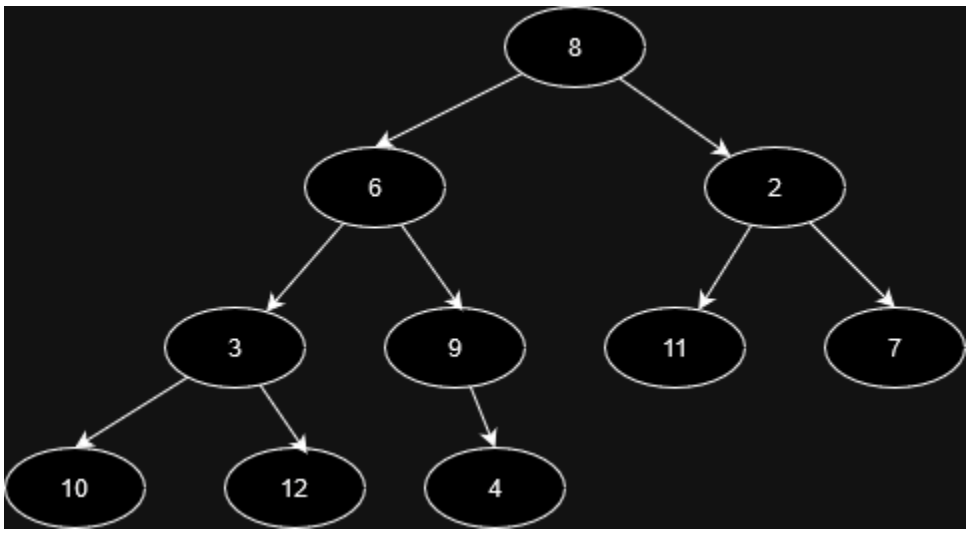
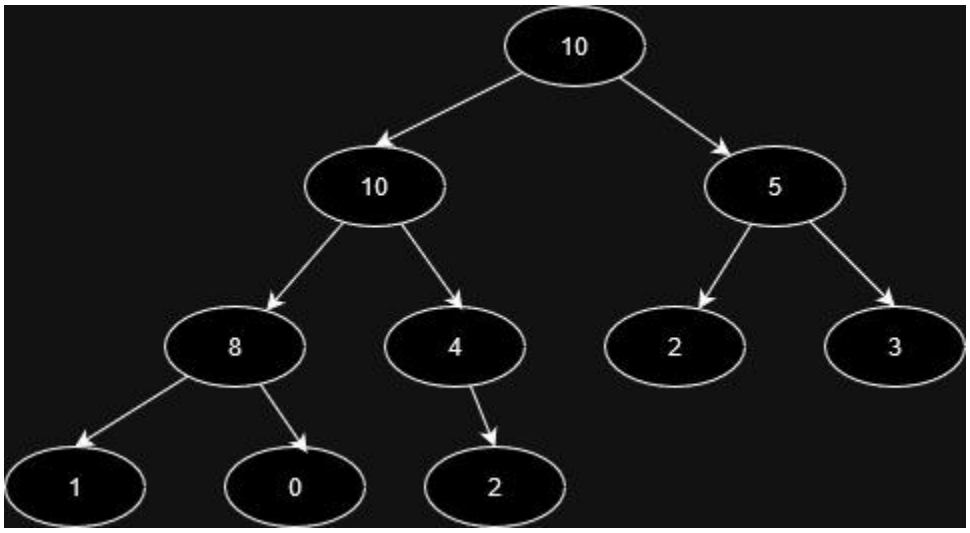
1. ¿Constituye el siguiente arreglo un montículo máximo?

A = [8,6,2,3,9,11,7,10,12,4] R/ No
Observen que si construyen el árbol se observa de esta manera:



Podemos observar que no se respeta que todo nodo sea mayor o igual a los valores de sus hijos, por ende no puede ser un max heap.

