Solutions to CS / MCS 401 Exercise Set #5-6 (Fall 2007)

Exercise H. Assume the recurrence

$$C(n) = C(0.8n) + C(0.5n) + C(0.2n) + n, \quad C(1) = 1$$

has a solution

$$C(n) = an^b + cn + d$$

for some real numbers a, b, c, and d. We ignore the fact that $0.2n$, $0.5n$, and $0.8n$ are not in general integers. Substituting the proposed solution into the recurrence, we obtain

$$\begin{aligned} an^b + cn + d \;=\; & a(0.2n)^b + c(0.2n) + d + \\ & a(0.5n)^b + c(0.5n) + d + \\ & a(0.8n)^b + c(0.8n) + d + \\ & + n \end{aligned}$$

or

$$a(1 - 0.2^b - 0.5^b - 0.8^b)n^b + (c - 0.2c - 0.5c - 0.8c - 1)n - 2d = 0.$$

Since this must hold for all n, the coefficients of n^b, n, and 1 each must equal 0.

$$a(1 - 0.2^b - 0.5^b - 0.8^b) = 0 \;\Rightarrow\; 1 - 0.2^b - 0.5^b - 0.8^b = 0 \quad (\text{since } a \neq 0),$$

$$c - 0.2c - 0.5c - 0.8c - 1 = 0 \;\Rightarrow\; c = -2,$$

$$2d = 0 \;\Rightarrow\; d = 0.$$

Now the condition $C(1) = 1$ implies $a + c + d = 1$, which together with the values of c and d above gives $a = 3$.

Let $f(x) = 1 - 0.2^x - 0.5^x - 0.8^x$. b is a root of $f(x)$. $f(x)$ is clearly an increasing function of x, since each of 0.2^x, 0.5^x, and 0.8^x decrease as x increases. $f(1) = -0.5$ and $f(2) = 0.07$. Thus $f(x)$ has a unique root (equal to b) , which must lie between 1 and 2 (probably closer to 2), and it can be approximated by bisection or Newton's method. With an initial guess of 2, Newton's method with 4 iterations gives $b = 1.8267247$, or to two decimal places $b \approx 1.83$.

Note: The complete solution is $C(n) = 3\,n^{1.83} - 2n$. We saw earlier that division into 3 equal-sized subproblems of total size $1.5n$, with linear divide/combine time, led to a $\Theta(n^{1.59})$ time algorithm. Here the division into three unequal-sized subproblems with the same total size raises the time to $\Theta(n^{1.83})$.

Exercise I. The inversions in the array **a** = (41, 16, 74, 33, 66, 54) are:

(41,16), (41,33), (74,33), (74,66), (74,54), (66,54).

The number of inversions is 6. Straight insertion sort would perform 6 comparisons in which it finds the elements out of order (1 for each inversion) and $n-1 = 5$ comparisons in which it finds the elements in order — a total of **11 comparisons**. Each comparison in which the elements are out of order is followed by an exchange, so there are **6 exchanges**.

Exercise J

a) Not a strict weak order. Here $(x,y) \sim (u,v)$ when

$$\textbf{not}(x < u \textbf{ and } y < v) \textbf{ and not}(u < x \textbf{ and } v < y).$$

Note $(0,0) \sim (2,-1)$ and $(2,-1) \sim (1,2)$, but $(0,0) \not\sim (1,2)$, so $\sim$ is not transitive.

b) A strict weak order. The equivalence classes consist are the ellipses centered at the origin, with semi-major axis along the X-axis, and semi-minor axis equal to half the semi-major axis.

c) A strict weak order. $(x,y) \sim (u,v)$ when $x-y = u-v$. For each real number α, there is an equivalence class consisting of those points (x,y) for which $x-y = \alpha$, or equivalently, $y = x-\alpha$. This class is the line with slope 1 and Y-intercept $-\alpha$.

d) Not a strict weak order. Note $(x,y) \sim (u,v)$ when **not**$(x < u-1)$ **and not**$(u < x-1)$. Thus $(x,y) \sim (u,v)$ when $|x-u| \le 1$. Note $(0,0) \sim (1,0)$ and $(1,0) \sim (2,0)$, but $(0,0) \not\sim (2,0)$, so $\sim$ is not transitive.

e) A strict weak order. For each integer i, there is an equivalence class consisting of $\{ r \mid r$ is a real number with $i \le r < i+1 \}$.

