(1) HEAP

Ange två eventuella implementationer (datastrukturer) för en heap. (1p)

(a) Tillämpa algoritmen "**heapify**" (se bilaga A) på sekvensen **3, 6, 7, 76, 36, 16, 2, 86, 96, 66** (anta att **det storsta värdet** ska finnas i rotpositionen) för att skapa en heap.

Förklara vad händer vid varje steg.

Varför börjar man med "**for i = [A.size / 2] downto 1**" i Build?

(2p)

(b) Använd algoritmen "**remove**" (se bilaga A) för att ta bort värdet **86** från heapen som konstruerats ovan. Ange alla antagande.
Visa resultatet och varje steg i algoritmen. Hur fungerar algoritmen?

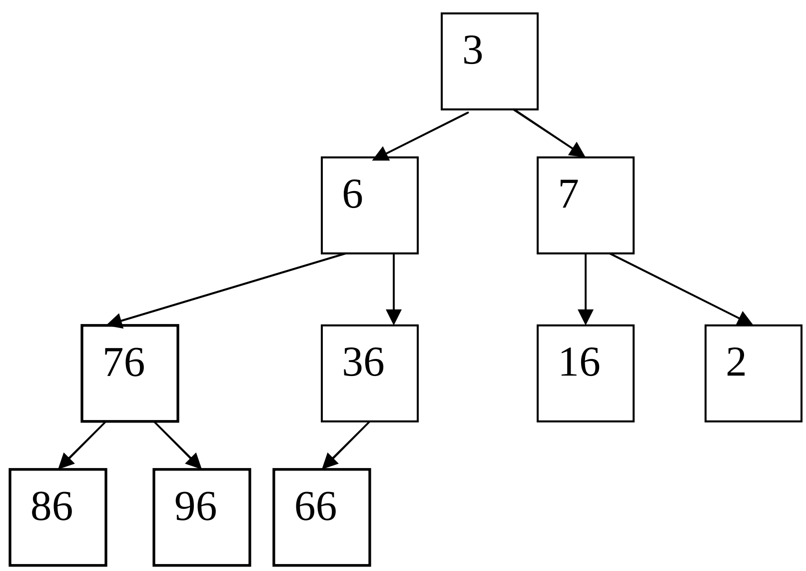
(1p)

(c) Använd algoritmen "**add**" (se bilaga A) för att lägga till värdet **90** till heapen som konstruerats ovan. Ange alla antagande.
Visa resultatet och varje steg i algoritmen. Hur fungerar algoritmen?

(1p)

Total 5p

3	**6**	**7**	**76**	**36**	**16**	**2**	**86**	**96**	**66**



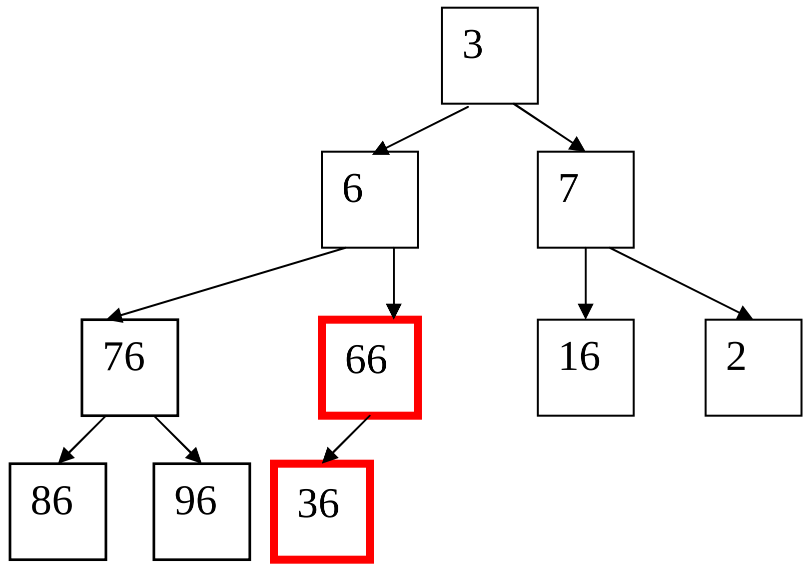
Start array and the corresponding tree

Heapify 3, 6, 7, 76, 36, 16, 2, 86, 96, 66

Array size = 10 hence we do for i = 5 downto 1 Heapify(A, i)

