

**Fall 08. CS513. HW 3, Due Oct 2.** You can assume any of the algorithms we discussed in the class: you should precisely state any such result you use without going into details. Your goal is as always to design the fastest algorithm possible and to use as little space as possible.

1. You are given an array  $A[1, \dots, n]$  of nonnegative numbers and a target  $W$ . A partition of array  $A$  into  $k$  pieces is given by  $l_0 = 0 < l_1 < l_2 < \dots < l_k = n$ . Design an algorithm to find a partition of *smallest* number of pieces such that  $\sum_{j=l_i+1}^{l_{i+1}} A[j] \leq W$  for each piece. You may assume that  $A[i] \leq W$  for all  $i$ .
2. You are given an array  $A[1, \dots, n]$  of nonnegative numbers and an integer  $k$ . A partition of array  $A$  into  $k$  pieces is given by  $l_0 = 0 < l_1 < l_2 < \dots < l_k = n$ . Design an algorithm to find a partition of  $k$  pieces such that  $\max_i \sum_{j=l_i+1}^{l_{i+1}} A[j]$  is minimized. How much space does your algorithm use?
3. The Levenshtein (or edit) distance between two strings is the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. Given strings  $S$  and  $T$ , both of length  $n$ , design an algorithm to compute the Levenshtein distance between  $S$  and  $T$ , as well as determine the series of operations needed to transform  $S$  to  $T$  using the minimum number of operations. How much time and space does your algorithm need?

**Extra Credit.** Solve the problem above using  $O(n)$  space.

4. Consider an ordered list of items of weight  $w_1, \dots, w_n$  and their respective access frequencies  $f_1, \dots, f_n$ ; all weights and frequencies are positive. The cost of item  $w_i$  in this list is sum of weights of items preceding it (including itself), weighted by the frequency of  $i$ , that is,

$$\text{cost}(i) = f_i \sum_{j \leq i} w_j.$$

Design an algorithm to permute the list such that the total cost  $\sum_i \text{cost}(i)$  is the smallest.