Problem Set 3

1. **Algorithm 1:** `maxSessions`(A)

Input: A set of sessions, $A = \{ \langle S_i, D_i, F_i \rangle \mid 1 \leq i \leq n \}$

Output: $B \subseteq A \mid B$ has max non-overlapping sessions

$A \leftarrow A$ sorted by F_i in ascending order

$\mathsf{last} \leftarrow F_1$

$B \leftarrow \emptyset$

for $i \leftarrow 2$ to n **do**
  if $S_i \geq \mathsf{last}$ **then**
    $\mathsf{last} \leftarrow F_i$
    $B \leftarrow B \cup \{ \langle S_i, D_i, F_i \rangle \}$

return B

Proposition 1.1. *The running time of* `maxSessions`(A) *is* $O(n \log n)$.

Proof. Sorting the input takes $O(n \log n)$ time, while the loop takes $O(n)$. Thus, the overall time complexity is dominated by the sorting step. □

Proposition 1.2. `maxSessions`(A) *is optimal.*

Proof. Assume, for contradiction,

$$\texttt{maxSessions}(A) = B = \{ \langle S_{b_i}, D_{b_i}, F_{b_i} \rangle \mid 1 \leq i \leq |B| \} \text{ is suboptimal.}$$

$$B \text{ is suboptimal} \implies \exists C = \{ \langle S_{c_i}, D_{c_i}, F_{c_i} \rangle \mid 1 \leq i \leq |C| \} \mid C \text{ is optimal.}$$

$$(B \text{ is suboptimal}) \wedge (C \text{ is optimal}) \implies |C| > |B|.$$

Without loss of generality, assume B and C are sorted by F in ascending order. Let $i = \min\{ 1 \leq j \leq |B| \mid b_j \neq c_j \}$. Since `maxSessions`$(A)$ greedily selects the earliest ending session, we have

$$\forall (1 \leq j \leq |B|),\ F_{b_j} \leq F_{c_j}.$$

Thus, we can set $c_i \leftarrow b_i$ and still have an optimal solution. We can repeat this process until $i = |B|$, which yields a solution with $|B|$ sessions.

$$|B| = |C| \implies B \text{ is optimal.}$$

Contradiction! □

2. Proposition. *If $G = (V, E, w : E \mapsto \mathbb{R}^+)$ is a connected undirected graph with nonnegative costs of edges and*

$$e = (u, v) \in E \mid w(e) = \min_{\forall e' \in E} w(e')$$

is its edge with minimum cost, then

$$\forall \text{ MST } T = (V, E_{\text{T}}, w_{\text{T}} : E \mapsto \mathbb{R}^+),\ e \in E_{\text{T}}.$$

Proof. Assume, for contradiction, $\exists$ MST $T \mid e \notin E_{\text{T}}$.

$$T \text{ is a tree} \implies \exists \text{ unique path between } u \text{ and } v.$$

Let E_p be the set of edges along this path.

$$w(e) = \min_{\forall e' \in E} w(e') \implies \exists e_p \in E_p \mid w(e_p) > w(e).$$

Let $E' = (E_{\text{T}} \cup \{ e \}) \setminus \{ e_p \}$ and

$$w'_{\text{T}}(e') = \begin{cases} w(e), & \text{if } e' = e, \\ w_{\text{T}}(e'), & \text{otherwise.} \end{cases}$$

We can now construct

$$T' = (V, E'_{\text{T}}, w'_{\text{T}}) \mid \sum_{\forall e' \in E'_{\text{T}}} w'_{\text{T}}(e') < \sum_{\forall e' \in E_{\text{T}}} w_{\text{T}}(e') \implies \neg(T \text{ is a MST}).$$

Contradiction! □

3. Algorithm 1: `minTies(`L`)`

Input: A set of ropes, $L = \{ \ell_i \mid 1 \leq i \leq n \}$

Output: An optimal full binary tree, where each leaf is a rope, and each internal node is a tie.

$V \leftarrow L$

$E \leftarrow \emptyset$

$H \leftarrow$ priority queue (binary min-heap) of integers, ordered by L

for $i \leftarrow 1$ to n **do**
  Insert ℓ_i into H

for $k \leftarrow 1$ to $n - 1$ **do**
  $i \leftarrow$ minimum element popped from H
  $j \leftarrow$ minimum element popped from H
  $V \leftarrow V \cup \{ i + j \}$
  $E \leftarrow E \cup \{ (i, i + j), (j, i + j) \}$
  Insert $i + j$ into H

return (V, E)

Proposition 3.1. `minTies(`L`)` *is optimal.*

Proof. Similar to the Huffman coding algorithm, since the cost of the tree is the sum of the lengths of all ropes and cost of internal nodes, except the root, *the two ropes with the smallest length must be at the bottom of the optimal tree,* as children of the lowest internal node (this internal node has two children since the tree is full). Otherwise, swapping these two ropes with whatever is lowest in the tree would improve the tieing. □

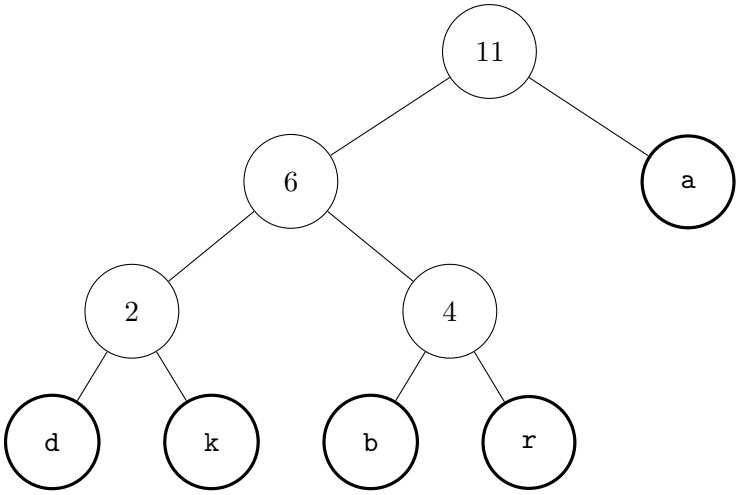
Proposition 3.2. *The running time of* `minTies(`L`)` *is* $O(n \log n)$.

Proof. In the worst case, we perform $n - 1$ heap operations, each of which take $O(\log n)$ time. Therefore, the worst-case running time of the algorithm is $O(n \log n)$. □

4. We take a closer look at our particular sequence and find that the four symbols are not equally abundant:

Symbol	Frequency
a	5
b	2
d	1
k	1
r	2

Any prefix-free encoding can be represented by a full binary tree—that is, a binary tree in which every node has either zero or two childre—where the symbols are at the leaves, and where each codeword is generated by a path from root to leaf, interpreting left as `0` and right as `1`. The following is an example of such an encoding for the five symbols:



Decoding is unique: a string of bits is decrypted by starting at the root, reading the string from left to right to move downward, and, whenever a leaf is reached, outputting the corresponding symbol and returning to the root:

Symbol	Codeword
a	1
b	010
d	000
k	001
r	011

5.