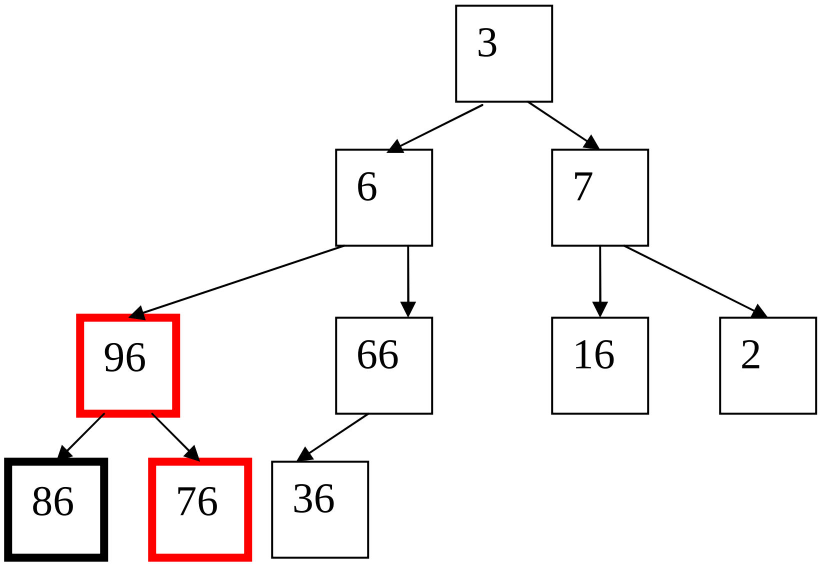
i = 5, A = **3, 6, 7, 76, 36, 16, 2, 86, 96, 66**

i = 5; (value 36); l = 10 (value 66), r = 11 (does not exist); largest = 10 (value 36)
largest != i (10 != 5) hence swap to give A = **3, 6, 7, 76, 66, 16, 2, 86, 96, 36**
Heapify(A, 10) has no effect on A (A[10] is a leaf node)



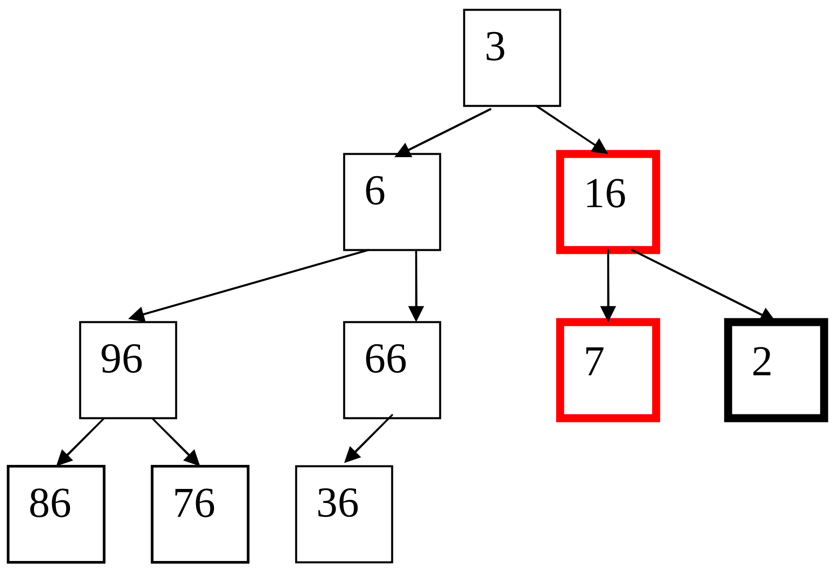
i = 4, A = **3, 6, 7, 76, 36, 16, 2, 86, 96, 66**

i = 4; (value 76); l = 8 (value 86); r = 9 (value 96); largest = 9 (value 96)
largest != i (9 != 4) hence swap to give A = **3, 6, 7, 96, 66, 16, 2, 86, 76, 36**
Heapify(A, 9) has no effect on A (A[9] is a leaf node)



