COL351 Holi2023: Tutorial Problem Set 3

1. Recall the order statistics algorithm discussed in class. Given an array A, we partition A into groups of size 5 each (possibly except one group), compute the median of each group, then recursively compute the median of the group-medians, and use it as a pivot to divide the array and recurse. How would the complexity of the algorithm change if we created groups of size 7 instead? How would it change if we created groups of size 3?

2. Consider n immiscible liquids and let $\rho[1], \ldots, \rho[n]$ denote their densities. A mixture containing, for all i, volume $V[i]$ of i'th liquid is poured into a separating funnel and the layers are allowed to separate. The tap at the bottom of the funnel is then opened, and a volume v of the mixture is removed (where $v \leq \sum_{i=1}^{n} V[i]$). Design an $O(n)$ time algorithm that takes the arrays ρ, V, and the number v as input, and determines for each i the volume of the i'th liquid removed.

3. Each node v of a binary tree T is labeled with a number x_v. An *independent set* is a subset S of nodes in which no two nodes are adjacent, i.e., have a parent-child relationship. The *weight* of a set S is $\sum_{v \in S} x_v$. Design a linear-time algorithm to find the maximum weight independent set of nodes of a given labeled binary tree.

4. Consider a collection of n stacks, $A_1, \ldots, A_n$, of integers with sizes $s_1, \ldots, s_n$ respectively. We wish to pop a total of at most p integers from these stacks so that their sum is maximized. Design a polynomial-time algorithm to compute this maximum possible sum. You may use $a[i][j]$ to denote the j'th integer from the top in A_i and $b[i][j]$ to denote the sum of top j integers in A_i. (Obviously, to be able to include $a[i][j]$ in your sum, you have to include all the elements above it in stack A_i, that is, $a[i][1], \ldots, a[i][j-1]$.)

5. [CLRS Problem 15-2, slightly modified] You are designing your own document processor (like LaTeX) which converts a given file into the pdf format. The input file is a sequence of n words of length $l_1, \ldots, l_n$, measured in number of characters. Each line of a pdf file can hold a maximum of M characters, and you are required to have one space between consecutive words on the same line. If a given line contains words i through j, where $i \leq j$, the number of extra space characters at the end of the line is $M - (j - i) - \sum_{k=i}^{j} l_k$. This quantity must be non-negative so that the words indeed fit on the line. We define the *messiness* of a word arrangement as the sum over all lines of the cubes of the numbers of extra space characters at the ends of lines. Design a polynomial-time algorithm to compute the messiness of the least messy arrangement of words into lines of the pdf file.

6. [Kleinberg-Tardos Chapter 6 Exercise 12] Suppose we want to replicate a file over a collection of n servers, labeled $S_1, \ldots, S_n$. Placing a copy of the file at server S_i results in a *placement cost* of c_i, for an integer $c_i > 0$.

   Now, if a user requests the file from server S_i, and no copy of the file is present at S_i, then the servers $S_{i+1}, S_{i+2}, S_{i+3}, \ldots$ are searched in order until a copy of the file is finally found, say at server S_j, for $j > i$. This results in an *access cost of* $j - i$. (Note that the lower indexed servers $S_{i-1}, S_{i-2}, \ldots$ are not consulted in this search.) The access cost is 0 if S_i holds a copy of the file. We will require that a copy of the file be placed at server S_n, so that all such searches will terminate, at the latest, at S_n.

   We'd like to place copies of the file at the servers so as to minimize the sum of placement and access costs. Formally, we say that a *configuration* is a choice, for each server S_i ($i = 1, 2, \ldots, n-1$) of whether to place a copy of the file at S_i or not. (Recall that a copy is always placed at S_n.) The *total cost* of a configuration is the sum of all placement costs for servers with a copy of the file, plus the sum of all access costs associated with all n servers.

Give a polynomial-time algorithm to find a configuration of minimum total cost.

7. [Kleinberg-Tardos Chapter 6 Exercise 21, rephrased] Consider the following generalization of the stock trading problem discussed in class. Here we are given as input an array $a_1, \ldots, a_n$, where a_i denotes the stock price in the i'th minute, and an integer $k \geq 1$. We are allowed to buy and sell stocks at most k times, but we must hold at most 1 share at any time. Formally, let b_i and s_i be the time of the i'th buy and the i'th sell transactions respectively, for $i = 1, \ldots, m$. Then we require $m \leq k$ and $1 \leq b_1 < s_1 < b_2 < s_2 < \cdots < b_m < s_m \leq n$, and our net profit is $\sum_{i=1}^{m}(a_{s_i} - a_{b_i})$. Design a polynomial-time algorithm to determine the maximum profit we can make out of the day.

8. You are planning an m-day-m-night hike in the Himalayas in the upcoming summer vacation. You have already identified the trail, which starts from location C_1, passes through locations $C_2, \ldots, C_{n-1}$, and ends at location C_n, where $n > m$. These C_i's are the potential locations where you could set up your camp for an overnight stay. You will start your hike from C_1 in the morning of day 1, spend your m nights at some subsequence of locations $C_1, \ldots, C_n$, and end your hike at C_n in the morning of day $m + 1$. However, you can only walk at most a distance d in one day. Moreover, each campsite C_i has a score s_i (considering proximity of water, scenic beauty, safety, etc.) and you want to maximize the total score of the campsites where you stay. Design a polynomial-time algorithm that takes an array $[s_1, \ldots, s_n]$ of scores, an array $[d_1, \ldots, d_n]$ where d_i is the distance from C_1 to C_i along the trail, and m, d, and computes the maximum possible score of your m-day hike subject to the constraint that you walk at most distance d every day. Note that you have the option of walking distance 0 on some days, that is, stay at the same C_i for more than one night. The score of each campsite is counted as many times as the number of nights you spend there.

9. You have registered today, on day 0, for running your first half-marathon, to be held on day m. You have found an awesome program to train for a half-marathon in n days. The program is an array $A = [a_0, \ldots, a_n]$ of numbers, and it recommends you to run a distance a_i on day i. Here, $a_0 = 0$km, and $a_n = 21.1$km, the half-marathon distance. However, since $n > m$, you don't have sufficiently many days to follow the program. Therefore, you decide to build your own training program $B = [b_0, \ldots, b_m]$ that is a subsequence of A, where $b_0 = 0$, and $b_m = a_n$, the half-marathon distance (you run distance b_i on day i). The difficulty of such a program is given by $\sum_{i=1}^{m}(b_i - b_{i-1})^2$. Design an algorithm to compute, given input A and m, the difficulty of a least difficult training program B.

10. Recall the definition of longest common subsequence (LCS) of two strings and the algorithm to compute LCS discussed in class. Some of you asked whether we can find the number of LCSs of two given strings using dynamic programming. Here is a formal description of the problem.

    Let $a[1..m]$ and $b[1..n]$ be two strings. We say that a pair $(i[1..k], j[1..k])$ of equal length integer sequences is a *common subsequence locator* of a and b if $1 \leq i[1] < i[2] < \cdots < i[k] \leq m$, $1 \leq j[1] < j[2] < \cdots < j[k] \leq n$, and for all $\ell \in \{1, \ldots, k\}$, $a[i[\ell]] = b[j[\ell]]$. Design a polynomial time algorithm that, given strings a and b, outputs the number of common subsequence locators of a and b having the maximum possible length.