A=[10,10,5,8,4,2,3,1,0,2] R/ Sí



Podemos observar que sí se respeta que todo nodo sea mayor o igual a los valores de sus hijos, haciéndolo en efecto un max heap.

2. Utilizando el siguiente pseudocódigo como base, modifíquelo para crear un algoritmo que produzca un min heap.

```
Input: A: Arreglo que representa un montículo
Input: i: Un índice del arreglo A

function Work(A, i)
  l <- LEFT(i)
  r <- RIGHT(i)
  if l <= A.heapsize ^ A[l] > A[i] then
    targetNode <- l
  else
    targetNode <- i
  end if
  if r <= A.heapsize ^ A[r] > A[targetNode] then
    targetNode <- r
  end if
  if targetNode != i then
    Swap(A[i], A[targetNode])
    Work(A, targetNode)
  end if
end function
```

Versión modificada **correcta**:

```
Input: A: Arreglo que representa un montículo
Input: i: Un índice del arreglo A

function Work(A, i)
  l <- LEFT(i)
  r <- RIGHT(i)
  if l <= A.heapsize ^ A[l] < A[i] then
    targetNode <- l
  else
    targetNode <- i
  end if
  if r <= A.heapsize ^ A[r] < A[targetNode] then
```

```
    targetNode <- r
  end if
  if targetNode != i then
    Swap(A[i], A[targetNode])
    Work(A, targetNode)
  end if
end function
```

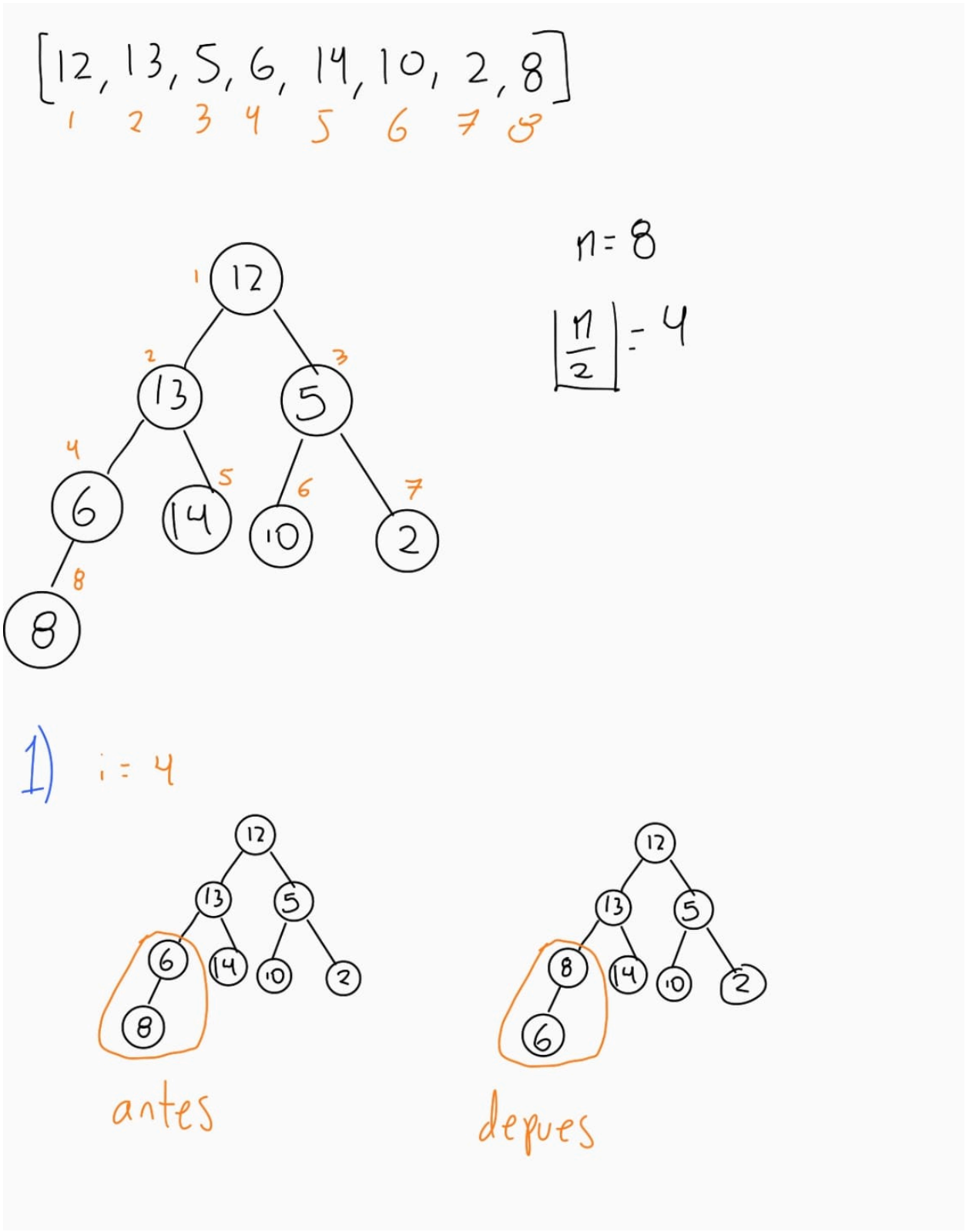
Noten que los valores que contienen r,l son índices del arreglo. Por eso ambos if revisan que estén dentro del rango del arreglo antes de accederlo. No queremos acceder algo que no existe, por esta misma razón cambiar los if a esto no tiene sentido:

Versión modificada **incorrecta**:

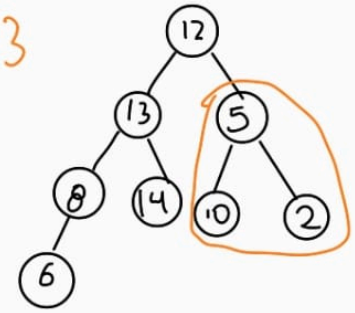
```
Input: A: Arreglo que representa un montículo
Input: i: Un índice del arreglo A

function Work(A, i)
  l <- LEFT(i)
  r <- RIGHT(i)
  if l >= A.heapsize ^ A[l] < A[i] then
    targetNode <- l
  else
    targetNode <- i
  end if
  if r >= A.heapsize ^ A[r] < A[targetNode] then
    targetNode <- r
  end if
  if targetNode != i then
    Swap(A[i], A[targetNode])
    Work(A, targetNode)
  end if
end function
```

Porque siempre estamos preguntando si los índices que tenemos están fuera (o en la última posición) del arreglo, lo cuál no tiene sentido ya que estos if los queremos usar para reacomodar el heap.

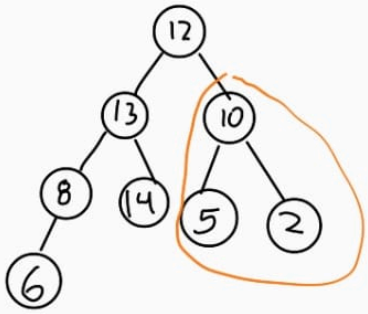
3. 1



2) i = 3

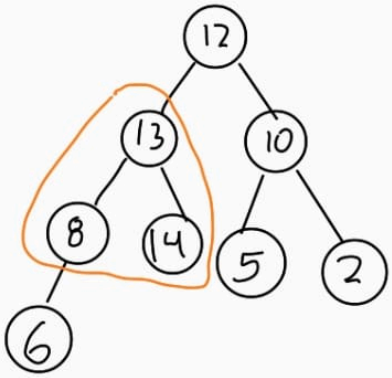


antes

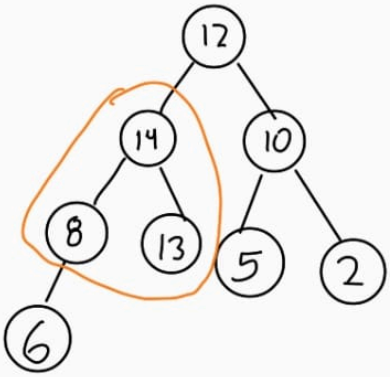


depues

3) i = 2

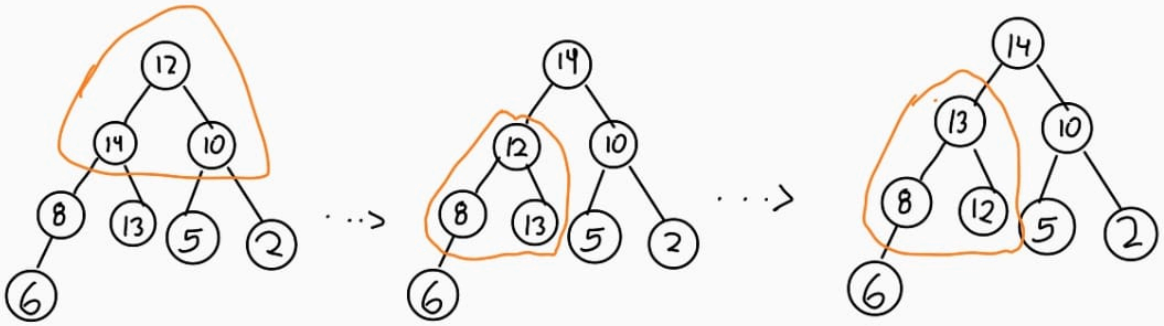


antes



despues

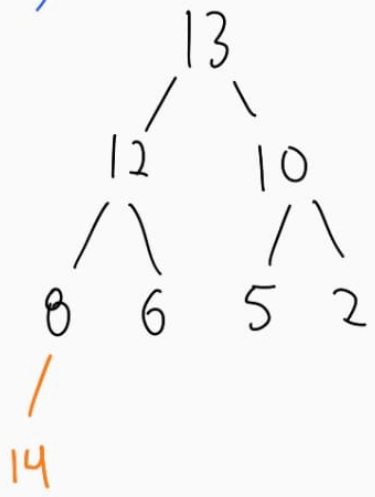
4) i = 1



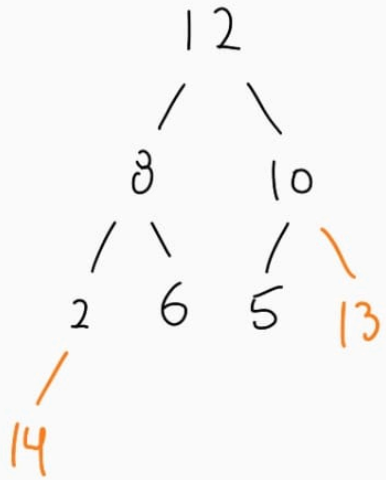
[14, 13, 10, 8, 12, 5, 2, 6]

ORDENAR

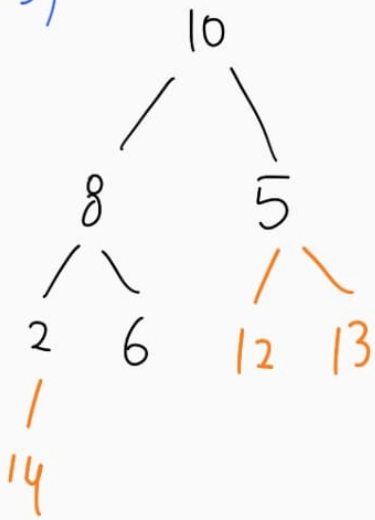
1)