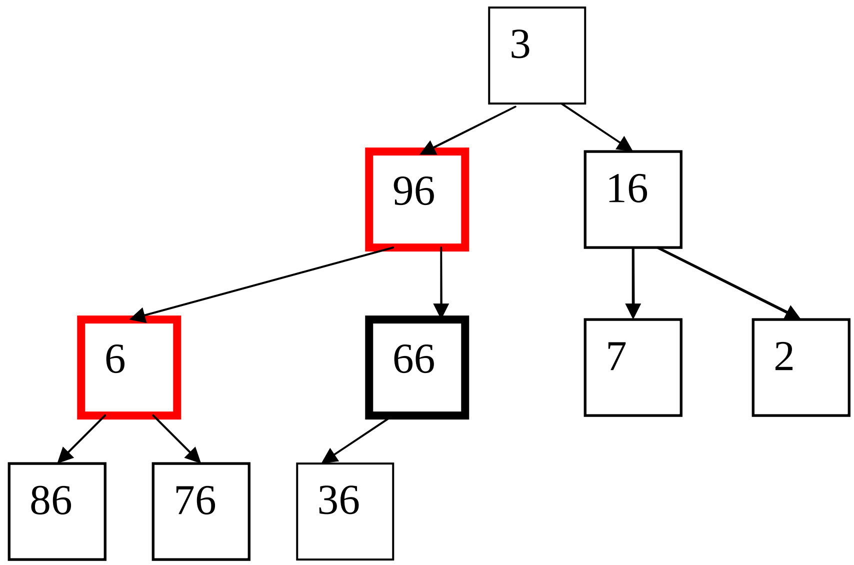
i = 3, A = **3, 6, 7, 96, 66, 16, 2, 86, 76, 36**

i = 3; (value 7); l = 6 (value 16); r = 7 (value 2); largest = 6 (value 16)
largest != i (6 != 3) hence swap to give A = **3, 6, 16, 96, 66, 7, 2, 86, 76, 36**
Heapify(A, 6) has no effect on A (A[6] is a leaf node)



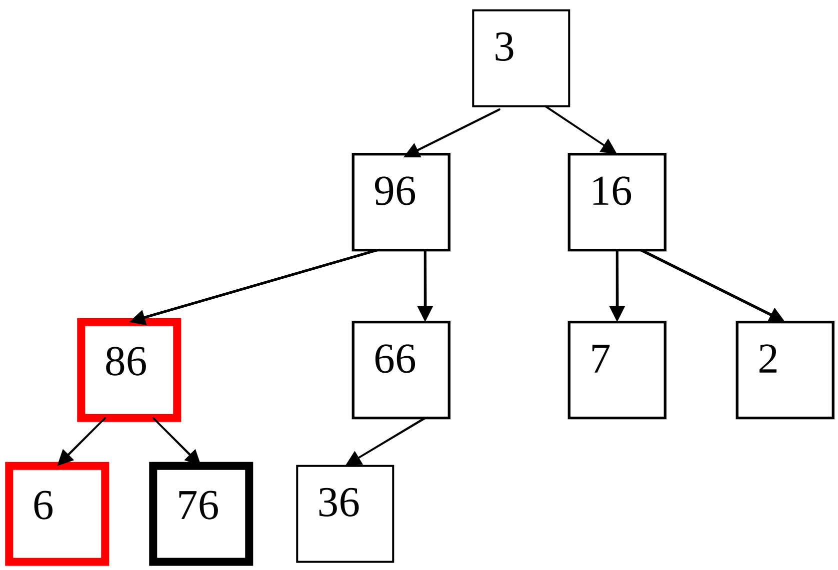
i = 2, A = **3, 6, 16, 96, 66, 7, 2, 86, 76, 36**

i = 2; (value 6); l = 4 (value 96); r = 5 (value 66); largest = 4 (value 96)
largest != i (4 != 2) hence swap to give A = **3, 96, 16, 6, 66, 7, 2, 86, 76, 36**