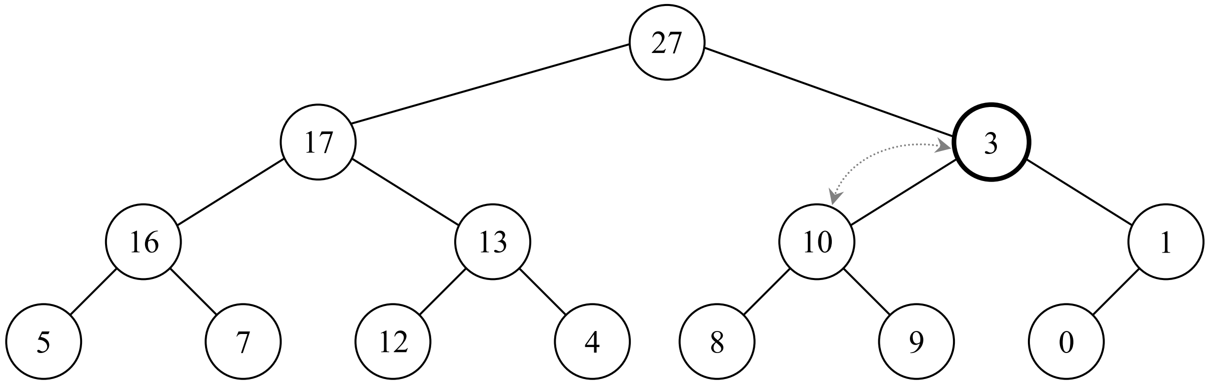
f) A strict weak order. For each real number α in $[0,1)$, there is an equivalence class consisting of $\{\alpha + i \mid i$ is an integer$\}$.

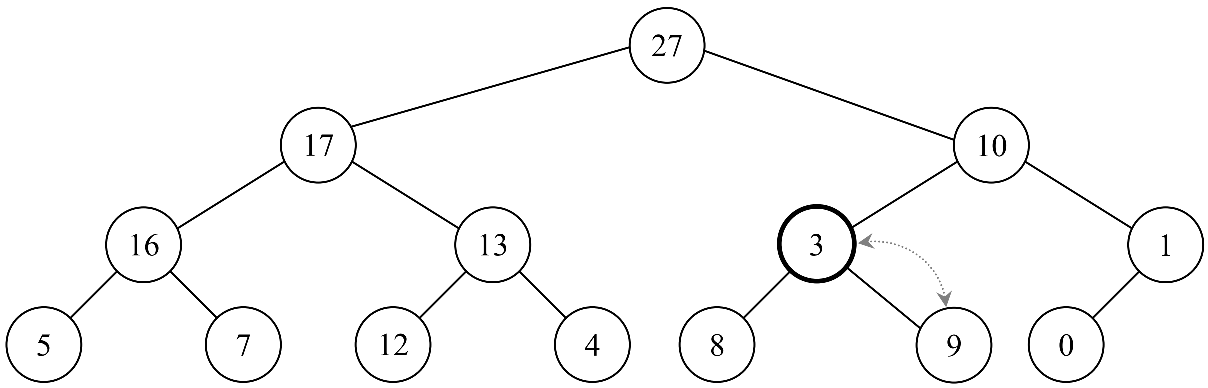
g) Not a strict weak order. Note $a \sim b$ if neither a nor b is a proper divisor of the other. So $2 \sim 3$ and $3 \sim 4$, but $2 \not\sim 4$. This means $\sim$ is not transitive.