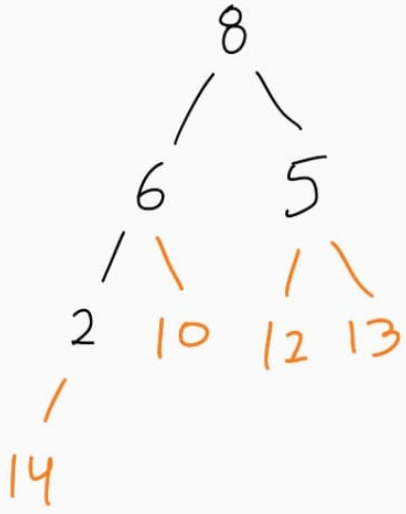
2)



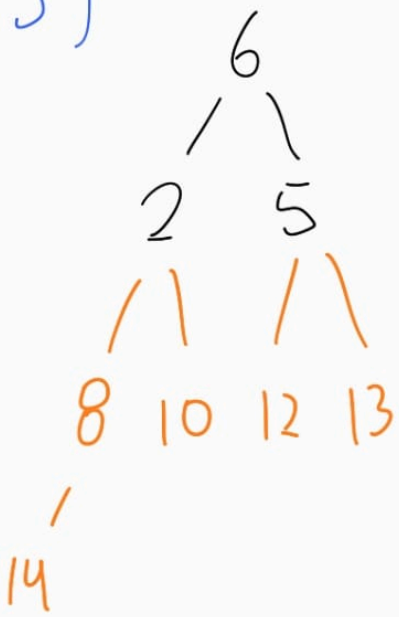
3)



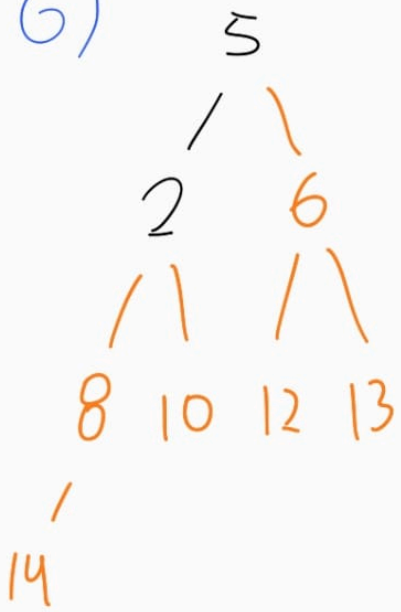
4)



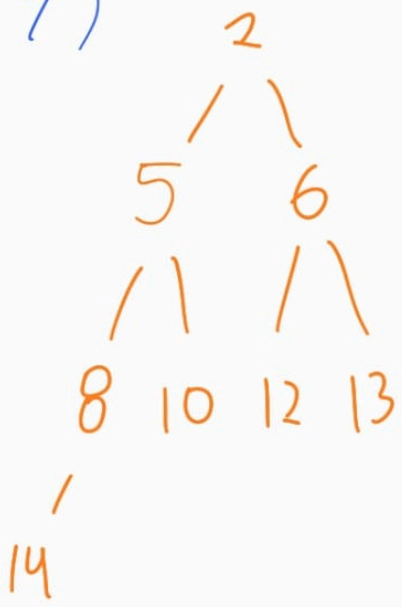
5)



6)



7)



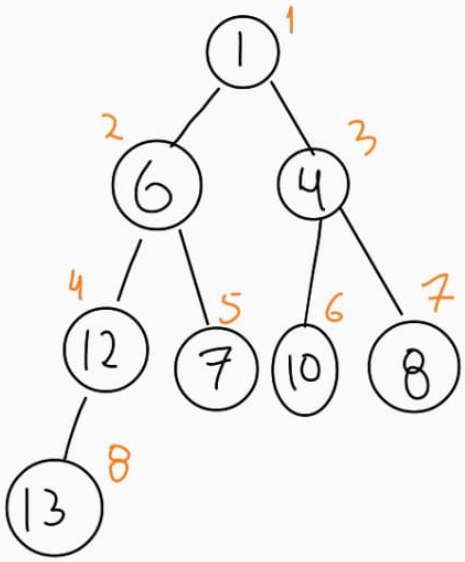
[2, 5, 6, 8, 10, 12, 13, 14]