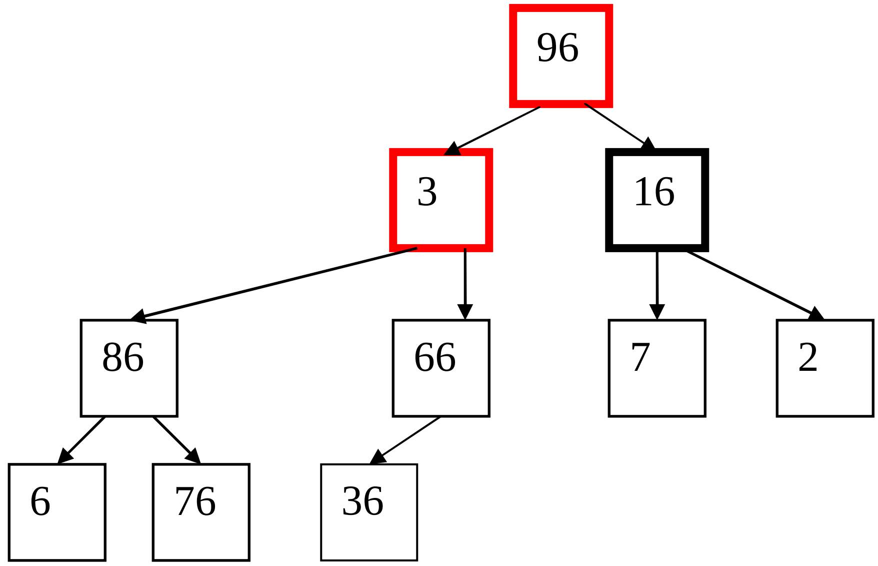


Heapify(A, 4) now has an effect since 4 is NOT a leaf node

i = 4, A = **3, 96, 16, 6, 66, 7, 2, 86, 76, 36**
i = 4; (value 6); l = 8 (value 86); r = 5 (value 76); largest = 8 (value 86)
largest != i (8 != 4) hence swap to give A = **3, 96, 16, 86, 66, 7, 2, 6, 76, 36**
Heapify(A, 8) has no effect on A (A[8] is a leaf node)



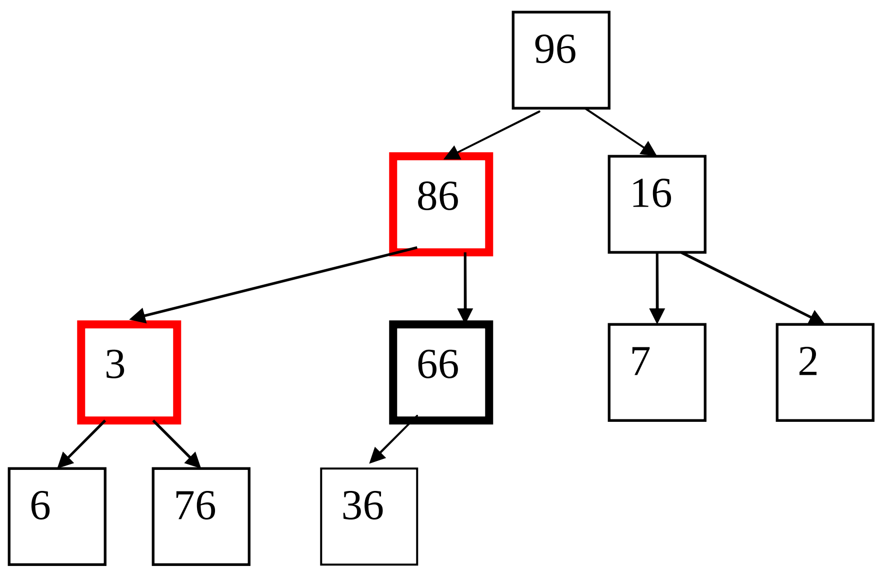
Now we come back to the first level of recursion with "i" set to 1
i = 1, A = **3, 96, 16, 86, 66, 7, 2, 6, 76, 36**
i = 1; (value 3); l = 2 (value 96); r = 3 (value 16); largest = 2 (value 96)
largest != i (2 != 1) hence swap to give A = **96, 3, 16, 86, 66, 7, 2, 6, 76, 36**