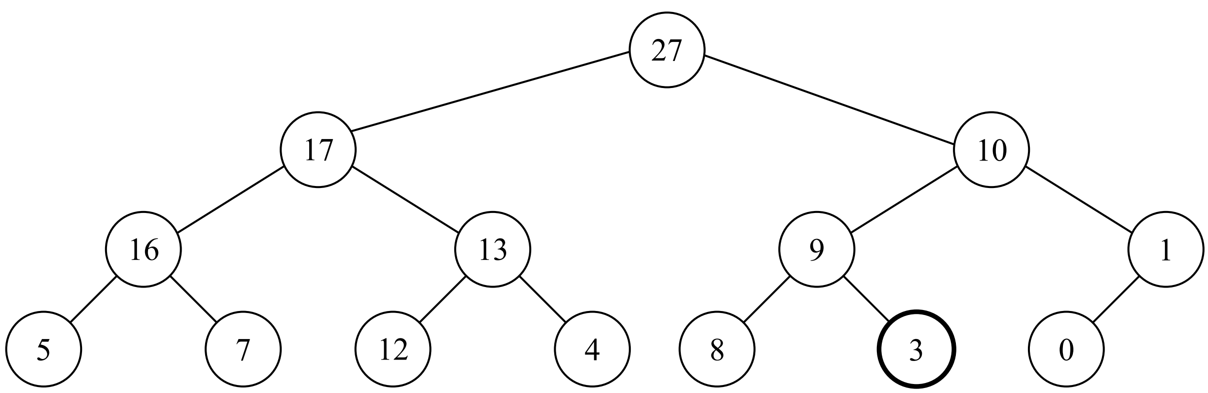
Exer 6.1-6

No. Note $a[4] = 6$ and $a[9] = 7$, so the element in node 4 fails to be greater than the element in its right child.

