# CSCI 570 Homework 3

*Due Date: Oct. 1, 2023 at 11:59 P.M.*

1. (**Greedy**) Given $n$ rods of lengths $L_1, L_2, \ldots, L_n$, respectively, the goal is to connect all the rods to form a single rod. The length and the cost of connecting two rods are equal to the sum of their lengths. Devise a greedy algorithm to minimize the cost of forming a single rod.

2. (**Greedy**) During a summer music festival that spans $m$ days, a music organizer wants to allocate time slots in a concert venue to various artists. However, each time slot can accommodate only one performance, and each performance lasts for one day.

   There are $N$ artists interested in performing at the festival. Each artist has a specific deadline, $D_i$, indicating the last day on which they can perform, and an expected audience turnout, $A_i$, denoting the number of attendees they expect to draw if they perform on or before their deadline. It is not possible to schedule an artist's performance after their deadline, meaning an artist can only be scheduled on days 1 through $D_i$.

   The goal is to create a performance schedule that maximizes the overall audience turnout. The schedule can assign performances for $n$ artists over the course of $m$ days.

   Note: The number of performances ($n$) is not greater than the total number of artists ($N$) and the available days ($m$), i.e., $n \leq N$ and $n \leq m$. It may not be feasible to schedule all artists before their deadlines, so some performances may need to be skipped.

   (a) Let's explore a situation where a greedy algorithm is used to allocate $n$ performance to $m$ days consecutively, based on their increasing deadlines $D_i$. If, due to this approach, a task ends up being scheduled after its specified deadline $D_i$, it is excluded (not scheduled). Provide an counterexample to demonstrate that this algorithm does not consistently result in the best possible solution.

   (b) Let's examine a situation where a greedy algorithm is employed to distribute $n$ performance across $m$ days without any gaps, prioritizing performances based on their expected turnouts $A_i$ in decreasing order.

If, as a result of this approach, a performance ends up being scheduled after its specified deadline $D_i$, it is omitted from the schedule (not scheduled). Provide a counterexample to illustrate that this algorithm does not consistently produce the most advantageous solution.

(c) Provide an efficient greedy algorithm that guarantees an optimal solution to this problem without requiring formal proof of its correctness.

3. (**Master Theorem**) The recurrence $T(n) = 7T(n/2) + n^2$ describes the running time of an algorithm $ALG$. A competing algorithm $ALG'$ has a running time of $T'(n) = aT'(n/4) + n^2 \log n$ What is the largest value of $a$ such that $ALG'$ is asymptotically faster than ALG?

4. (**Master Theorem**) Consider the following algorithm `StrangeSort` which sorts $n$ distinct items in a list $A$.

   (a) If $n \leq 1$, return A unchanged.

   (b) For each item $x \in A$, scan $A$ and count how many other items in $A$ are less than $x$.

   (c) Put the items with count less than $n/2$ in a list $B$.

   (d) Put the other items in a list $C$.

   (e) Recursively sort lists $B$ and $C$ using `StrangeSort`.

   (f) Append the sorted list C to the sorted list B and return the result.

   Formulate a recurrence relation for the running time $T(n)$ of `StrangeSort` on an input list of size $n$. Solve this recurrence to get the best possible $O(\cdot)$ bound on $T(n)$.

5. (**Master Theorem**) For the given recurrence equations, solve for $T(n)$ if it can be found using the Master Method. Else, indicate that the Master Method does not apply.

   (a) $T(n) = T(n/2) + 2^n$

   (b) $T(n) = 5T(n/5) + n \log n$–$1000n$

   (c) $T(n) = 2T(n/2) + \log^2 n$

   (d) $T(n) = 49T(n/7) - n^2 \log n^2$

   (e) $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$

6. (**Divide-and-Conquer**) We know that binary search on a sorted array of size $n$ takes $\Theta(\log n)$ time. Design a similar divide-and-conquer algorithm for searching in a sorted singly linked list of size $n$. Discuss its worst-case runtime complexity.

7. (**Divide-and-Conquer**) We know that mergesort takes $\Theta(n \log n)$ time to sort an array of size $n$. Design a divide-and-conquer mergesort algorithm for sorting a singly linked list. Discuss its worst-case runtime complexity.

