Introduction to Algorithms	**Problem Set 3 (3 problems)**
CS 4820 Spring 2023	**Due: Feb 16, 2023 (Thursday) 11:59 pm**

Hand in your solutions electronically on Gradescope (except for coding problems, which should be submitted as indicated in the question). Your submission should be typeset (except for hand-drawn figures). Collaboration is encouraged while solving the problems, but:

1. list the NetID's of those in your group;
2. you may discuss ideas and approaches, but you should not write a detailed argument or code outline together;
3. notes of your discussions should be limited to drawings and a few keywords; you must **write up the solutions and write code on your own**.

Remember that when a problem asks you to design an algorithm, you must also prove the algorithm's correctness and analyze its running time, i.e., the running time must be bounded by a polynomial function of the input size.

(1) (10 points) One day while resting under an apple tree, Anand figured out how to perfectly toss a ring onto any peg in the first attempt. Anand goes to Apple Fest to show off their newfound skill. Anand won a raffle and earned the right to play a lucrative new version of the ring toss. Here, you are given n rings and n pegs with radii $r_1, ..., r_n$, are kept in that order from left to right. Every time a ring is tossed correctly onto a peg, that peg along with the ring is completely removed. If you toss correctly onto peg j and it has left neighbor peg i and right neighbor peg k, then you get $r_i \times r_j \times r_k$ dollars. If j does not have a left neighbor and has a right neighbor or j does not have a right neighbor and has a left neighbor, then you get $r_j \times r_k$ dollars and $r_i \times r_j$ dollars respectively. If j is the only peg left, then you get r_j dollars. Anand needs your help to figure out what is the maximum amount of money they can make from this game.

The input format (passed via stdin, terminated by a newline) is as follows:

- One line containing one positive integer: n, the number of pegs.
- One line containing n positive integers, the radii r_i of each peg for $1 \leq i \leq n$.

Your program should output, to stdout, 1 line:

- The first line should contain the maximum amount of money Anand can make from this game.
- Your output should be terminated by a newline.

Here are some input bounds:

- $1 \leq n \leq 250$.
- $1 \leq r_i \leq 50$.

The 5820 version of the problem has some additional tasks:

- Print what order Anand should throw rings in to make as much money as possible. Anand is particular about saving up earlier pegs for the last. So, if there are multiple orders that attain maximum money, Anand would prefer so that the reverse of the order of the pegs is lexicographically smallest.

For 5820 version, the input format and input bounds will remain the same. Your program should output, to stdout, 2 lines:

- First line should contain the maximum amount of money Anand can make from this game.
- Second line should contain n integers, the order of pegs which Anand should follow to toss rings onto (if there are many, then output the one so that the reverse of the order is lexicographically the smallest).
- Your output should be terminated by a newline.

Here are some hints:

- We will use dynamic programming to solve this!
- One way to define state is let $OPT(l, r)$ be maximum amount you can make if pegs $l, \ldots, r$ were left. A natural attempt then is to choose the first peg to throw the ring at and then recurse on left half and right half.
- This doesn't work! Say you want to compute $OPT(l, r)$ and you let ring be thrown onto peg m, and you recurse on two halves by computing $OPT(l, m-1)$ and $OPT(m+1, r)$. The issue is that whenever you will throw ring onto peg $m-1$, the points you get from that throw will include the value of the leftmost peg of the right half. And you don't know in advance what that rightmost peg will be! It could be that you may have already thrown a ring on peg $m+1$ say and now $m+2$ is present there. Working with an example should clarify more why this doesn't work.
- In other words, choosing m to be the first peg to toss a ring onto caused $m-1$ and $m+1$ to become neighbors and then you have to somehow keep track of who is the actual neighbor which seems messy.
- The idea to overcome this is to choose m to be the last peg to toss a ring onto in the range (l, r)!
- If we let m be the last peg to toss a ring onto, how will we know what its neighbors will be? It will be that m will be the only ring left after we recurse in the range (l, r) after we recurse on left and right ranges but there are pegs alive outside the range that could become its neighbors. This seems to get messy as well.
- The idea to get out of this pickle is to redefine $OPT(l, r)$ to help us keep neighbors alive.
- Let $OPT(l, r)$ be maximum amounts of points you can get by tossing rings onto pegs $l+1, \ldots, r-1$ but not at pegs l, r! This is nice as when you choose m to be the last peg to toss a ring onto, your left and right half computations will be $OPT(l, m)$ and $OPT(m, r)$. After this, the three pegs left will be l, m, r. Then, for computing $OPT(l, r)$, by definition, you want to keep l, r alive and just toss a ring onto m.
- Our final answer, won't be $OPT(1, n)$ as that will keep $1, n$ alive. It will instead be $OPT(0, n+1)$.
- For 5820 students, the way to get the reverse earliest lexicographic string is to have a parent array. And also keep track of rightmost peg that gives you optimal value. You can use recursion to print your solution.

