1. Illustrate the operation that removes and returns the maximum element on the max-heap \( A = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1] \).

\[ [1, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2] \]
and index 1.

This part is best to see in the tree representation. The bubble steps according to \text{maxHeapify} in the array representation, just as a help to check your solution:

\[
\begin{align*}
13 & \rightarrow 12 \\
12 & \rightarrow 9 \\
12 & \rightarrow 6
\end{align*}
\]

(1 correct solution needs more informative pictures.)

2. Let \( A = [15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1] \) be a max-heap. Illustrate the operation of adding 10.

We apply \text{insert} on \( A \) and 10. First \( A.\text{heap} - \text{size} := 12 + 1 = 13 \). Following the book, we have first \( A[A.\text{heap} - \text{size}] = A.13 := -\infty \). Then we change the label \(-\infty\) into 10: \( A[A.\text{heap} - \text{size}] = A[13] := 10 \). Then 10 ‘bubbles upwards’ according to lines 4–6 of the procedure \text{heapIncreaseKey}, where initially \( i = 13 \) and \( \text{key} = 10 \). This part is best to see in the tree representation. Apply the procedures as for example given in the book. A complete solution needs more informative than only the arrays; pictures are more informative here.

\[
\begin{align*}
[15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1, 10] \\
[15, 13, 9, 5, 12, 10, 7, 4, 0, 6, 2, 1, 8] \\
[15, 13, 10, 5, 12, 9, 7, 4, 0, 6, 2, 1, 8]
\end{align*}
\]

3. Apply quicksort to the following input:

\[ 8 \ 5 \ 2 \ 7 \ 1 \ 3 \ 4 \ 6 \]

The recursion tree for the application of quicksort:

\[
\begin{array}{c}
[8, 5, 2, 7, 1, 3, 4, 6] \\
/ \\
\end{array}
\]
We give two applications of partition. Take way in which the application of an algorithm is given in the book is excellent (better than what is written below, use the thing below mainly to check your answers.)

The pivot is in boldface. In the first application not much happens because the pivot is the largest element. At indices 1, . . . , 6 the contents of the array at that index is swapped with itself.

$\begin{array} \{(i), 8(j), 5, 2, 7, 1, 3, 4, 6, 9\} & j = 0 \\
\{(i), 8(j), 5, 2, 7, 1, 3, 4, 6, 9\} & j = 0 \\
\{(i), 5(j), 7, 1, 3, 4, 6, 9\} & j = 1 \\
\{(i), 5(j), 7, 1, 3, 4, 6, 9\} & j = 1 \\
\{(5(i), 7(j), 1, 3, 4, 6, 9\} & j = 2 \\
\{(5, 7(i), 1(j), 3, 4, 6, 9\} & j = 3 \\
\{(5, 7, 1(i), 3(j), 4, 6, 9\} & j = 4 \\
\{(5, 7, 1, 3(i), 4(j), 6, 9\} & j = 5 \\
\{(5, 7, 1, 3, 4(i), 6(j), 9\} & j = 6 \\
\{(5, 7, 1, 3, 4, 6(i), 9(j))\} & j = 7 \\
\{(5, 7, 1, 3, 4, 6, 9\} &
\end{array}$

Next we consider the array $[8, 5, 2, 7, 1, 3, 4, 6]$.
4. Partition can be implemented in many different ways.
   Consider for example a variation where we use $A[p]$ as pivot instead of $A[r]$.
   You may consider even two different solutions: one where you only manipulate with the pivot initially, and one where you consider a ‘mirrored version’ of partition.
   (No solution yet.)

5. Here is an example of an alternative implementation of partition. (Pseudo-code skipped.) Apply this algorithm to the input we already used above:

   $\begin{array}{cccccccc}
   8 & 5 & 2 & 7 & 1 & 3 & 4 & 6 \\
   \end{array}$

   or, maybe better, to

   $\begin{array}{cccccccc}
   3 & 6 & 1 & 7 & 8 & 2 & 5 & 4 & 6 \\
   \end{array}$

   (No solution yet.)