Exer 6.2-1

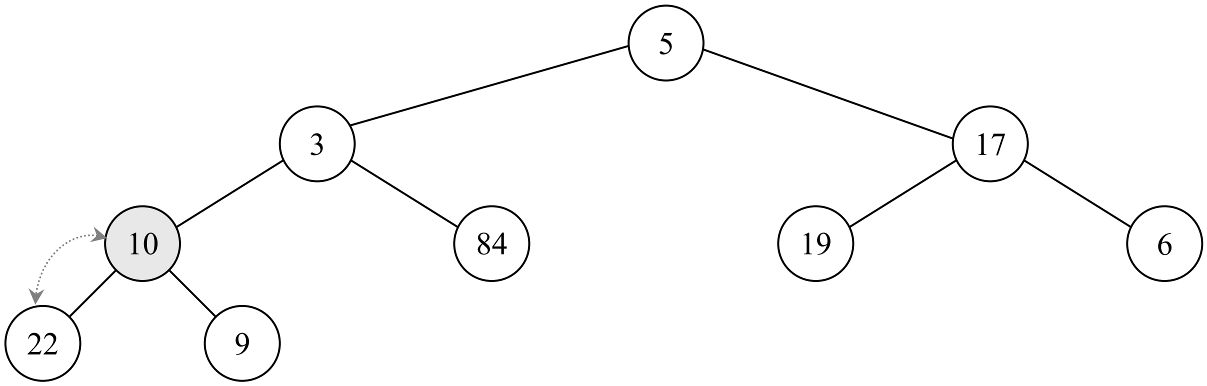

27
17
3
16
13
10
1
5
7
12
4
8
9
0

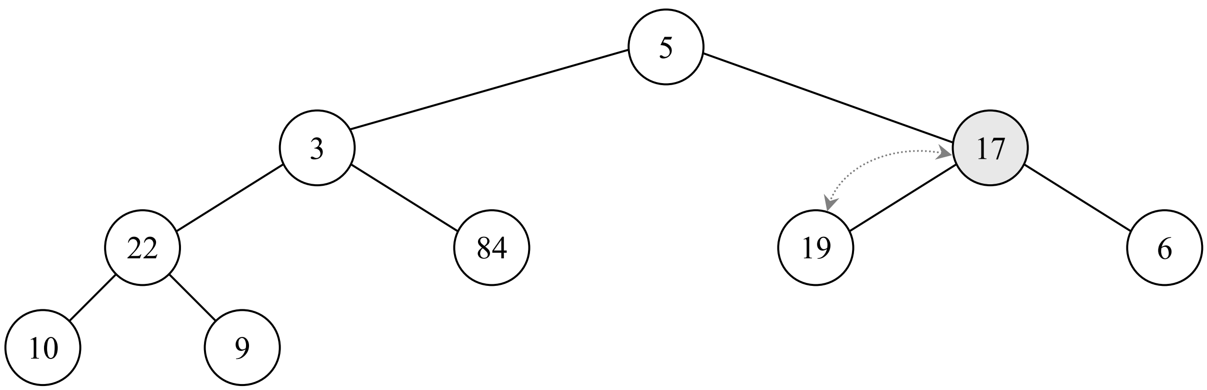

27
17
10
16
13
3
1
5
7
12
4
8
9
0

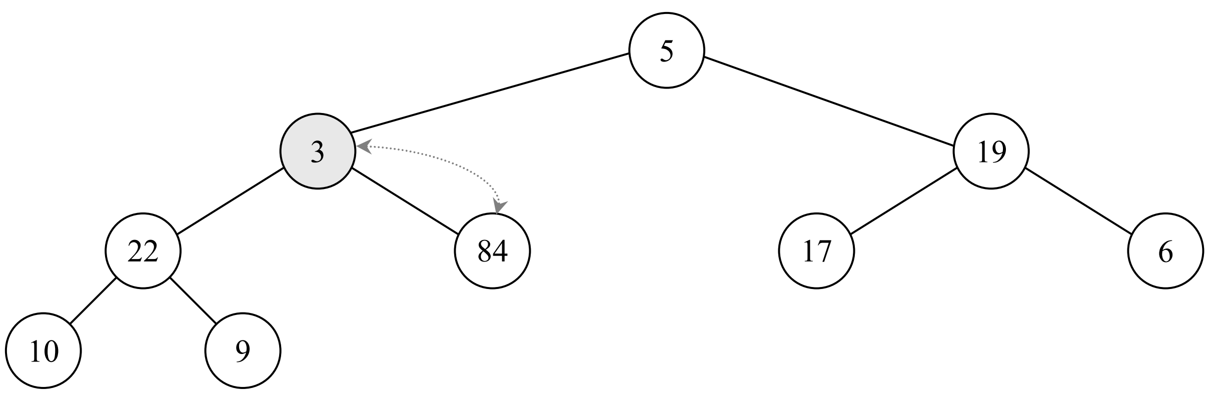

27
17
10
16
13
9
1
5
7
12
4
8
3
0

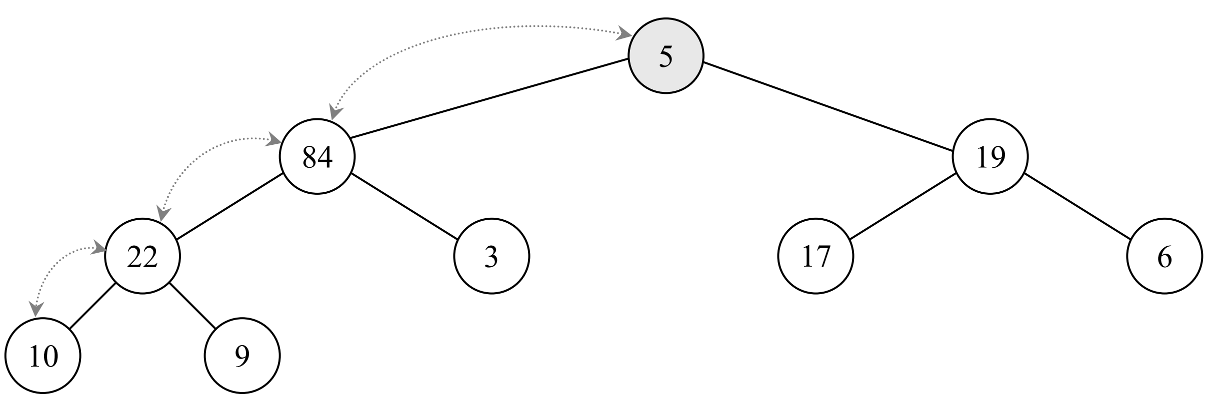
Exer 6.3-1

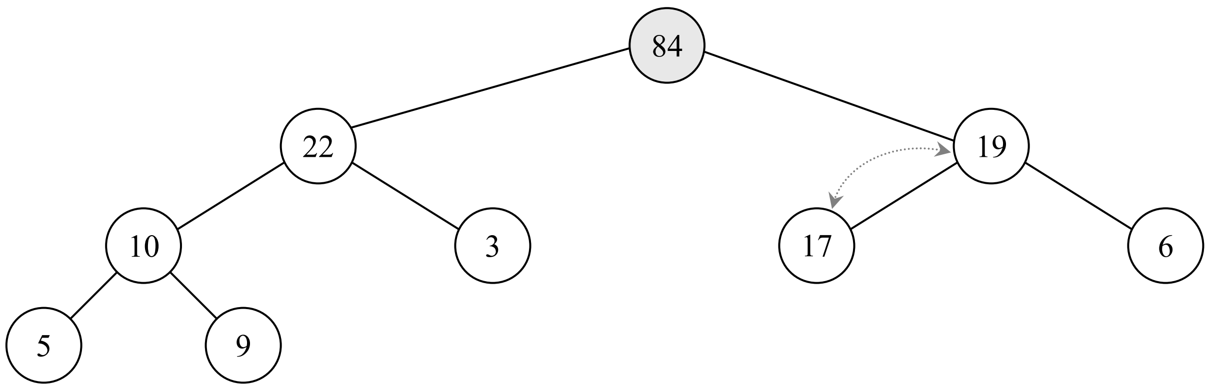
A = | 5 3 17 10 84 19 6 22 9 |











