1. [15] To prove that \( f(n) \in \mathcal{O}(g(n)) \), or equivalently \( g(n) \in \Omega(f(n)) \), one has to exhibit two constants, \( c \) and \( n_0 \) such that \( f(n) \leq c \cdot g(n) \) for all \( n \geq n_0 \). For part a and b below answer “yes” or “no” and provide a proof of your answer.

(a) For \( g_1, g_2 \in \mathcal{O}(g(n)) \), \( f(n) = g_1(n) + g_2(n) \in \mathcal{O}(g(n)) \). Answer:

(b) \( \min\{f(n), g(n)\} \in \Theta(f(n) + g(n)) \) Answer:

(c) What is the time complexity of the following code fragment? Give tight upper and lower bounds.

\[
\begin{align*}
m &:= 0; \\
for (\text{int } i = 0; i < n; i++) \\
  &\quad \text{for (int } j = 0; j < n-i; j++) \\
  &\quad\quad \text{for (int } k = 0; k < i; j++) \\
  &\quad\quad\quad \text{m++;} \\
\end{align*}
\]

\( T(n) \in \)
2. [15] Let \( T \) be a (min) heap storing \( n \) keys. Give an efficient algorithm for reporting all the keys in \( T \) that are smaller than or equal to a given query key \( x \) (which is not necessarily in \( T \)). The keys do not need to be reported in sorted order. Ideally, your algorithm should run in \( O(k) \) time, where \( k \) is the number of keys reported.
3. [15] Regarding red-black trees

(a) From the RB tree of figure 1 (dotted lines mean red), delete 30.

(b) Again from the RB tree of figure 1 (the original one, before part (a)), delete 1.
Figure 1: red-black tree for question 3
4. [15] Put the following values into an initially empty (2,4)-tree:

\[15, 6, 12, 30, 22, 17, 10, 5, 9, 14, 23, 1, 17, 29, 18, 24, 7, 26, 35, 33, 27.\]
5. Regarding splay trees
   (a) Insert into an initial empty splay tree the values
       1, 5, 8, 7, 3, 2.
   (b) then delete 3
6. [15] The QuickSort algorithm in the text uses the last element of the input sequence as the pivot in a partition subroutine.

(a) Use the decision tree to model operation of a sorting method on three \((n = 3)\) elements as a comparison-based sorting algorithm.

(b) What is the running time QuickSort on an already sorted sequence?

(c) Show that the best-case time complexity of QuickSort is \(\Theta(n \log n)\).
7. [15] Give a linear \( O(n) \) time algorithm sorting \( n \) values in range 0..\( n^3 - 1 \). (Hint: represent a value \( x \) as \((i, j, k)\) where \( x = i \cdot n^2 + j \cdot n + k \).)