3.2

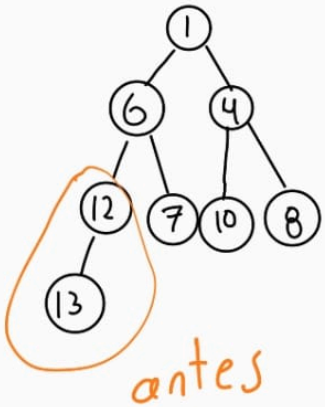
[1, 6, 4, 12, 7, 10, 8, 13]
1 2 3 4 5 6 7 8

$n = 8$

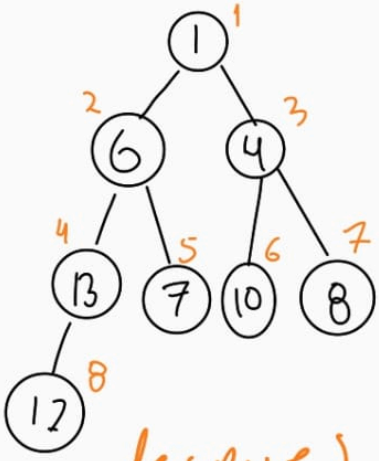
$\lfloor \frac{n}{2} \rfloor = 4$



1) $i = 4$

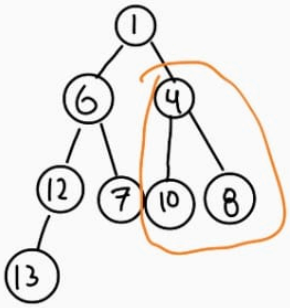


antes

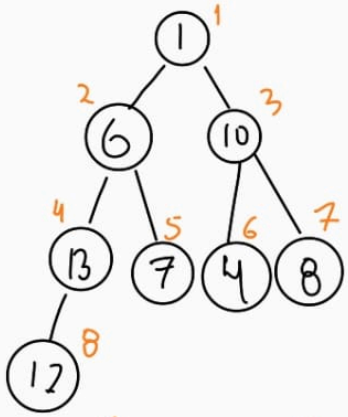


despues

2) i = 3