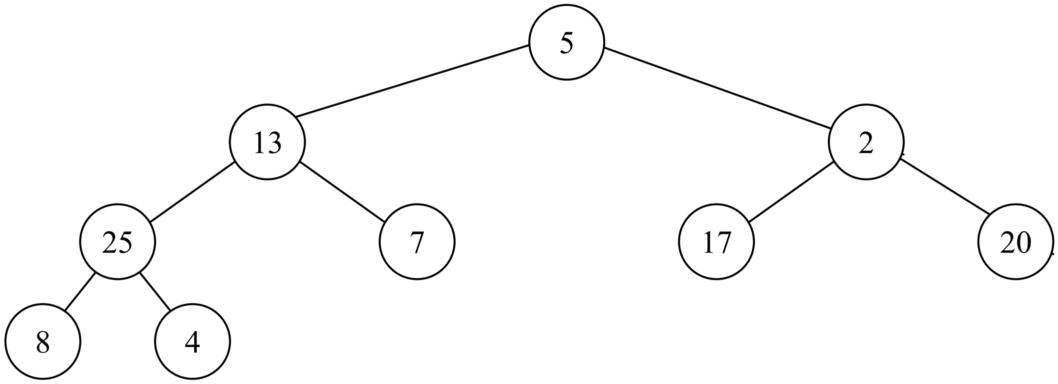
Exer 6.3-2

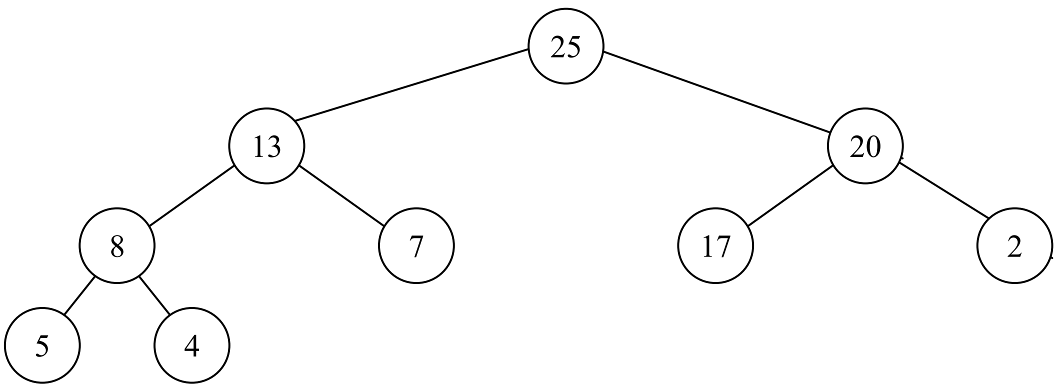
In order to apply *max-heapify*() to node i, the left and right subtrees of node i (i.e., the subtrees rooted at nodes $2i$ and $2i+1$) must already be heaps. By applying *max-heapify*() to nodes in order of decreasing i, this will always be the case.

