Little Johnny is extremely fond of watching television. His parents are off for work for the period \([S, F]\), and he wants to make full use of this time by watching as much television as possible: in fact, he wants to watch TV non-stop the entire period \([S, F]\). He has a list of his favorite \(n\) TV shows (on different channels), where the \(i\)-th show runs for the time period \([s_i, f_i]\), so that the union of \([s_i, f_i]\) fully covers the entire time period \([S, F]\) when his parents are away.

(a) (10 points) Little Johnny doesn’t mind to switch to the show already running, but is very lazy to switch the TV channels, and so he wants to find the smallest set of TV shows that he can watch, and still stay occupied for the entire period \([S, F]\). Design an efficient \(O(n \log n)\) greedy algorithm to help Little Johny. Do not forget to carefully argue the correctness of your algorithm, using either the “Greedy Always Stays Ahead” or the “Local Swap” argument.

**Solution:** INSERT YOUR SOLUTION HERE

(b) (4 points). Assume now that Little Johnny will only watch shows from the beginning till end (except show starting before \(S\) or ending after \(F\)), but now he fetches another TV from the adjacent room, so that he can potentially watch up to two shows at a time. Can you find a strategy that will give the smallest set of TV shows that he can watch on the two TVs, so that at any time throughout the interval \([S, F]\) he watches at least one (and at most two) shows. (**Hint:** Try to examine your algorithm in part (a).)

**Solution:** INSERT YOUR SOLUTION HERE
Remember the efficient implementation of Huffman codes in time $O(n \log n)$. Maintain a priority queue $Q$ of all current frequencies (initially the $n$ given frequencies), then repeatedly extract min twice, and insert the sum of the extracted frequencies. Here we will design a more efficient implementation assuming that the initial frequencies are already sorted: $f_1 \leq f_2 \leq \ldots \leq f_n$. We proceed in a sequence of steps outlined below.

(a) Recall, a priority queue $Q$ should support (at least) the following operations: $Q \leftarrow \text{INITIALIZE}$ (create empty priority queue $Q$), $\text{Min}(Q)$ (return smallest element of $Q$), $\text{EXTRACT-MIN}(Q)$ (extract a record $v$ in $Q$ with the smallest field $v.value$), $\text{INSERT}(Q, v)$ (insert a record $v$ into $Q$ according to field $v.value$). E.g., using heaps we could implement INITIALIZE and MIN in time $O(1)$, and EXTRACT-MIN and INSERT in time $O(\log n)$. Let us say that the priority queue is monotone if any new INSERT($v$) operation must have $v.value$ which is at least as large as the value $v'.value$ used by the previous INSERT($v'$) operation. E.g., ignoring all the other EXTRACT-MIN operations, the overall sequence of INSERT operations is non-decreasing. Show how to implement a monotone priority queue so that all four operations INITIALIZE, MIN, EXTRACT-MIN and INSERT take time $O(1)$.

Solution: INSERT YOUR SOLUTION HERE

(b) Now let us come back to the Huffman code implementation when the frequencies $f_1 \leq f_2 \leq \ldots \leq f_n$ are already sorted. We will replace the original priority queue $Q$ with the “union” of two monotone queues $Q_1$ and $Q_2$, where $Q_1$ will only contain the $n$ original frequencies, and $Q_2$ will only contain the frequencies of “meta-characters”, which are the $(n - 1)$ virtual characters used when we merge two previous (regular or mega-) characters. Please “complete all the dots” in the following pseudocode for the fast implementation of Huffman’s algorithm using monotone priority queues $Q_1$ and $Q_2$. The algorithm takes an array $F$ of non-decreasing frequencies and will compute the optimal Huffman tree $T$ for $F$, returning the root of $T$ as the answer.

Fast-Huffman($F, n$)
$Q_1 \leftarrow \text{INITIALIZE}$
$Q_2 \leftarrow \text{INITIALIZE}$
for $i = 1$ to $n$
    Create a new node $z$ with $z.value = F[i]$, $z.p = z.left = z.right = \text{nil}$
    $\text{INSERT}(Q_1, z)$
for $i = 1$ to $n - 1$
    ... //Fill code to extract two nodes $x$ and $y$ with smallest frequencies in $Q_1 \cup Q_2$
    Create a new node $z$ with $z.left = x$, $z.right = y$, $z.p = \text{nil}$
Set the parents \( x.p = z, \ y.z = z \)
Set \( z.value = \ldots \) //Fill the value of \( z \) correctly
\( \ldots \) //Fill code to Insert \( z \) into “appropriate” monotone queue
\textbf{return} \( \ldots \) // Fill code to return the root of the Huffman tree

\textbf{Solution:} INSERT YOUR SOLUTION HERE

(c) Using part (a), argue that your implementation in part (b) takes time \( O(n) \).

\textbf{Solution:} INSERT YOUR SOLUTION HERE

(d) Argue that you implementation in part (b) is correct. Namely, that the queues \( Q_1 \) and (especially!) \( Q_2 \) are \textit{indeed monotone}, and that the algorithm does exactly what the standard algorithm does, but in a more efficient manner. Where did we (crucially) use the fact that \( f_1 \leq \ldots \leq f_n \)?

\textbf{Solution:} INSERT YOUR SOLUTION HERE
Recall Problem 8.1 from the previous homework, namely: You wish to provide the exact change of $T$ cents using the minimum number of coins, and you have denominations in quarters, dimes, nickels and pennies. Now assume that you have an unbounded number of coins of each denomination: infinitely many pennies, nickels, dimes, and quarters.

Recall the greedy algorithm for this problem: “Find the largest coin whose value $v$ is less than or equal to $T$. Add this coin to your solution and recurse on $(T - v)$.”

In this problem, we will argue that the greedy algorithm will always find an optimal solution in this case. That is, the optimal solution can safely use the largest (always available!) coin of value at most $T$. We will do a case analysis:

(a) (1 Point) Argue optimality for the case when $T < 5$.

**Solution:** INSERT YOUR SOLUTION HERE

(b) (2 Points) Consider the case $5 \leq T < 10$, and use (a) above.

**Solution:** INSERT YOUR SOLUTION HERE

(c) (2 Points) Consider $10 \leq T < 25$, and use (a) and (b).

**Solution:** INSERT YOUR SOLUTION HERE

(d) (7 Points) Consider $25 \leq T$, and use (a), (b) and (c).

**Solution:** INSERT YOUR SOLUTION HERE