2. (15 puntos) Determinar el orden de complejidad del algoritmo no recursivo óptimo (evaluar el mejor y peor caso).

```
x = 0;
for (i = 1; i <= n; i++){
  j = 1;
  k = 0;
  while (j <= i){
    j *= 2;
    k++;
    x += 2;
  }
}
```

Se pide:

a) (5 puntos) Seleccionar O.A.
b) (10 puntos) Contar O.A
c) (5 puntos) Definir la complejidad (Notación O)

Explicar brevemente cada uno de los puntos.

a) Operación activa: Operación particular fundamental en el problema.

j = 1;	j *= 2;
k = 0;	k++;
	x += 2;

Punto (+1) por cada instrucción correcta.

b) Conteo O.A.:

Ciclo:

i = 1	j = 2 k=1	k = 1 veces
i = 2	j = 2, j = 4 k = 1, k = 2	k = 2 veces
i = 3	j = 2, j = 4 k = 1, k = 2	K = 2 veces
...	...	...
i = n	j = 2, j = 4, ..., k = 1, k = 2, ...	2^k = j, el número de iterac iones k depende de j

Entonces, para calcular el número de iteraciones totales dependiendo de j se calcula.

$2^k = j$ /Aplico $\log_2()$

$\log_2(2^k) = \log_2(j)$

$k*\log_2(2) = \log_2(j)$

$k = \log_2(j)$ ===> Número de iteraciones dado j

El ciclo for se ejecuta n veces, $\log_2(j)$ veces internamente.

$T(n) = \sum_{j=1}^{n} log_2(n)$

+2 puntos por indicar y justificar el mejor caso es igual al peor caso.

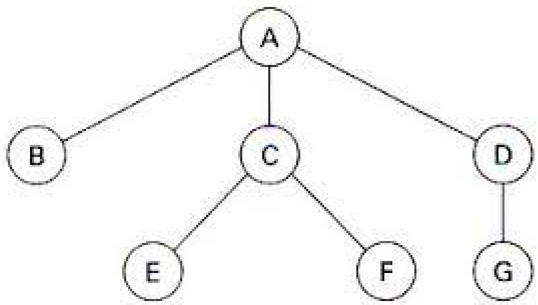
+8 puntos, explicación de conteo de ejecución de instrucciones.

c) Si consideramos el peor caso, la complejidad será: $O(n*\log_2(n))$

+5 puntos, complejidad correcta.

3. (20 puntos) Implemente un algoritmo iterativo que permita imprimir un árbol por niveles. Utilice la siguiente interfaz para la función:

```
void printLevels(Nodo* root){
  …
}
```



Por ejemplo, si ejecuta **printLevels(root)**, el algoritmo mostraría:

```
A
B C D
E F G
```

No se considerará como respuesta correcta la salida A B C D E F G.

En la prueba se indica que el árbol es de tipo binario.

```
void printLevels(Nodo* root){
  Cola* c;
  c.push(root);
  int k = 0;
  while(!c.empty()){
    int noHijos = 0;
    for(i = 0; i < 2^k; i++){
      Nodo* aux = c.pop();
      cout << aux->data << endl;
      if (aux->hijoDer){
        c.push(aux-hijoDer);
      }else{
        noHijos++;
      }
      if (aux->hijoIzq){
        c.push(aux-hijoIzq);
     }else{
        noHijos++;
     }
  }
  k = 2*k - noHijos;
  } //Fin while
}
```

4. (20 puntos) Dibuje las inserciones y eliminaciones de un árbol AVL. Indique claramente el factor de balance y el tipo de rotación en caso de ser necesario.

a) (10 puntos) Insertar 68, 45, 29, 75, 90, 70 y 34.
b) (10 puntos) Sobre el árbol generado en el punto a), elimine los siguientes nodos: 68, 45, 34 y 90. (Reemplace el mayor de los menores).

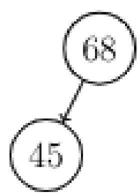
Sea ordenado en la inserción y eliminación del árbol.

