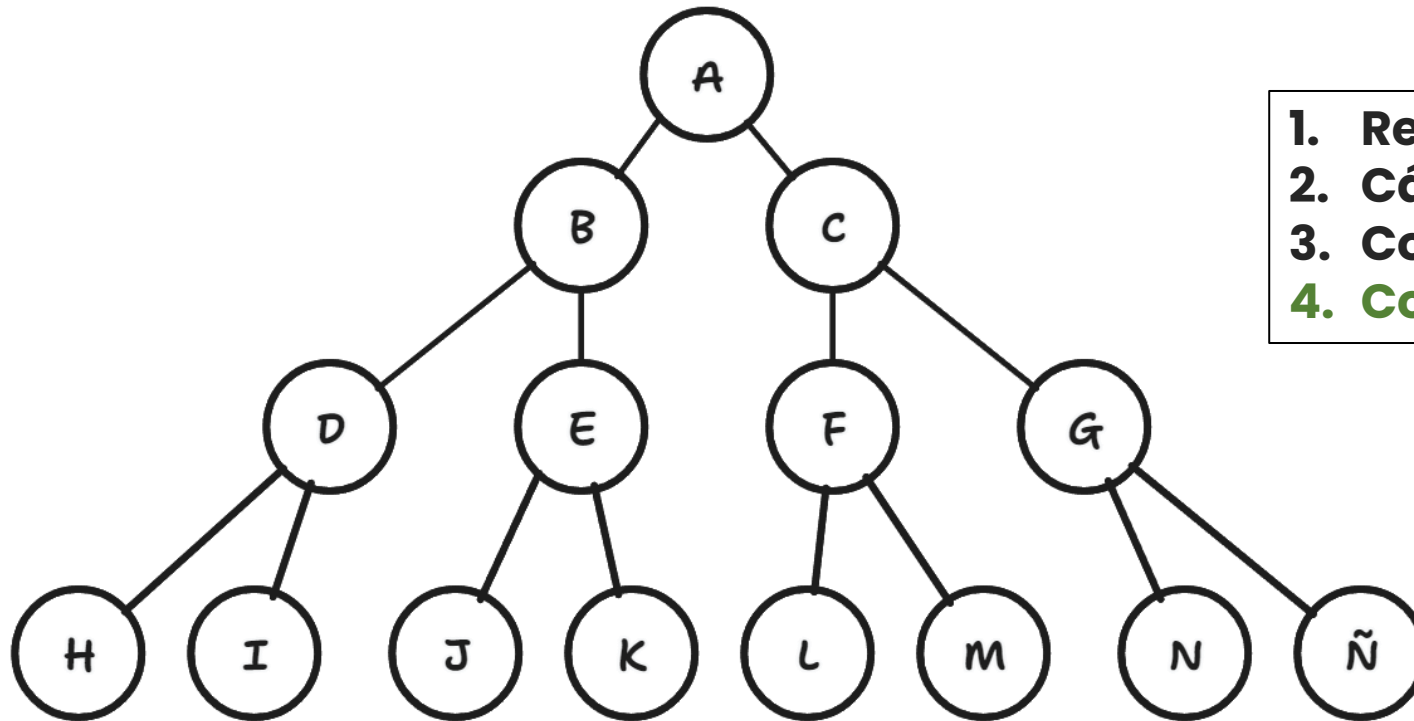
7. Consulta 3 (datos de la Heap y niveles)

```
public string ObtenerNiveles(){
    string resultado = "";
    int nivelActual = -1;
    int n = this.Tamaño();
    for (int i = 1; i <= n; i++) {
        int nivel = 0;
        int indice = i;

        while (indice > 1) {
            indice >>= 1;
            nivel++;
        }
        if (nivel != nivelActual) {
            nivelActual = nivel;
            resultado += "\nNivel " + nivelActual + ":\n";
        }
        Proceso proceso = heap[i];
        resultado += "Proceso: " + proceso.nombre +
            ", Tiempo: " + proceso.tiempo +
            ", Prioridad: " + proceso.prioridad + "\n";
    } // fin del for
    return resultado;
}
```

1. Recorrido de nodos
2. Cálculo de nivel
3. Contador de niveles
4. Concatenación

7. Consulta 3 (datos de la Heap y niveles)



1. Recorrido de nodos
2. Cálculo de nivel
3. Contador de niveles
4. Concatenación

-	A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

FIN
