1. (a) Present a procedure \texttt{shiftUp}(A, i) that \textit{lifts} the key \(A[i]\) at its proper level in 2-heap \(A[1..i]\).

(b) Simulate the transformation of the array \(A[1..8] = [3, 1, 4, 1, 5, 9, 2, 6]\) into a 2-heap by lifting each key \(A[2], A[3], \ldots, A[8]\) at its proper level.

(c) Simulate the transformation of the same array into a 2-heap by shifting \textit{down} the keys \(A[4], A[3], A[2], \text{and } A[1]\).

2. We can find the \(k\) largest items of an array \(A[1..n]\)

   (a) in time \(O(kn)\) using "brute-force",

   (b) in time \(O(n \log n)\) applying sorting, and

   (c) in time \(O(n + k \log n)\) by building a heap.

Sketch the algorithms and justify their complexity.

(d) Could we find the \(k\) largest items in time \(O(n)\), or \(O(k)\)? How, or why not?

3. Prim’s algorithm can be implemented to run in time \(O(e \log e)\), where \(e\) is the number of edges in the graph. Present the algorithm, and analyze its complexity. Assume that the graph is given as adjacency lists, which give for each node a list of weighted edges emanating from the node. (Hint: store the edges that emanate from nodes covered by the tree in a priority queue.)

4. (a) At the lecture we discussed a dynamic programming algorithm for computing how many rectilinear paths lead from square \((0,0)\) to square \((n,n)\). Simulate how the algorithm computes the value \(L(5,5)\).

(b) One can observe a regularity that allows \(L(n,n)\) to be computed more efficiently. Based on this, present a more efficient algorithm for the task.

Let \(C(n)\) be the number of possible orders for performing the matrix multiplications \(M_1 \times \cdots \times M_n\). It is determined by these recurrences: \(C(1) = 1\) and \(C(n) = \sum_{k=1}^{n-1} C(k)C(n-k)\), when \(n \geq 2\). (Here’s why: The ‘\(\times\)’ performed last can be between matrices \(M_k\) and \(M_{k+1}\) for each \(k = 1, \ldots, n-1\). The sequence \(M_1 \times \cdots \times M_k\) can be evaluated in \(C(k)\) different orders, and \(M_{k+1} \times \cdots \times M_n\) in \(C(n-k)\) different orders.)

5. Present a recursive algorithm based directly on the above recurrences, to compute \(C(n)\). Estimate its complexity. (Hint: Write down the recurrence for \(T(n)\), simplify it, and examine the difference \(T(n) - T(n-1)\).)

6. Present a dynamic programming algorithm for computing \(C(n)\). Analyze its complexity.