Se considera el **factor de balance** como $fb = Nivel\ Hijo\ Izquierdo\ -\ Nivel\ Hijo\ Derecho$. Los valores del lado izquierdo de cada uno de los nodos indica el factor de balace, fb. Los valores con signo $+$ muestran desbalances hacia la izquierda, mientras, que los valores con signo $-$ muestran desbalances hacia la derecha.

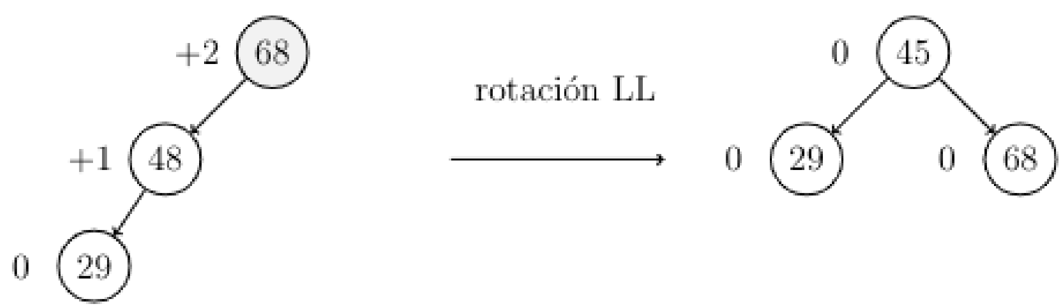
a) Insertar nodos.

Paso 1: Insertar 68

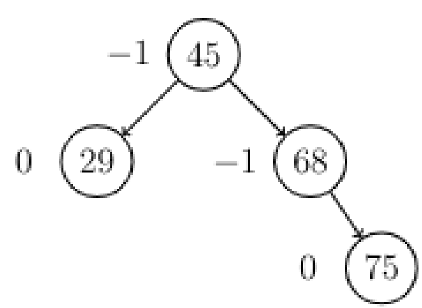
Paso 2: Insertar 45.



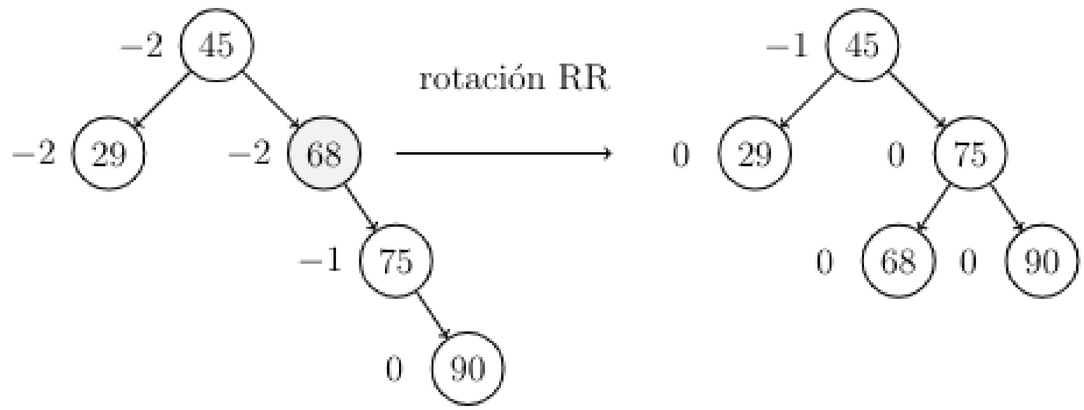
Paso 3: Insertar 29.