Heapify(A, 2) now starts a (second) recursive sequence

i = 2, A = **96, 3, 16, 86, 66, 7, 2, 6, 76, 36**

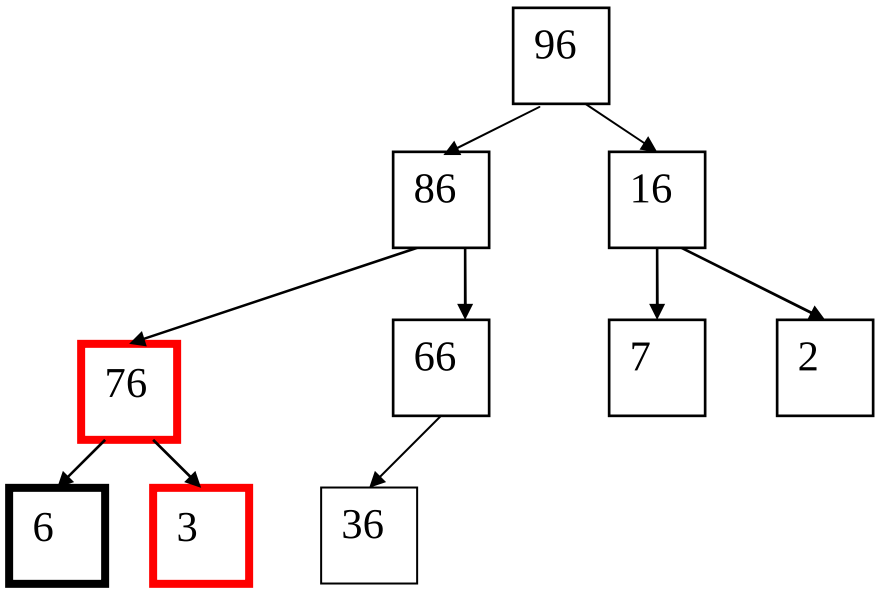
i = 2; (value 3); l = 4 (value 86); r = 5 (value 66); largest = 4 (value 86)
largest != i (2 != 4) hence swap to give A = **96, 86, 16, 3, 66, 7, 2, 6, 76, 36**



Heapify(A,4) now starts another recursive call

i = 4, A = **96, 86, 16, 3, 66, 7, 2, 6, 76, 36**

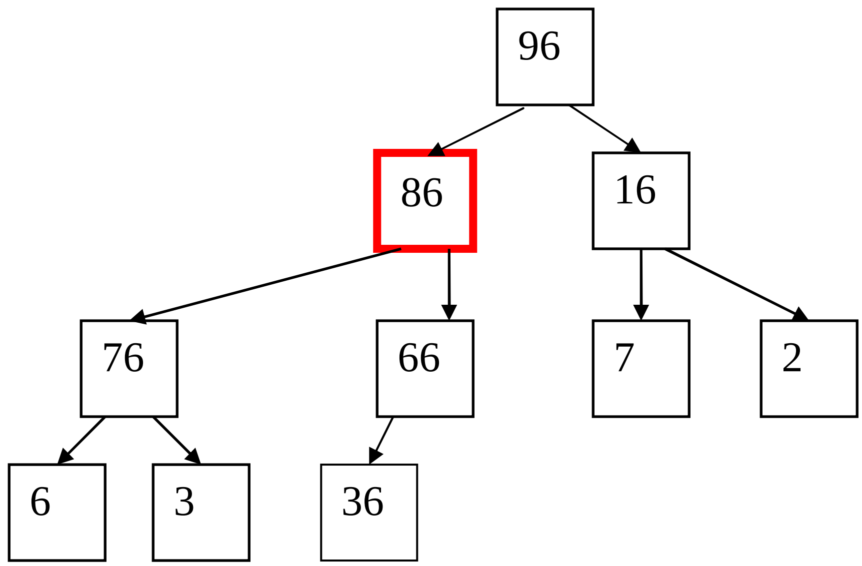
i = 4; (value 3); l = 8 (value 6); r = 9 (value 76); largest = 9 (value 76)
largest != i (9 != 4) hence swap to give A = **96, 86, 16, 76, 66, 7, 2, 6, 3, 36**



