CS 344: Design and Analysis of Computer Algorithms **Rutgers: Fall 2024**

Homework #3

Name(s): FIRST$_1$ LAST$_1$, FIRST$_2$ LAST$_2$, FIRST$_3$ LAST$_3$

Homework Policy

- If you leave a question completely blank, you will receive 20% of the grade for that question. This however does not apply to the extra credit questions.
- You may also consult all the materials used in this course (lecture notes, textbook, slides, etc.) while writing your solution, but no other resources are allowed.
- Unless specified otherwise, you may use any algorithm covered in class as a "black box" – for example you can simply write "sort the array in $\Theta(n \log n)$ time using merge sort".
- Remember to always **prove the correctness** of your algorithms and **analyze their running time** (or any other efficiency measure asked in the question). See "Practice Homework" for an example.
- The extra credit problems are generally more challenging than the standard problems you see in this course (including lectures, homeworks, and exams). As a general rule, only attempt to solve these problems if you enjoy them.
- **Groups:** You are allowed to form groups of size *two* or *three* students for solving each homework (you can also opt to do it alone if you prefer). The policy regarding groups is as follows:
  - You can pick different partners for different assignments (e.g., from HW1 to HW2) but for any single assignment (e.g., HW1), you have to use the same partners for all questions.
  - The members of each group only need to write down and submit a single assignment between them, and all of them will receive the same grade.
  - For submissions, only one member of the group submits the full solutions on Canvas and lists the name of their partners in the group. The other members of the group also need to submit a PDF on Canvas that contains only a single line, stating the name of their partner who has submitted the full solution on Canvas (Example: Say A, B, and C are in one group; A submits the whole assignment and writes down the names A, B, and C. B and C only submit a one-page PDF with a single line that says "See the solution of A").
  - You are allowed to discuss the questions with any of your classmates even if they are not in your group. **But each group must write their solutions independently.**

Problem 1. You are given a stream of integers. Given a window size W, output the median of the last W numbers. The algorithm should run in $n \log W$ for a stream of n integers.

Problem 2. You are given an array of characters $S[1:n]$ such that each $S[i]$ is a small case letter from the English alphabet. You are also provided with a black-box algorithm *dict* that given two indices $i, j \in [n]$, $dict(i, j)$ returns whether the sub-array $S[i:j]$ in array S forms a valid word in English or not in $O(1)$ time. (Note that this algorithm is provided to you and you do not need to implement it in any way).

Design and analyze a dynamic programming algorithm to find whether the given array S can be partitioned into a sequence of valid words in English. The runtime of your algorithm should be $O(n^2)$.

Example: Input Array: $S = [maytheforcebewithyou]$.

Assuming the algorithm *dict* returns that $may, the, force, be, with$ and you are valid words (this means that for instance, for may we have $dict(1, 3) =$ True), this array can be partitioned into a sequence of valid words.

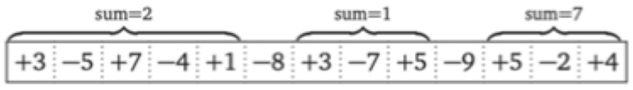
Problem 3. Suppose you are given an array $A[1:n]$ of integers that can be positive or negative. A *contiguous* subarray $A[i:j]$ of A is called a **positive** subarray if sum of its entries is strictly positive, i.e.,

$$\sum_{k=i}^{j} A[k] > 0.$$

Design and analyze a dynamic programming algorithm that in $O(n^2)$ time finds the smallest number of positive contiguous subarrays of A such that every positive entry of A belongs to exactly one of these subarrays. You can assume that A has at least one positive entry and thus the answer is *at least* one.

(25 points)

Example: The correct answer for the following array is 3 subarrays.



Problem 4. We define a number to be **special** if it can be written as:

$$n = a \cdot 197 + b \cdot 232$$

where a and b are non-negative integers. For example:

- 696 is special because it can be written as $696 = 0 \cdot 197 + 3 \cdot 232$.
- 2412 is special because it can be written as $2412 = 4 \cdot 197 + 7 \cdot 232$.
- 267 is not special (since it cannot be written as $a \cdot 197 + b \cdot 232$ for non-negative a and b).

The goal of this problem is to write a dynamic programming algorithm **Special(n)** that, given a positive integer n, either finds non-negative integers a and b such that $n = a \cdot 197 + b \cdot 232$, or returns "no solution" if no such a and b exist.

