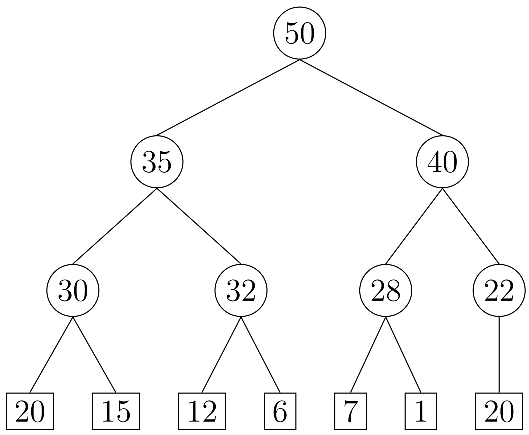
Exercise 5.1 Max-Heap operations (1 point).

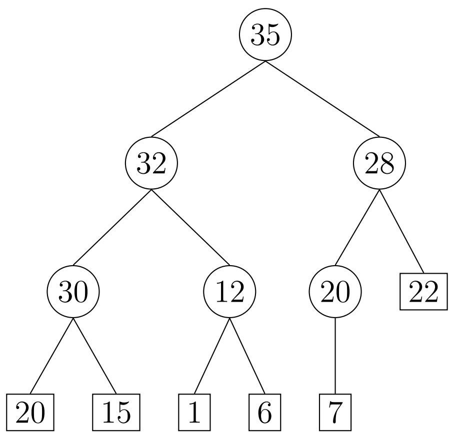
Consider the following max-heap:



Draw the max-heap after two ExtractMax operations.

Solution:

Note that ”max” was always replaced with the last (rightmost) value.



Exercise 5.3 Counting function calls in recursive functions (1 point).

For each of the following functions g, h, and k, provide an asymptotic bound in big-O notation on the number of calls to f as a function of n. You can assume that n is a power of two.

Algorithm 1

(a)

```
function g(n)
    i ← 1
    while i < n do
        f()
        i ← i + 2
    end while
    g(n/2)
    g(n/2)
    g(n/2)
end function
```

Solution:

Let $T(n)$ be a function that describes the number of calls to f, we have:

$$T(n) = \sum_{i=1}^{n/2} 1 + 3T\left(\frac{n}{2}\right) = \frac{n}{2} + 3T\left(\frac{n}{2}\right) = \frac{1}{2}n + 3T\left(\frac{n}{2}\right)$$

This allows us to use the master theorem with $a = 3$ and $b = 1$ and since $\log_2 3 > 1$ we have the case $\log_2 a > b$. This means $T \leq O(n^{\log_2 3})$

Algorithm 2

(b)

```
function h(n)
    i ← 1
    while i < n do
        f()
        i ← i + 1
    end while
    k(n)
    k(n)
end function

function k(n)
    i ← 2
    while i < n do
        f()
        i ← i^2
    end while
    h(n/2)
end function
```

Solution:

Let $H(n)$ and $K(n)$ be functions that describe the number of calls to f in h and k respectively. We begin by examining these functions independently (with simplifications).

$$H(n) = \sum_{i=1}^{n} 1 + 2K(n) = n + 2K(n)$$

and

$$K(n) = \sum_{i=1}^{\log_2(\log_2 n)} 1 + H\left(\frac{n}{2}\right) = \log_2(\log_2 n) + H\left(\frac{n}{2}\right)$$

And since K in H is being called with the same parameter n we can simply substitute.

$$H(n) = n + 2\left(\log_2(\log_2 n) + H\left(\frac{n}{2}\right)\right) = n + 2\log_2(\log_2 n) + 2H\left(\frac{n}{2}\right)$$

We can ignore the $\log(\log n)$ term as $\log(\log n) \leq O(n)$ and this allows us to use the master theorem with $a = 2$ and $b = 1$ and since $\log_2 2 = 1$ we have the case $\log_2 a = b$. This means $H \leq O(n \log n)$.

Using this information, we return to $K(n)$.

$$K(n) \leq \log_2(\log_2 n) + \left(\frac{n}{2} \cdot \log \frac{n}{2}\right) \leq O(n \log n)$$

Exercise 5.4 Bubble sort invariant (1 point).

Consider the pseudocode of the bubble sort algorithm on an integer array $A[1, \ldots, n]$:

Algorithm 3 BUBBLESORT(A)

```
for 1 ≤ j < n do
    for 1 ≤ i < n do
        if A[i] > A[i + 1] then
            t ← A[i]
            A[i] ← A[i + 1]
            A[i + 1] ← t
        end if
    end for
end for
return A
```

(a) Formulate an invariant $INV(j)$ that holds at the end of the j-th iteration of the outer for-loop.

Solution:

$INV(j)$ = After every j-th iteration, the last j elements of the array are sorted and at their correct position.

(b) Using the invariant from part (a), prove the correctness of the algorithm. Specifically, prove the following three assertions:

(1) $INV(1)$ holds.

(2) If $INV(j)$ holds, then $INV(j + 1)$ holds (for all $1 \leq j < n$).

(3) $INV(n)$ implies that BUBBLESORT(A) correctly sorts the array A.

Solution:

(1) $INV(1)$ holds because the first iteration will perform swaps every time a value with index $i + 1$ is smaller than the one with index i. This "takes" the largest value in A to the end of the array (index n) and thus the last element is sorted and in its correct position.

(2) Suppose $INV(j)$ holds for some j, with j an index of A. This means that the last j items are already sorted before the $(j + 1)$-th iteration.

The $(j+1)$-th iteration will, analogously to the description in item 1, take the largest value of the subarray $A[1, ..., n - j]$ to index $n - j$.

Since we assumed $INV(j)$ was true, we know the last j items are already sorted, and, by placing the largest remaining item at position $n - j$ (right before the already-sorted block), it follows that the last $j + 1$ elements are all sorted and in their correct positions. This means the invariant holds for $(j + 1)$

(3) $INV(n)$ claims that the last n elements are sorted and in the correct position, which are all elements of the array. This implies A is correctly sorted.