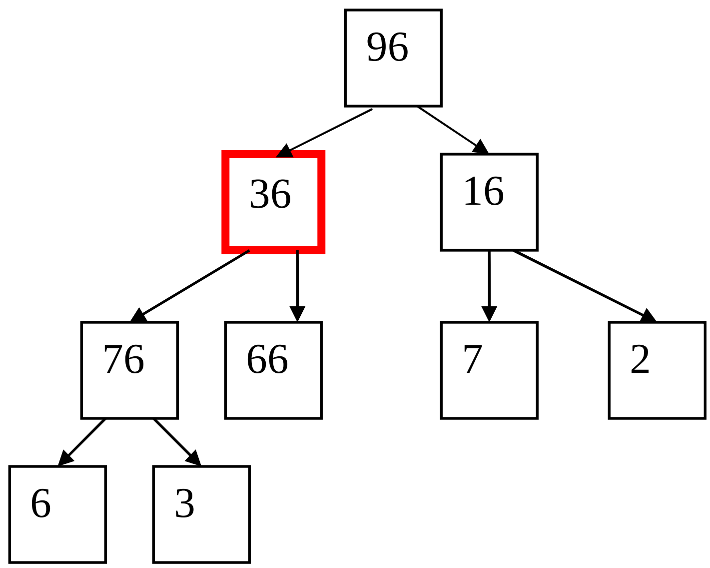
Heapify(A, 9) has no effect on A (A[9] is a leaf node)

Now the sequence has been heapified!

Remove 86 from the heap constructed above A = 96, 86, 16, 76, 66, 7, 2, 6, 3, 36

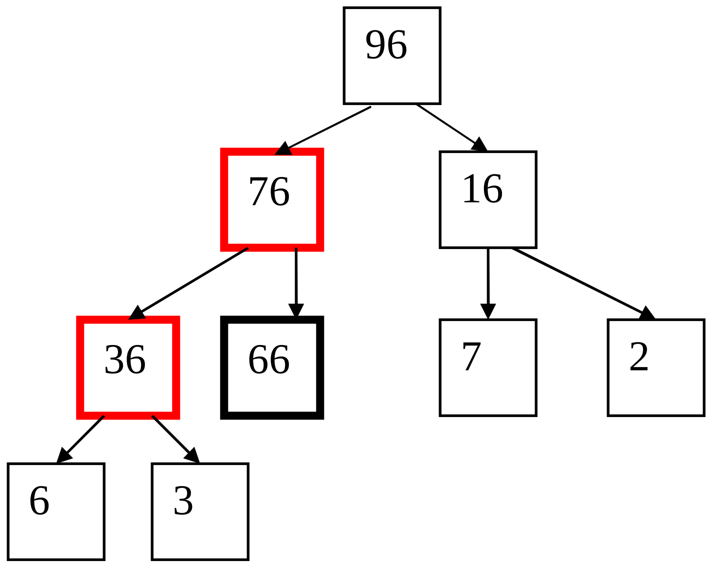


Remove(H, 2) A.size = 10 hence A[2] = A[10] ➔ A[2] = 36 and then A.size = 9



Then perform heapify on A = **96, 36, 16, 76, 66, 7, 2, 6, 3 with r = 2** i = 2 A = **96, 36, 16, 76, 66, 7, 2, 6, 3**

i = 2; (value 36); l = 4 (value 76); r = 5 (value 66); largest = 4 (value 76)
largest != i (4 != 2) hence swap to give A = **96, 76, 16, 36, 66, 7, 2, 6, 3**