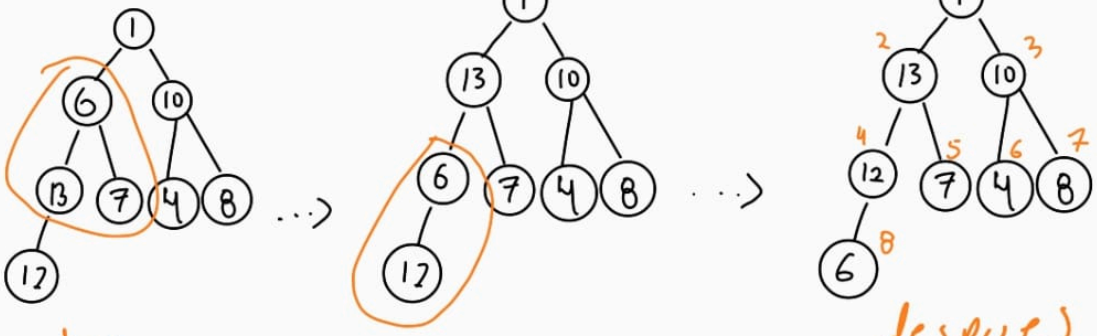


antes



despues

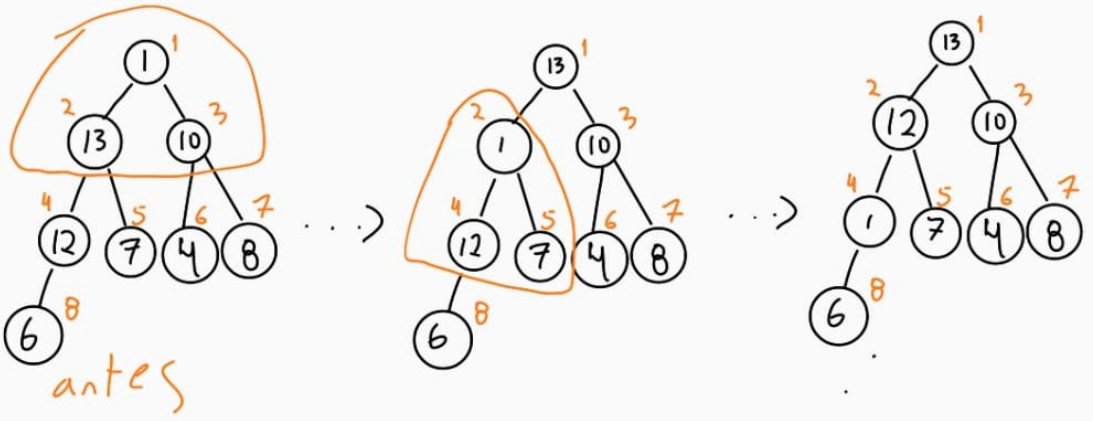
3) i = 2

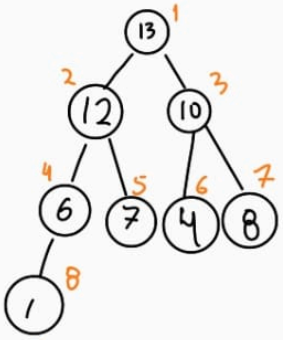


antes

despues

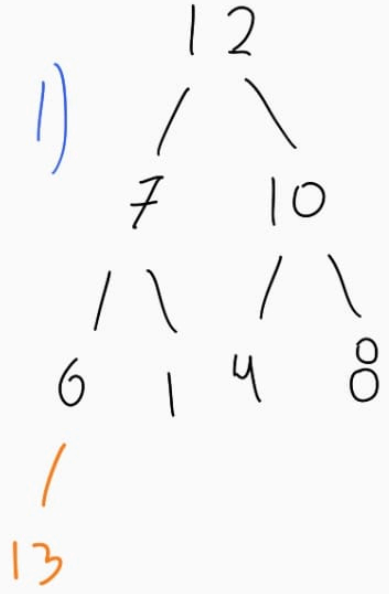
4) i = 1



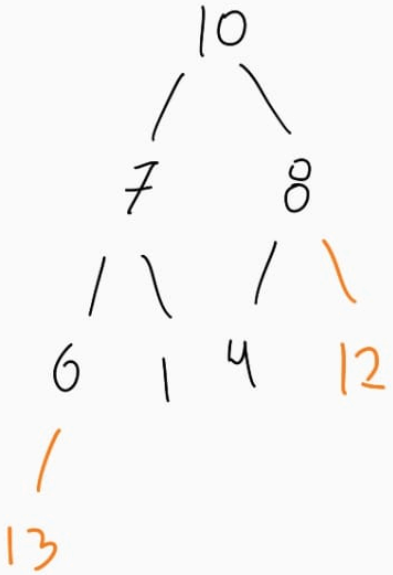


ORDENAR

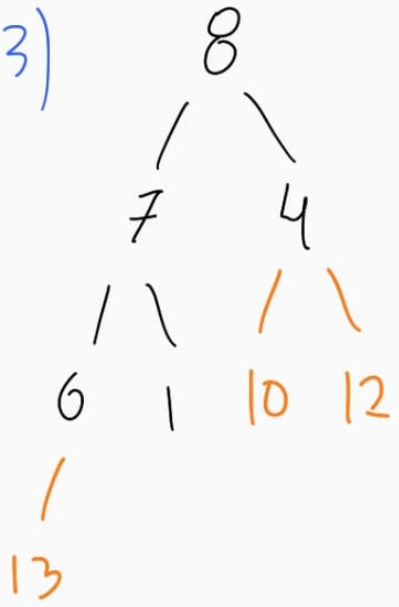
1)



