Practice Problems 3

Problem 1. Show the result of executing the Interval Partitioning algorithm from class on the example shown in Fig. 1. Label each interval with the color that it is assigned ($\{1, 2, \ldots\}$). What is the minimum number of colors needed?

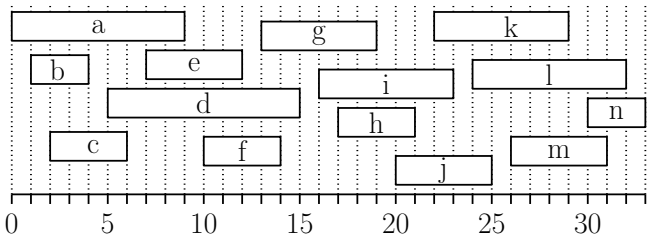

Figure 1: Interval partitioning.

Problem 2. Recall the interval scheduling problem: Given a set of n start-finish intervals, $[s_i, f_i]$, find a maximum-sized subset of intervals that are pairwise disjoint. In class, we showed that earliest finish first (EFF) is optimal. Consider the following alternative greedy solutions:

ESF: Earliest start first: Sort the intervals by start time s_i.

SDF: Shortest duration first: Sort by the intervals by duration $f_i - s_i$.

If there are any ties, break them in favor of earliest finish time. We repeatedly schedule the first interval according to the sorted order, and then remove all intervals that overlap it. For a given set of intervals I, let $\text{Opt}(I)$ denote the maximum number of non-overlapping intervals, and let $\text{ESF}(I)$ and $\text{SDF}(I)$ denote the number of intervals scheduled by each of the above solutions.

Answer the following questions:

(a) Show (by giving a counterexample) that ESF is not optimal. Further, explain how to extend your counterexample to an arbitrarily large interval sets I so that the performance ratio $\text{Opt}(I)/\text{ESF}(I)$ is arbitrarily large.

(b) Consider the following variant ESF, called ESF*. For simplicity, let's assume that there are no duplicate intervals. First, go through and remove any interval that completely contains another. That is, as long as there exist itervals $[s_i, f_i]$ and $[s_j, f_j]$, such that $[s_i, f_i] \subset [s_j, f_j]$, remove $[s_j, f_j]$ from the set of requests. Repeat until there are no nested intervals. (You do not need to explain how to do this. Assume that you are given a procedure that does this.) Then run standard ESF on the remaining "nested-free" set of requests.

Prove that ESF* is optimal (for the original set of requests).

(c) Show (by giving a counterexample) that SDF is not optimal.

(d) Prove that for any instance I, $\text{SDF}(I) \geq \text{Opt}(I)/2$, that is, SDF schedules at least half as many as the optimum.

Problem 3. In class we presented a greedy algorithm for scheduling a set of n tasks, in which each task is given a duration t_i and deadline d_i. We showed that scheduling the tasks in increasing order of deadline minimizes the *maximum* lateness. (Recall that if task i is scheduled at time $s(i)$, then its lateness is $\ell_i = \max(0, s(i) + t_i - d_i)$.) Define the *average lateness* to be $(1/n)\sum_{i=1}^{n} \ell_i$.

(a) Provide a counterexample to show that scheduling tasks in increasing order of *deadline* does not minimize average lateness. Briefly explain your example.

(b) Provide a counterexample to show that scheduling tasks in increasing order of *duration* does not minimize average lateness. Briefly explain your example.

(c) Suppose that we redefine lateness to be $\ell_i = s(i) + t_i - d_i$ (ignoring the "max"). Thus, if the task finishes before the deadline, its lateness is negative. (Intuitively, this can be thought of as a bonus, which can be applied to reduce the positive lateness of some other task.) Prove that if tasks are scheduled in increasing order of duration, then average lateness (under this modified definition) will be minimized. (Hint: Use the same sort of exchange argument we used in class to prove that earliest deadline first minimizes maximum lateness.)