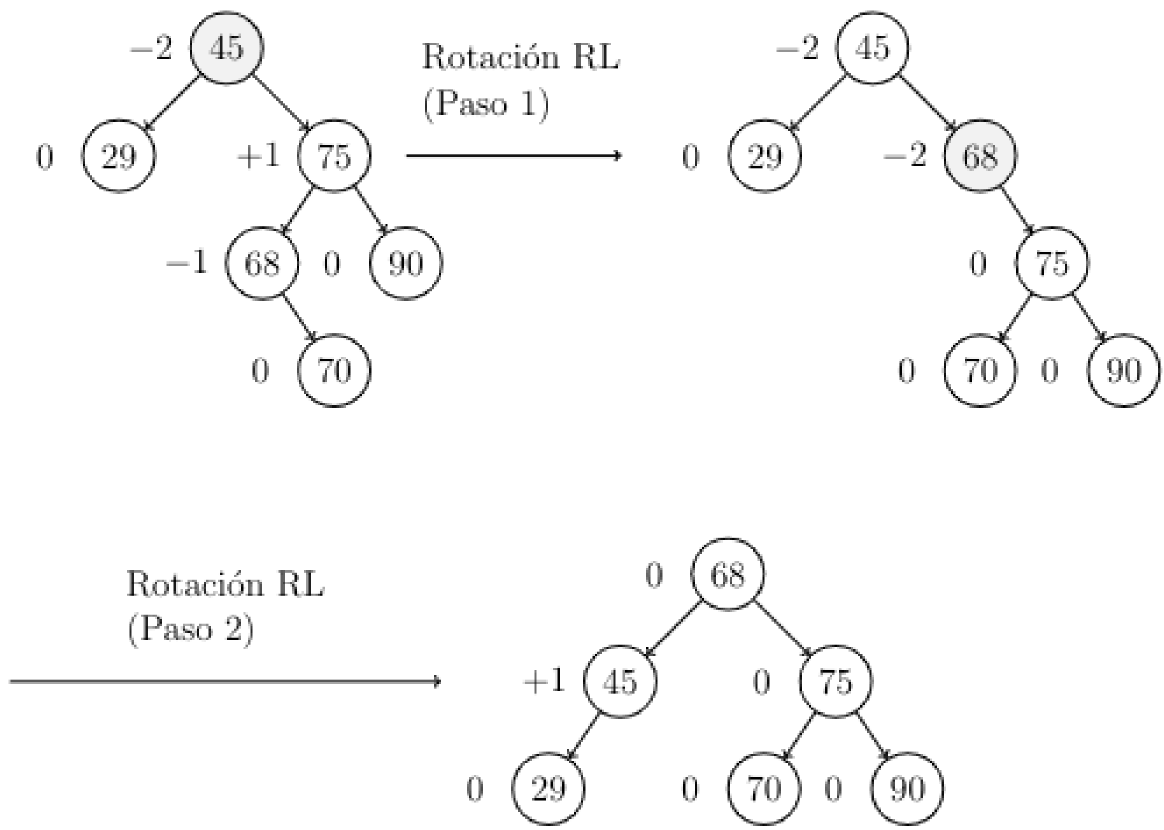
Paso 4: Insertar 75.



Paso 5: Insertar 90.