Remember to separately write your specification of recursive formula in plain English , your recursive solution for the formula and its proof of correctness , the algorithm using either memoization or bottom-up dynamic programming , and runtime analysis.

Problem 5. For the most part, we haven't been optimizing for space in our dynamic programs. But for many of the problems we've discussed it is possible to get away with much less space. The goal of this problem is for you do out one such example.

Recall the subsetsum problem from class. INPUT: set $S = \{s_1, \ldots, s_n\}$ of non-negative integers and a target number B. OUTPUT: set $S' \subseteq S$ with $\sum_{x \in S'} = B$ or FALSE if no such S' exists.

The algorithm we showed in class used $O(nB)$ time and $O(nB)$ space. Write pseudocode for an algorithm that solves subset sum in $O(nB)$ time but only needs $O(B)$ space in addition to the input itself. Your algorithm must return the actual set S'. For this problem you ONLY need to write pseudocode.

Problem 6. ImportExport(A,B)

Input: Two arrays $A[0 \ldots n-1]$ and $B[0 \ldots n-1]$, both of length n, representing the prices of tulips in country A and country B at different times, respectively. All entries in both arrays are positive integers.

Output: Return the maximum possible profit, defined as:

$$\max_{j>i}\{B[j] - A[i]\}$$

or return 0 if all such $B[j] - A[i]$ are negative. You only need to return the value of the maximum profit, not the actual indices i and j.

Example:

Given the input:

$$A = [40, 18, 20, 25, 12, 13, 19, 20, 5, 7, 3]$$
$$B = [15, 50, 5, 30, 34, 19, 28, 33, 20, 20, 20]$$

The maximum possible profit is achieved by buying tulips at time $i = 4$ (when $A[4] = 12$) and selling them at time $j = 7$ (when $B[7] = 33$), which gives a profit of:

$$B[7] - A[4] = 33 - 12 = 21$$

Therefore, the output is 21.

- Give pseudocode for a recursive $O(n)$-time algorithm for ImportExport(A,B).
- Give pseudocode for a $O(n)$-time dynamic programming algorithm for ImportExport(A,B).

Problem 7. Suppose we have n persons with their heights given in an array $P[1:n]$ and n skis with heights given in an array $S[1:n]$. Design and analyze a greedy algorithm that in $O(n \log n)$ time assign a ski to each person so that the average difference between the height of a person and their assigned ski is minimized. The algorithm should output a permutation σ of $\{1, \ldots, n\}$ with the meaning that person i is assigned the ski $\sigma(i)$ such that

$$\frac{1}{n} \cdot \sum_{i=1}^{n} |P[i] - S[\sigma(i)]|$$

is minimized.

Example: If $P = [3, 2, 5, 1]$ and $B = [5, 7, 2, 9]$, we can output $\sigma = [2, 1, 4, 3]$ with value

$$\frac{1}{4} \cdot \sum_{i=1}^{4} |P[i] - B[\sigma(i)]| = \frac{1}{4} \cdot (|P[1] - B[2]| + |P[2] - B[1]| + |P[3] - B[4]| + |P[4] - B[3]|) = \frac{1}{4} \cdot (4 + 3 + 4 + 1) = 3,$$

which you can also check is the minimum value.

Problem 8. Given a set of jobs, each with a deadline and a processing time, schedule the jobs to minimize the maximum lateness, where lateness is defined as the time difference between a job's completion and its deadline (if it's completed after the deadline). Write an algorithm and prove its correctness.

Problem 9. You are given two integer arrays gas$[0 \dots n-1]$ and cost$[0 \dots n-1]$, where:

- gas$[i]$ represents the amount of gas available at the i-th gas station.
- cost$[i]$ represents the amount of gas required to travel from the i-th gas station to the $(i+1)$-th gas station.

Your car starts with an empty gas tank, and you want to travel around the circular route once in the clockwise direction, beginning at one of the gas stations. The car has an unlimited gas tank, so it can collect all the gas available at each station as it passes.

The goal is to determine whether there exists a starting gas station from which you can travel around the circuit once. If such a starting gas station exists, return its index. Otherwise, return -1. If there exists a solution, it is guaranteed to be unique. Design an algorithm that runs in $O(n)$ time.

Challenge Yourself. You are given a string consisting of just the characters $'a'$, $'b'$, or $'c'$. The task is to sort the string in lexicographical order $(a < b < c)$ by finding the minimum number of swaps needed to achieve this. Each swap can exchange two characters in the string.