Remark: When constructing a counterexample, try to make the counterexample as simple as possible. For example, a counterexample with three tasks is better than one with five tasks, because it is easier to understand. Also, avoid ambiguous situations. For example, if your algorithm schedules the earliest deadline first, you should not have two identical deadlines and then impose assumptions about which one the algorithm will choose first.

Problem 4. Prof. DM drives his EV from College Park to Miami Florida along I-95. He starts with a full charge and can go for R miles until he needs to recharge. Let $x_0 < x_1 < \cdots < x_n$ denote the locations of the various charging stations along the way, measured in miles from College Park. Let $x_0 = 0$ and x_n be the entire distance to Miami. Present an algorithm which, given R and $\langle x_0, \ldots, x_n \rangle$, determines the *fewest number* of recharging stops he needs to make, without exhausting his battery. Justify the correctness of your algorithm. (You may assume that the gap between consecutive stations, $x_i - x_{i-1}$, never exceeds R.)

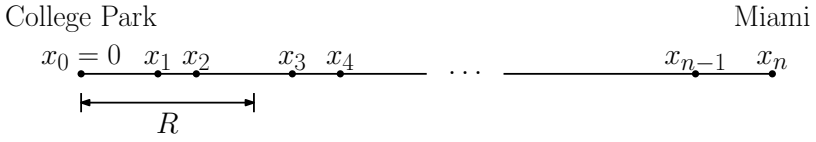

Figure 2: Minimum recharging stops.

Briefly justify your algorithm's correctness and derive your algorithm's running time.

Problem 5. You are given a collection of files $\{f_1, \ldots, f_n\}$ files that are to be stored on a tape. File f_i is s_i bytes long. The tape is long enough to store all the files. The probability of

accessing file f_i is p_i, where $0 \leq p_i \leq 1$, and $\sum_{i=1}^{n} p_i = 1$. The tape is rewound before each access, and so the time to access any file is proportional to the distance from the front of the tape to the end of the file.

A *layout* of files on the tape is given by a permutation $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ of the numbers $\{1, \ldots, n\}$. (For example, Fig. 3 shows the layout $(4, 2, 1, 3)$.) Given a layout π, the *expected cost* of accessing the ith file on the tape is the product of its access probability and the distance from the start of the tape to the end of the file. The *total cost* of a layout π is the sum of the expected costs for all the files, denoted $T(\pi)$.

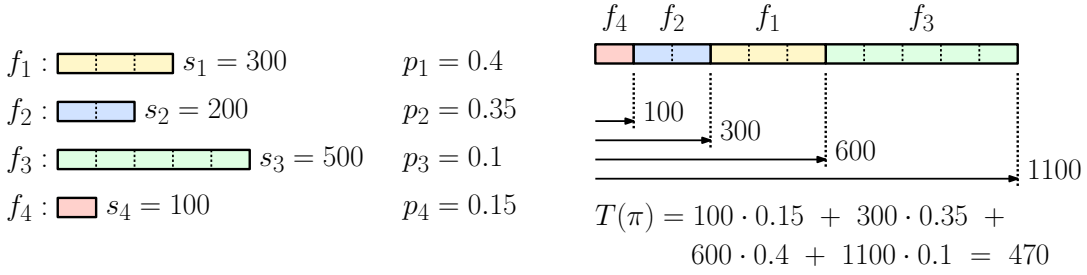

Figure 3: Placing files to minimize access time.

(a) Present a (short) counterexample so show that laying out the files on the tape in increasing order of size (s_i) is not optimal.

(b) Present a (short) counterexample so show that laying out the files on the tape in decreasing order of access probability (p_i) is not optimal.

(c) Present an algorithm, which given s_i's and p_i's, determines a layout π of minimum total cost. Prove your algorithm's correctness and derive its running time. (Hint: Use a greedy approach based on some simple function of both s_i and p_i.)