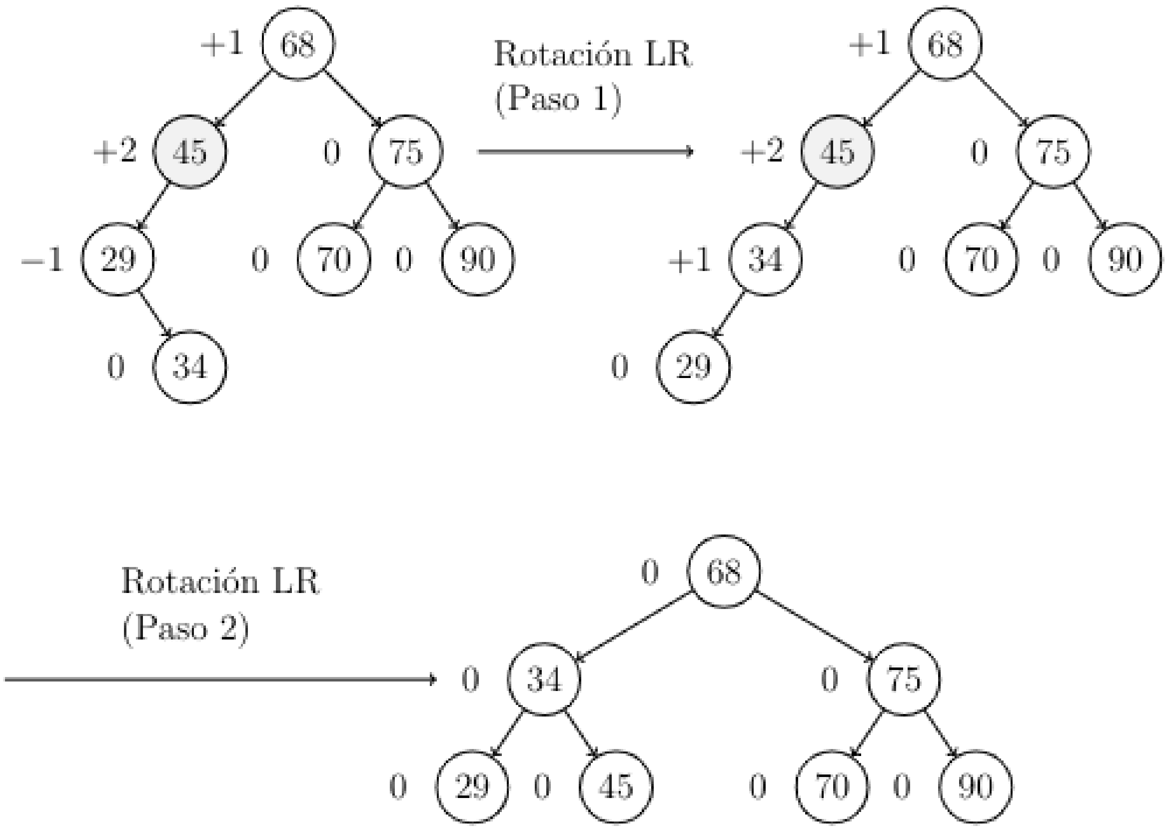


Paso 6: Insertar 70.



Paso 7: Insertar 34.

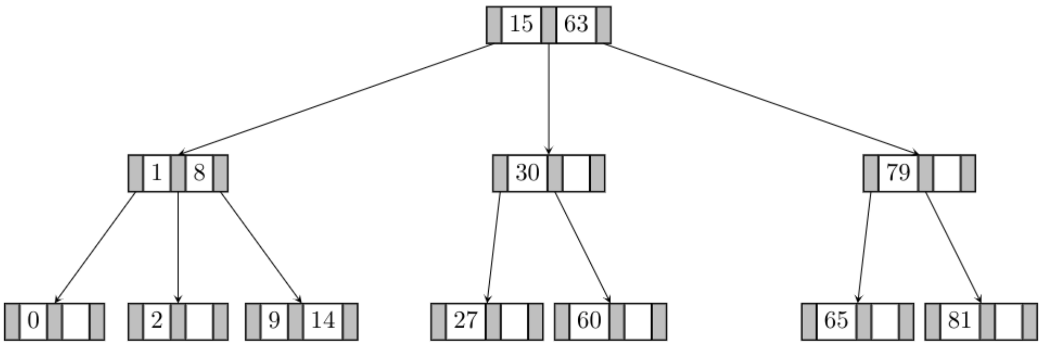
Paso 7: Insertar 34.



b) Elminar nodos.

1. Eliminar 68.

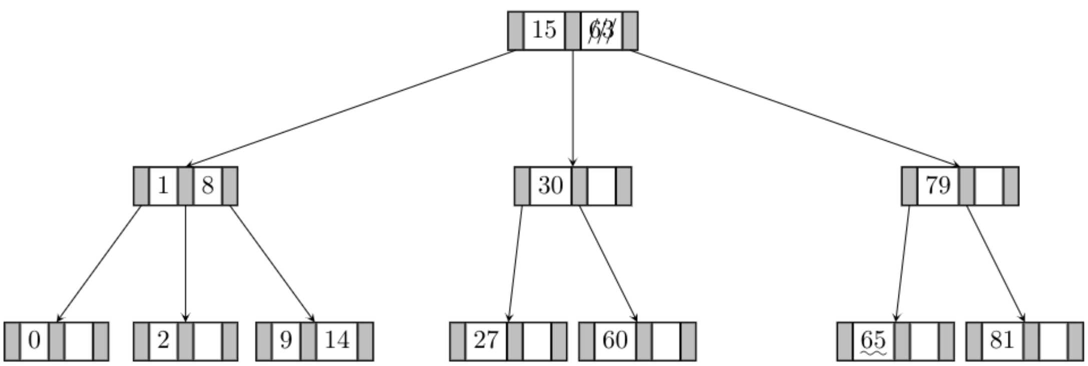
5. **(10 puntos) Eliminar el nodo 63 en el siguiente árbol B (M=3).**