6. We use quicksort and partition as given in the book.
   Give an example of a best-case input of length 7 for quicksort. Also draw the recursion tree.
   An example of a best-case input array: $[1, 3, 2, 6, 5, 7, 4]$.
   (Recursion tree omitted here.)

7. We use quicksort and partition as given in the book.
   Give an example of a worst-case input for quicksort. Also draw the recursion tree.
   An example of a worst-case input array: $[7, 6, 5, 4, 3, 2, 1]$.
   If the pivot is in every call of partition the smallest or largest element, the recursion tree gets maximal depth. If the pivot is in every call of partition the largest element, then the test $A[j] < x$ succeeds every time, and we perform a maximum number of steps in the application of partition.
   So in terms of number of steps a decreasing input array is even worse than an increasing input array. However, for the performance in terms of $O$ they are equally bad.
   (Recursion tree omitted here.)

8. Adapt quicksort (as given in the book) so that it sorts sequences in non-increasing (instead of non-decreasing) order.
We change in the algorithm for permutation the test $A[j] \leq x$ into $A[j] > x$.

**Algorithm** partition$(A, p, r)$:

$$
x := A[r]\\
i := p - 1\\
\text{for } j = p \text{ to } r - 1 \text{ do}\\
\quad \text{if } A[j] > x \text{ then}\\
\quad \quad i := i + 1\\
\quad \quad \text{exchange } A[i] \text{ with } A[j]\\
\quad \text{exchange } A[i + 1] \text{ with } A[r]\\
\text{return } i + 1
$$

9. Explain why the running time of partition (as given in the book) is in $O(n)$.

(Remark: even in $\Theta(n)$.)

**Algorithm** partition$(A, p, r)$:

$$
x := A[r]\\
\text{this assignment is carried out 1 time}\\
i := p - 1\\
\text{this assignment is carried out 1 time}\\
\text{for } j = p \text{ to } r - 1 \text{ do}\\
\quad \text{the assignment for } j \text{ is carried out } r - p + 1 \text{ times}\\
\quad \text{we enter the loop } r - p \text{ times}\\
\quad \text{if } A[j] \leq x \text{ then}\\
\quad \quad \text{for a fixed } j \text{ this test is done 1 time}\\
\quad \quad \quad i := i + 1\\
\quad \quad \text{for a fixed } j \text{ this assignment is done 1 time in the worst case}\\
\quad \quad \quad \text{exchange } A[i] \text{ with } A[j]\\
\quad \quad \text{for a fixed } j \text{ this swap is done 1 time in the worst case}\\
\quad \quad \quad \text{exchange } A[i + 1] \text{ with } A[r]\\
\quad \text{this exchange is done 1 time}\\
\text{return } i + 1
$$

So the worst-case time complexity is determined by the cost of the for-loop which is given in the worst case by $\sum_{j=1}^{n-1} 1 = n - 1$. So partition is in $O(n)$.

10. Determine for each of the following functions $f$ and $g$ whether $f \in O(g)$ and/or $g \in O(f)$.
(a) \( f(n) = 5n^2 + 3n + 7 \) and \( g(n) = n^3 \).

\( f \in \mathcal{O}(g) \) but \( g \notin \mathcal{O}(f) \).

(b) \( f(n) = \sum_{i=1}^{n} i \) and \( g(n) = n^2 \).

We have \( \sum_{i=1}^{n} i = \frac{1}{2}n(n + 1) \). Hence \( f \in \mathcal{O}(g) \) and also \( g \in \mathcal{O}(f) \).

(c) \( f(n) = n^n \) and \( g(n) = n! \).

We have \( n! \in \mathcal{O}(n^n) \). We do not have \( n^n \in \mathcal{O}(n!) \).

(d) \( f(n) = n \log_2 n \) and \( g(n) = n\sqrt{n} \).

We have \( n \log n \in \mathcal{O}(n\sqrt{n}) \) but not \( n\sqrt{n} \in \mathcal{O}(n \log n) \). (Add explanation.)