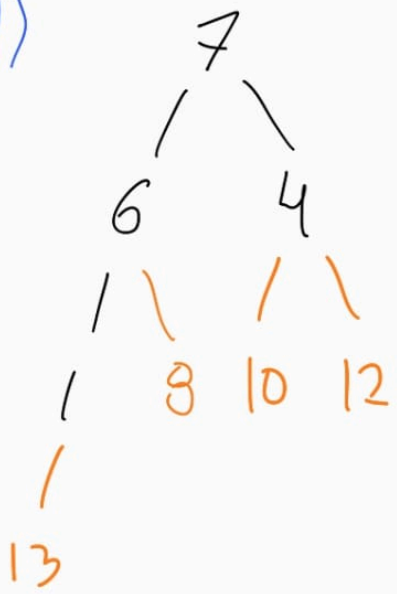
2)



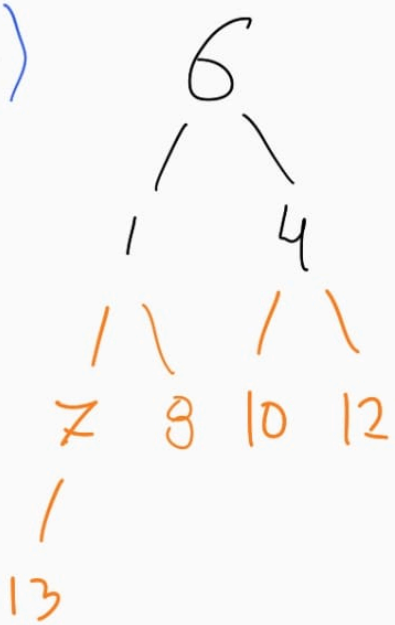
3)



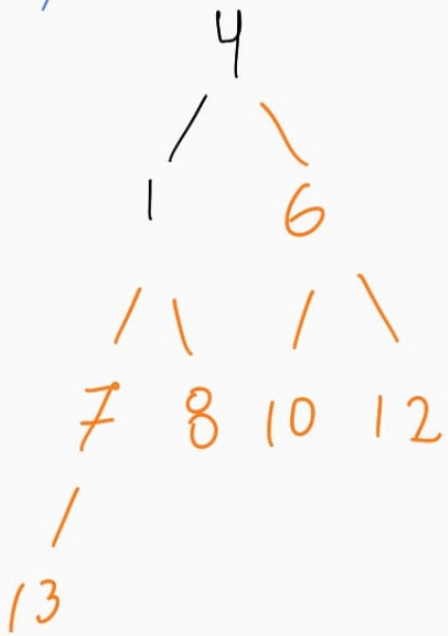
4)



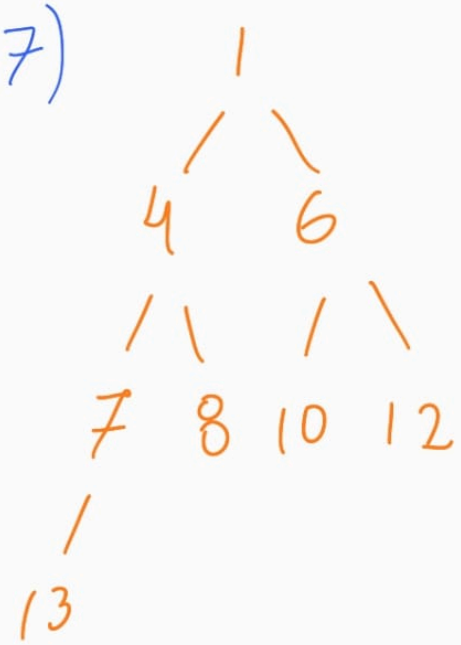
5)



6)



7)



[1, 4, 6, 7, 8, 10, 12, 13]