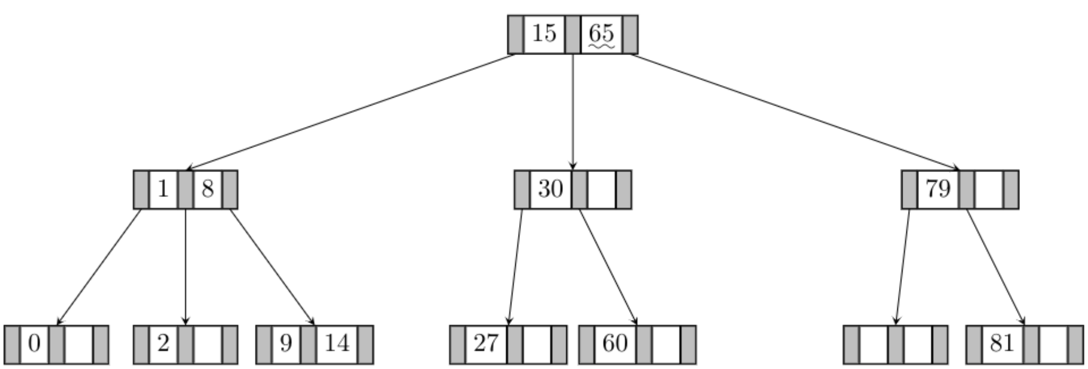
Indicar, en caso de ser necesario, si el nodo se debe redistribuir o mezclar. Explicar brevemente cada una de las operaciones que se realicen.

Solución:

a) Eliminamos el valor 63 y seleccionamos el nodo de menor valor del subárbol de mayores.

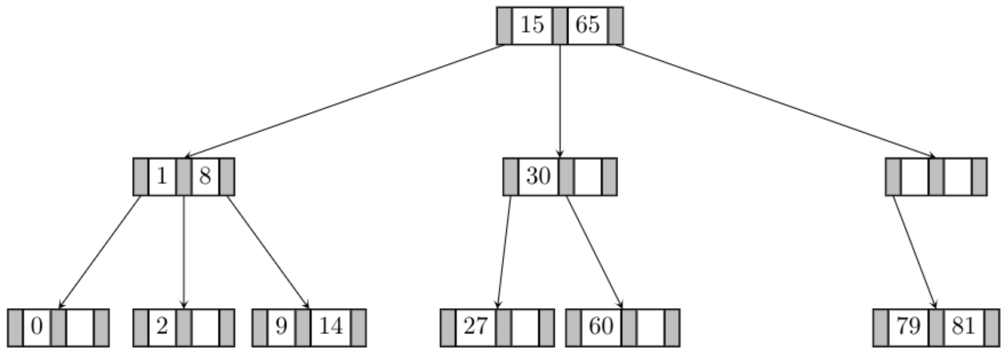


b) Reemplazamo el valor eliminado 63 por el valor 65.