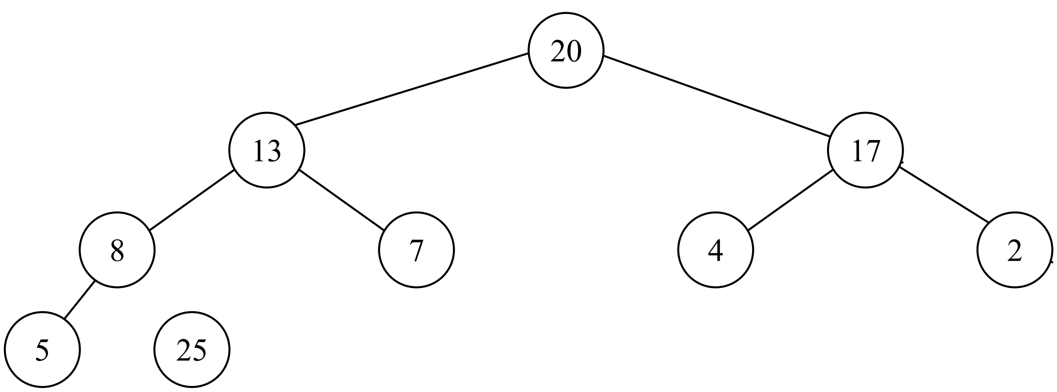
Exer 6.4-1

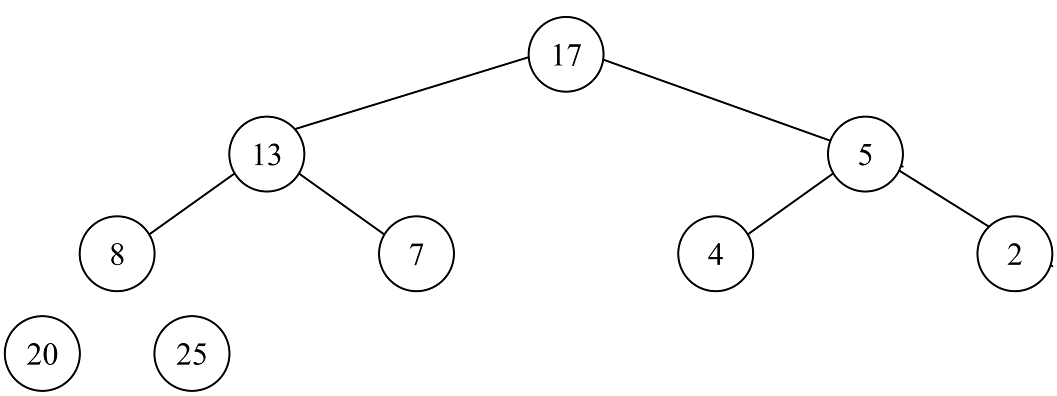
A = | 5 13 2 25 7 17 20 8 4 |

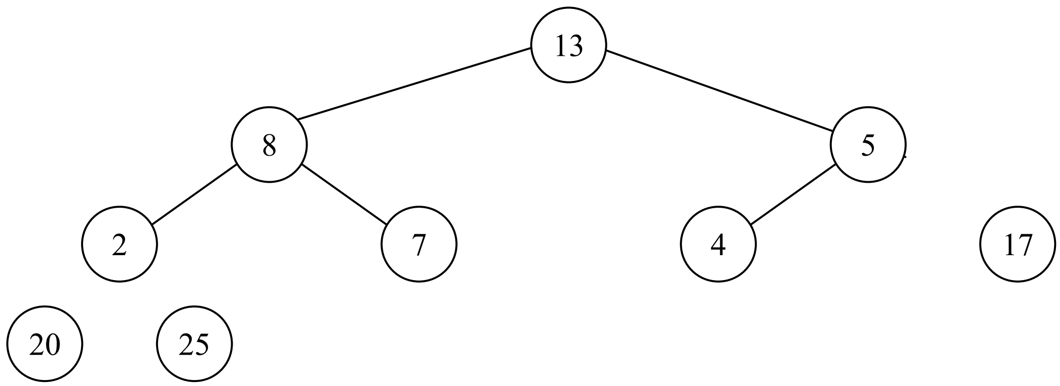
The diagrams below show the original array, the array after *build_Max_Heap*(), and the array after each step of converting the heap into a sorted array. Note you need show only the first three steps (first five diagrams).