8. (**Divide-and-Conquer**) Imagine you are responsible for organizing a music festival in a large field, and you need to create a visual representation of the stage setup, accounting for the various stage structures. These stages come in different shapes and sizes, and they are positioned on a flat surface. Each stage is represented as a tuple $(L, H, R)$, where $L$ and $R$ are the left and right boundaries of the stage, and $H$ is the height of the stage.

   Your task is to create a skyline of these stages, which represents the outline of all the stages as seen from a distance. The skyline is essentially a list of positions ($x$-coordinates) and heights, ordered from left to right, showing the varying heights of the stages.

   Take Fig. 1 as an example: Consider the festival setup with the following stages: (2, 10, 5), (8, 16, 3), (5, 12, 9), (14, 19, 7). The skyline for this festival setup would be represented as: (2, 5, 5, 9, 12, 3, 14, 7, 19), with the x-coordinates sorted in ascending order.

   (a) Given the skyline information of $n$ stages for one part of the festival and the skyline information of $m$ stages for another part of the festival, demonstrate how to compute the combined skyline for all $m+n$ stages efficiently, in $O(m + n)$ steps.

   (b) Assuming you've successfully solved part (a), propose a Divide-and-Conquer algorithm for computing the skyline of a given set of $n$ stages in the festival. Your algorithm should run in $O(n \log n)$ steps.

9. (**Dynamic Programming**) Imagine you are organizing a charity event

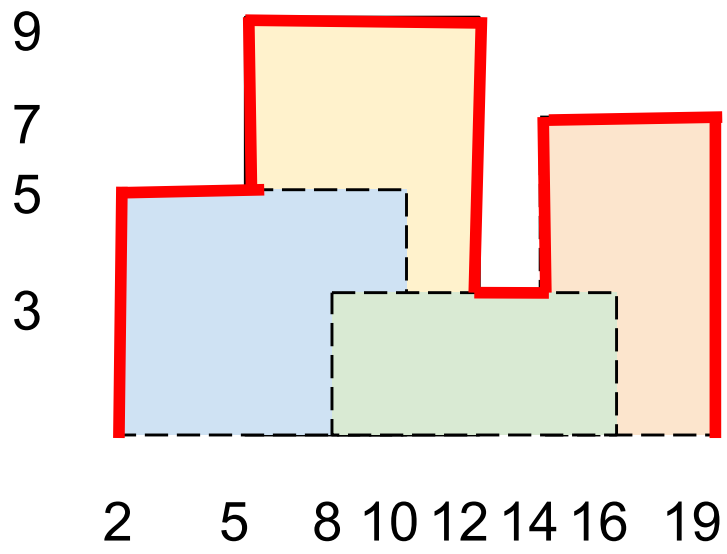


Figure 1: Example of festival stage setup.

to raise funds for a cause you care deeply about. To reach your fundraising goal, you plan to sell two types of tickets.

You have a total fundraising target of $n$ dollars. Each time someone contributes, they can choose to buy either a 1-dollar ticket or a 2-dollar ticket. Use **Dynamic Programming** to find the number of distinct combinations of ticket sales you can use to reach your fundraising goal of $n$ dollars?

For example, if your fundraising target is 2 dollars, there are two ways to reach it: 1) sell two 1-dollar tickets; 2) sell one 2-dollar ticket.

(a) Define (in plain English) subproblems to be solved.

(b) Write a recurrence relation for the subproblems

(c) Using the recurrence formula in part b, write pseudocode using iteration to compute the minimum number of packages to meet the objective.

(d) Make sure you specify base cases and their values; where the final answer can be found.

(e) What is the runtime complexity of your solution? Explain your answer.

10. (**Dynamic Programming**) Assume a truck with capacity $W$ is loading. There are n packages with different weights, i.e. $(w_1, w_2, \ldots w_n)$, and all the weights are integers. The company's rule requires that the truck needs to take packages with exactly weight $W$ to maximize profit, but the workers like to save their energies for after work activities and want to load as few packages as possible. Assuming that there are combinations of packages that add up to weight $W$, design an algorithm to find out the minimum number of packages the workers need to load.

    (a) Define (in plain English) subproblems to be solved.

    (b) Write a recurrence relation for the subproblems

    (c) Using the recurrence formula in part b, write pseudocode using iteration to compute the minimum number of packages to meet the objective.

    (d) Make sure you specify base cases and their values; where the final answer can be found.

    (e) What is the worst case runtime complexity? Explain your answer.