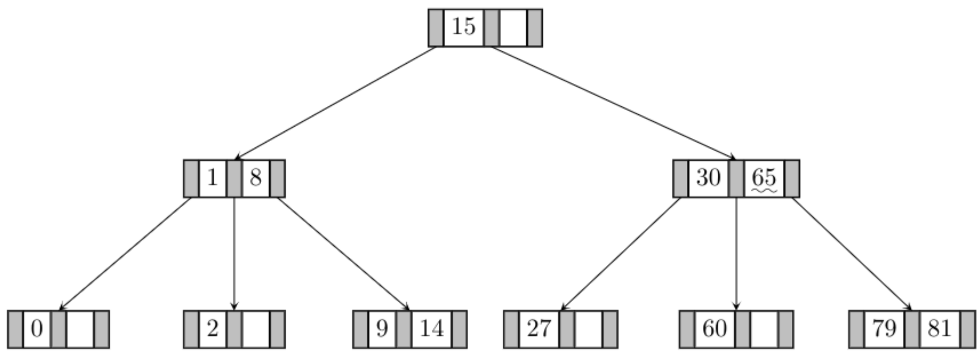


c) En el nivel tres del árbol B, se produce un **underflow**. Como no hay nodos hermanos que puedan ceder un valor al nodo faltante, se aplica:

1. *Underflow*: Mezcla 79 81.



2. Como el nodo del nivel 1 tiene elementos, fusiona el nodo del nivel 2 que tiene elementos faltantes.

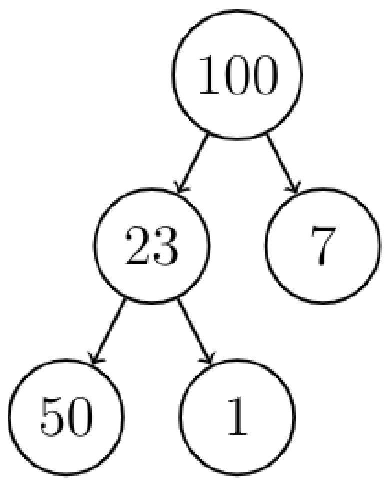


6. (15 puntos) Convierta el siguiente árbol a un MIN-HEAP. La primera fila representa los índices de un arreglo y la segunda sus valores.

índice	0	1	2	3	4	5
valor		100	23	7	50	1

a) (5 puntos) Dibuje el árbol representado por el arreglo.
b) (10 puntos) Dibuje el árbol convertido a MIN-HEAP. (Explique paso a paso)

a)



b)

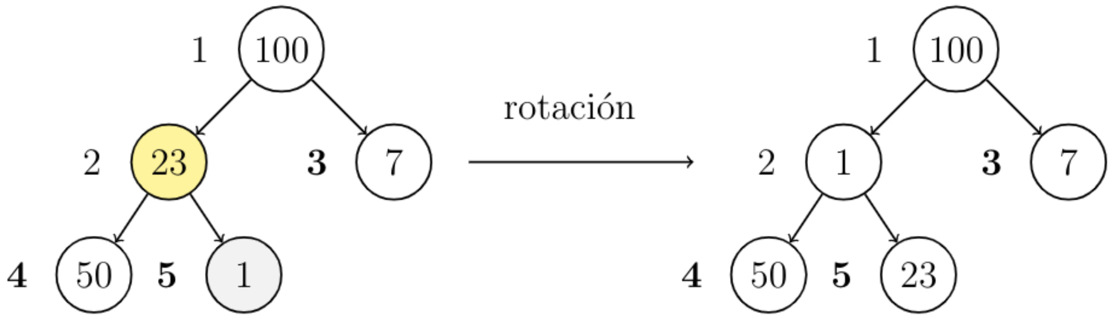
Solución:

- Para convertir el árbol a un MIN-HEAP hay que estimar cuantos elementos terminales hay en el árbol por definición. Si N=5, los nodos desde $\lfloor \frac{N}{2} \rfloor + 1$ a N cumplen con ser terminales y con la propiedad de ser HEAP.
- Hay que analizar desde $\lfloor \frac{N}{2} \rfloor$ hasta 1 (en ese orden), si cumple con la propiedad de ser HEAP.

En nuestro caso, el nodo etiquetado sería el nodo $\lfloor \frac{N}{2} \rfloor$ cuyo índice es 2 debe cumplir con la propiedad de ser MIN-HEAP.

Solución:

1. El nodo con índice 2 tiene que ser MIN-HEAP. Para eso, se rota el valor del índice 5 con el valor contenido en el índice 2.



2. El nodo con índice 1 tiene que ser MIN-HEAP. Para eso, se rota el valor del índice 2 con el valor contenido en el índice 1. Luego, el valor del índice 2 con el valor del índice 5.