1. Let the array \( A = (81, 56, 41, 88, 8, 49, 57, 58, 35) \). Using Figures 6.3 and 6.4 as models, illustrate the operation of Heapsort on \( A \).

2. Consider the following alternative to Build-Max-Heap.

   \[
   \text{Recursive-Build-Max-Heap}(A, i) \\
   \text{if } \text{Left}(i) \leq A.\text{heap-size} \\
   \quad \text{then } \text{Recursive-Build-Max-Heap}(A, \text{Left}(i)) \\
   \text{if } \text{Right}(i) \leq A.\text{heap-size} \\
   \quad \text{then } \text{Recursive-Build-Max-Heap}(A, \text{Right}(i)) \\
   \text{Max-Heapify}(A, i)
   \]

   This procedure would be invoked by performing:

   \[
   A.\text{heap-size} \leftarrow A.\text{length} \\
   \text{Recursive-Build-Max-Heap}(A, 1)
   \]

   Justify the correctness of \text{Recursive-Build-Max-Heap}.

3. Describe the running time of \text{Recursive-Build-Max-Heap}(A, 1) by justifying and solving a recurrence.

4. Using Figure 7.1 as a model, illustrate the operation of Partition on the array: \( A = (81, 35, 41, 88, 8, 49, 57, 58, 56) \)

5. In exercises 5 and 6, we analyze another way of bounding Quicksort’s average running time. The goal here is to determine how many times (on average) that \text{Randomized-Partition} must be called on an array of size \( n \) before all the partitions have size \( n/2 \) or less. If this value is a constant, that would imply that \( \Theta(n) \) time is needed on average to divide a problem of size \( n \) into subproblems of size at most \( n/2 \).

   Usually, one call to \text{Randomized-Partition} will result in one partition with size greater than \( n/2 \), more than half the elements. We will model this situation by choosing a random real number \( r_1 \) between 1/2 and 1. In the next iteration, we choose another random real number \( r_2 \) between 1/2 and 1 and multiply \( r_2 \) times \( r_1 \). We keep doing this until the result is less than 1/2. Here is an algorithm that formalizes this process.

   \[
   \text{Reduce-To-Half}() \\
   x \leftarrow 1 \\
   \text{while } x > 1/2 \\
   \quad \text{do } r \leftarrow \text{a random real number } \geq 1/2 \text{ and } \leq 1 \\
   \quad \quad x \leftarrow r \times x
   \]
Determine the probability \( p_1 \) that \( x \leq 3/4 \) after exactly one iteration. Briefly justify your answer.

6. A probability of \( p_1 \) from the previous exercise implies that \( 1/p_1 \) is an upper bound on the average number of iterations to reduce \( x \) from \( x = 1 \) to \( x \leq 3/4 \).

Prove that the average number of iterations performed by \textsc{Reduce-To-Half} is less than or equal to 5. Hint: Bound the average number of iterations to reduce \( x \) from \( x = 3/4 \) to \( x \leq 1/2 \).

7. In exercises 7–9, we want to code and analyze an efficient implementation that allows insertion and deletion of integers between 1 and \( k \) and answer queries about how many numbers are in the range \([a..b]\). We want each of these three operations to take \( O(\log k) \) time per operation.

For example, if \( k = 7 \), then after inserting 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 3, 5, then a query of \([2..4]\) should result in 6.

We will use two arrays \( A \) and \( B \) of length \( k \) to keep track of the counts. The array \( A \) will keep a count of each number, e.g., \( A = \langle 2, 2, 3, 1, 2, 1, 1 \rangle \) for the above example. The other array \( B \) will keep track of various ranges of values.

The ranges will be based on binary search. For \( k = 7 \), the initial range will be \([1..7]\). Because the midpoint is 4, then \( B[4] \) will store the number of values in this range. One iteration of binary search would result in searching either \([1..3]\) or \([5..7]\). The number of values in these ranges would be stored in \( B[2] \) and \( B[6] \) respectively. So, the insertion procedure is:

\[
\text{INSERT}(v) \\
lo \leftarrow 1 \\
hi \leftarrow k \\
\text{while } lo \leq hi \\
\quad \text{do } mid \leftarrow \lfloor (lo + hi)/2 \rfloor \\
\quad \quad B[mid] \leftarrow B[mid] + 1 \\
\quad \text{if } v < mid \\
\quad \quad \text{then } hi \leftarrow mid - 1 \\
\quad \text{else if } v > mid \\
\quad \quad \text{then } lo \leftarrow mid + 1 \\
\quad \text{else return}
\]

What would be the differences for the \textsc{Delete} operation?

What is the final value of \( B \) after inserting 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 3, 5?

8. Use mathematical induction to prove that the while loop in \textsc{Insert} runs at most \( O(\log k) \) iterations.

9. Provide pseudocode for a \textsc{Query} operation that runs in \( O(\log k) \) time.
10. Consider the problem of finding the range of values from order statistic \( i \) to order statistic \( j \) in an array. Provide and justify a \( \Theta(n + n \lg(j - i + 1)) \) worst-case algorithm to find and sort this range of values. You may assume that all the values are distinct.

11. (Extra Credit) Prove that the average number of iterations performed by \textsc{Reduce-To-Half} in Exercise 5 is less than or equal to 4.

12. (Extra Credit) Prove that the average number of iterations performed by \textsc{Reduce-To-Half} in Exercise 5 is equal to 3.