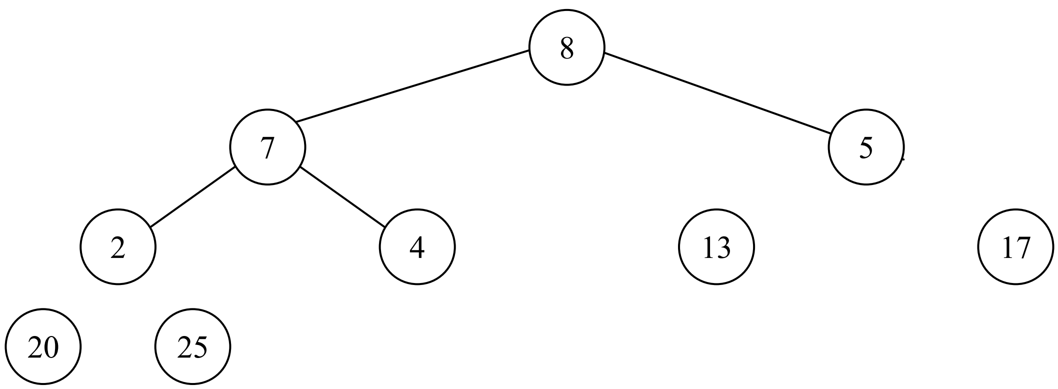

5
13
2
25
7
17
20
8
4

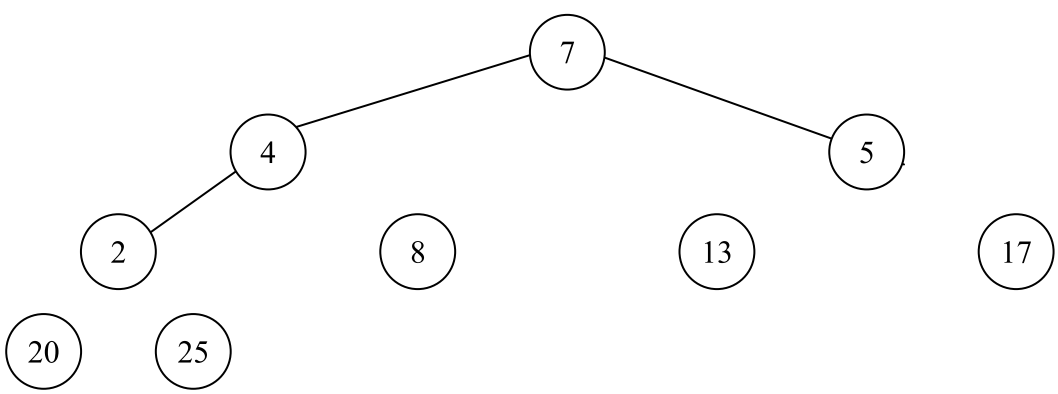

25
13
20
8
7
17
2
5
4

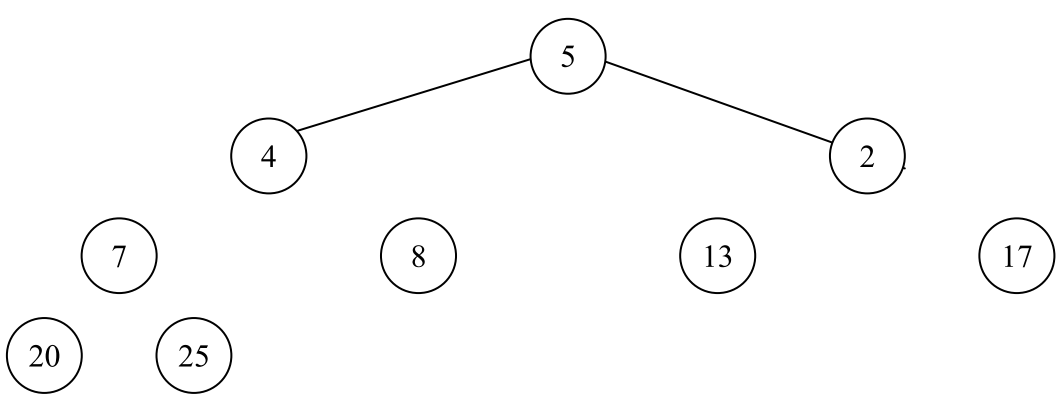

20
13
17
8
7
4
2
5
25

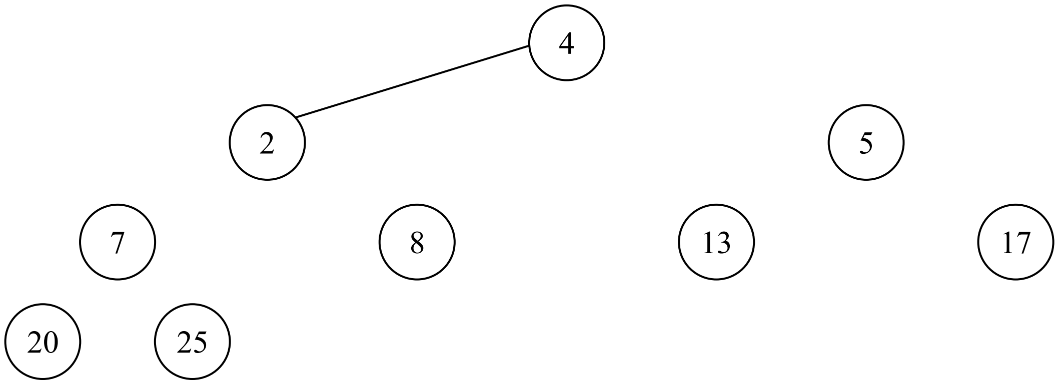

17
13
5
8
7
4
2
20
25

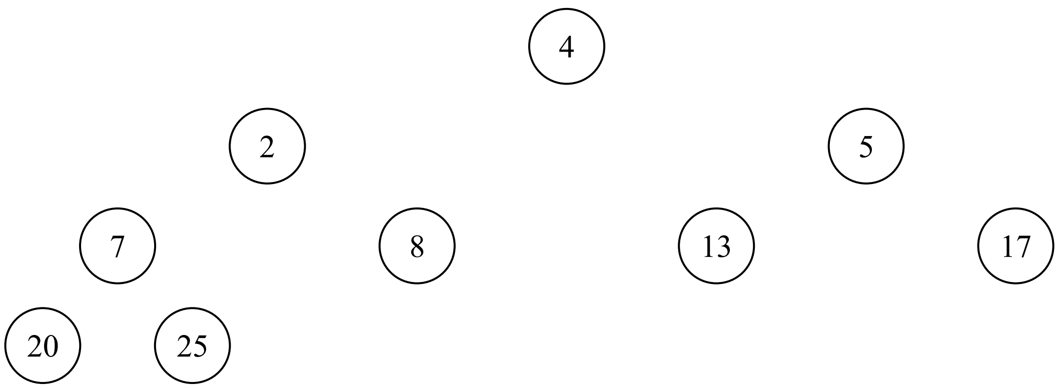

13
8
5
2
7
4
17
20
25


8
7
5
2
4
13
17
20
25


7
4
5
2
8
13
17
20
25


5
4
2
7
8
13
17
20
25





Exercise 6.5-7

```
heap-delete( A, i, n);
    swap(A[i],A[n]);
    n = n − 1;
    while ( i > 1 and A[i] > A[⌊i/2⌋] )
        swap(A[i], A[⌊i/2⌋] );
        i = ⌊i/2⌋;
    while ( 2i ≤ n )
        if ( 2i+1 ≤ n and A[2i+1] > A[2i] )
            p = 2i+1;
        else
            p = 2i;
        if ( A[i] < A[p] )
            swap(A[i], A[p]);
            i = p;
        else
            return;
```

Note: the first while loop moves A[i] up if necessary; the second moves it down if necessary. At most one of the loops will actually be traversed.