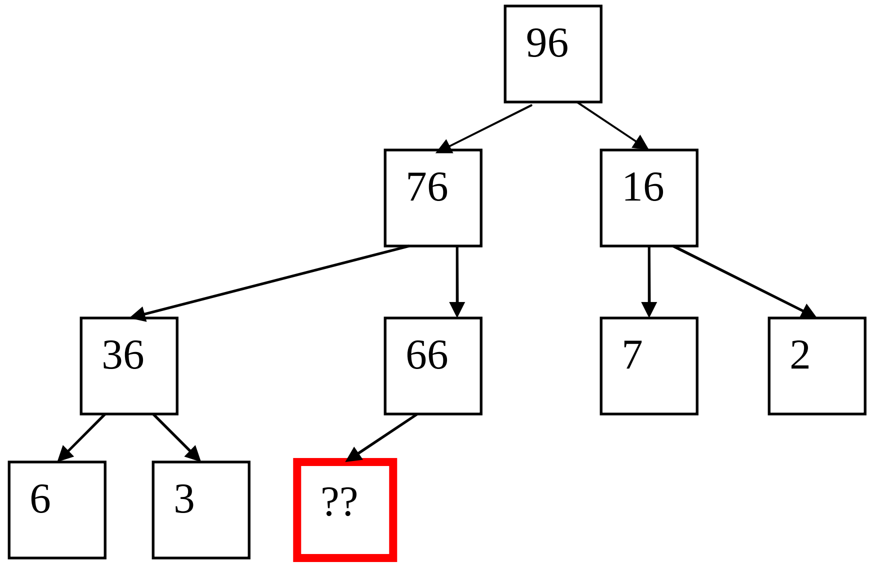
Now we have a recursive call on 4 i.e. heapify(A, 4)

i = 2 A = **96, 76, 16, 36, 66, 7, 2, 6, 3**

i = 4; (value 36); l = 8 (value 6); r = 9 (value 3); largest = 4 (value 66)

and in this case i == largest so there is no swap and the sequence has been heapified!

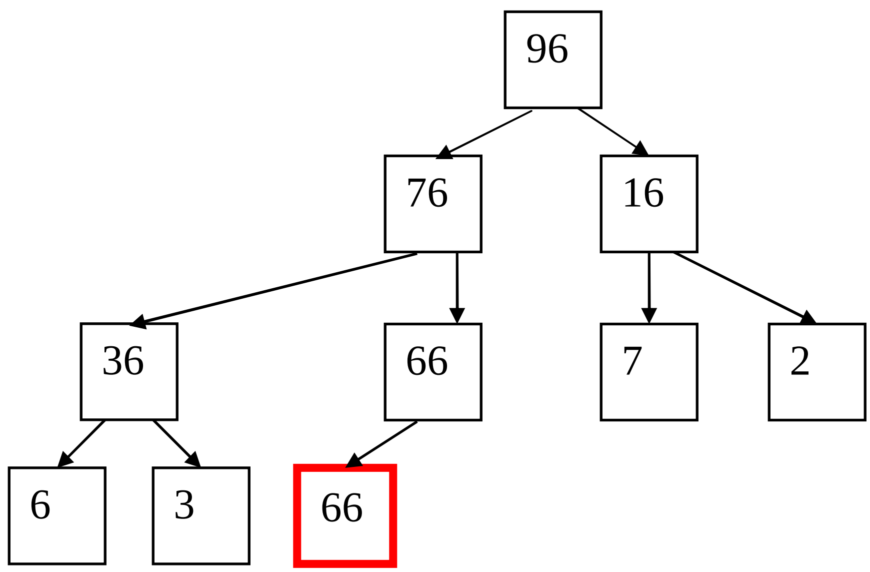
Now add 90 to the heap constructed above