Note that students enrolled in CS 4820 will submit rings-4820 assignment and those enrolled in CS 5820 will submit rings-5820 assignment on autograder.

(2) (10 points) You have just been hired as a new part-time employee of a local escape room business. There are three rooms that each run the same scenario, so when a group of guests arrives they could go to any one of the three. Each room has an employee who monitors the guests and provides hints. You cannot have two groups in your room at a time. Each group has rented a room for different amounts of time, and has paid a different amount for tickets, based on the size of the group, when they bought the tickets, and any discounts they might have. You are allowed to choose with groups you want each week, and your boss will then assign a schedule for those groups as long as you don't go over your allotted maximum amount of time with the room each week.

a. (ungraded - 0 points, useful exercise but do not submit) As a new employee, you are not paid per group, but per minute you work. In order to maximize your pay, you want to work as many minutes as possible. The escape room is open for M minutes each week. You have a list of groups of length n, where each group i has scheduled the room for m_i minutes. Assume all $m_i > 0$. Write an algorithm to choose which groups to take to earn the most money, while not exceeding M minutes of using the room. Your runtime should be bounded by a polynomial function of M and n.
b. (10 points) Now that you have experience, your boss will offer you a bonus by paying you a set fraction of the ticket cost for each group, meaning your pay will vary from group to group. You will receive r_i money for taking group i. Given some list of n groups where each group i needs the room for m_i minutes and will pay r_i, write an algorithm to choose which groups to take to earn the most money, while not exceeding M minutes of using the room. Assume all $m_i > 0, r_i > 0$. Your runtime should be bounded by a polynomial function of M and n.
c. (5820 STUDENTS ONLY) (10 points) Your boss has decided to run a promotion. When a group purchases tickets, they can also purchase some number of lottery tickets where they can spin a wheel for a prize if they find a hidden item in the room. It's possible for each ticket to win a prize, but you only have P prizes. Your boss is a very cautious person and wants you to make sure you don't take groups with more than P total lottery tickets. Given some list of n groups where each group i needs the room for m_i minutes, will pay r_i, and has p_i lottery tickets, write an algorithm to choose which groups to take to earn the most money, while not exceeding M minutes of using the room or P lottery tickets. Assume all $m_i > 0, r_i > 0, p_i \geq 0$. Your runtime should be bounded by a polynomial function of M and n, and P.

(3) (10 points) In American politics, *gerrymandering* refers to the process of subdividing a region into electoral districts to maximize a particular political party's advantage. This problem explores a simplified model of gerrymandering in which the region is modeled as one-dimensional. Assume that there are $n > 0$ *precincts* represented by the vertices $v_1, v_2, \ldots, v_n$ of an undirected path. A *district* is defined to be a contiguous interval of precincts; in other words a district is specified by its endpoints $i \leq j$, and it consists of precincts $v_i, v_{i+1}, v_{i+2}, \ldots, v_j$. We will refer to such a district as $[i, j]$.

Assume there are two parties A and B competing in the election, and for every $1 \leq i \leq j \leq n$, $P[i, j]$ denotes the probability that A wins in the district $[i, j]$. The probability matrix P is given as part of the input. Assume that the law requires the precincts to be partitioned into exactly k disjoint districts, each containing at least $s_{\min}$ and at most $s_{\max}$ nodes. You may also assume the parameters $n, k, s_{\min}, s_{\max}$ are chosen such that there is at least one way to partition the precincts into k districts meeting the specified size constraints. Your task is to find an efficient algorithm to gerrymander the precincts into k districts satisfying the size constraints, so as to maximize the expected number of districts that A wins.