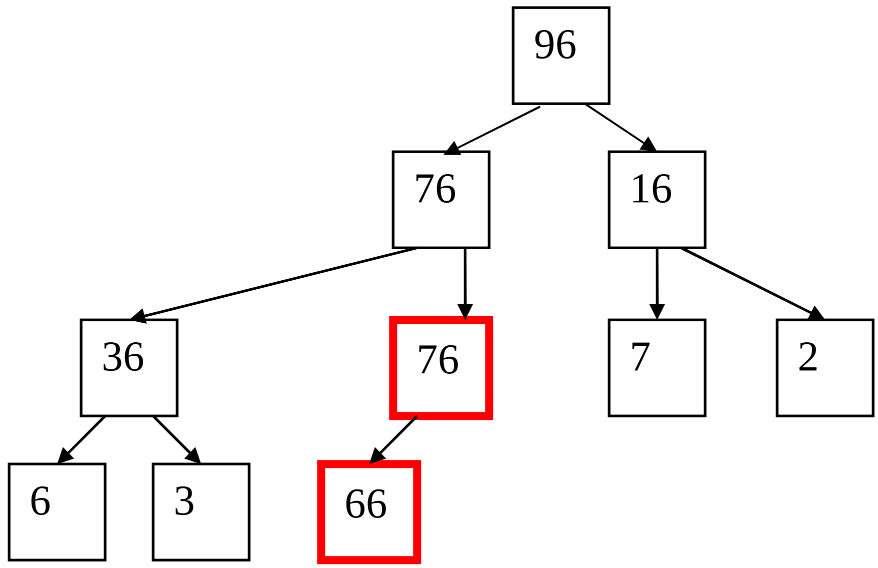
A.size is now 10 and i = A.size i.e. i is 10 and v = 90

```
Add (H, v)
        let A = H.array
        A.size++
        i = A.size
        while i > 1 and A[Parent(i)] < v do
                A[i] = A[Parent(i)]
                 i = Parent(i)
            end while
        A[i] = v
end Add
```

i = 10; while i > 1 and A[Parent(i)] < v do {A[i] = A[Parent(i)] i = Parent(i)}
so 10 > 1 (true) Parent(10) is 5; A[5] < 90 ➔ 66 < 90 ➔ A[10] = 66; i = 5;



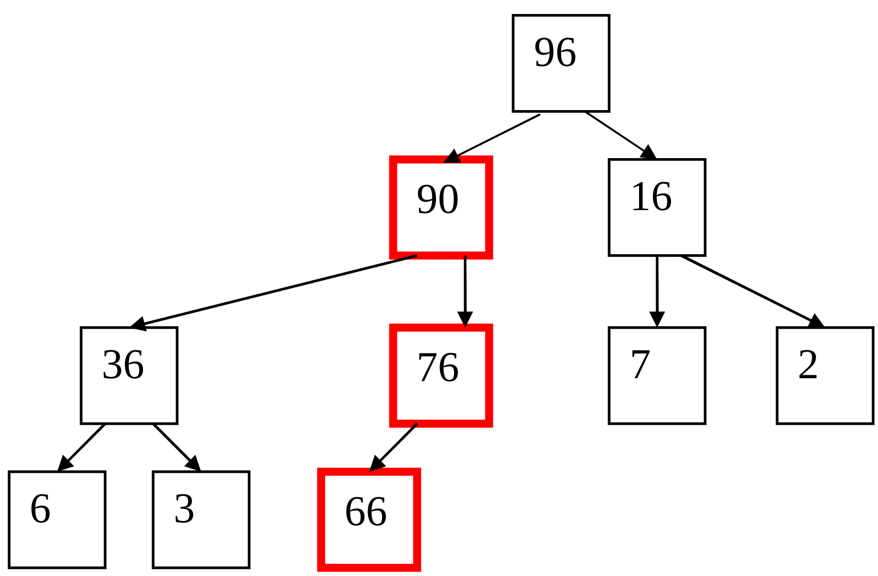
i = 5; while i > 1 and A[Parent(i)] < v do {A[i] = A[Parent(i)] i = Parent(i)}
so 5 > 1 (true) Parent(5) is 3; A[3] < 90 ➔ 66 < 90 ➔ A[5] = 76; i = 3;



i = 3; while i > 1 and A[Parent(i)] < v do {A[i] = A[Parent(i)] i = Parent(i)}
so 3 > 1 (true) Parent(3) is 1; A[1] < 90 ➔ **NO ACTION** (96 is NOT less than 90)

end the while loop with ther value of i at 3

A[3] = 90 --- i.e. the value is now inserted in the correct place in the heap, giving



The addition is now finished.