ASSIGNMENT - 2
Analysis and Design of Algorithms - CSE222

Q1 - **(15 points) Multiplying k n-digit Numbers**
We have k numbers, A[1], A[2], . . . A[k], each with up to n digits, with k much larger than n. (Aside about a slower algorithm, you can skip this: Note that if we multiply one number at the time, we may need to multiply a Ω(kn) digit number by an n digit number Ω(k) times, giving a total cost of at least Ω(k^2).)

The goal is to compute their product A[1] × A[2] × . . . × A[k] more efficiently using the following two facts.
• There an algorithm for multiplying two numbers with n digits each in O($n^{1.6}$) time.
• The product of k numbers each with n digits has at most O(kn) digits.

(a) **(5 points)** Show that if we have the product of A[1] . . . A[mid] and A[mid + 1] . . . A[k] where mid = k/2 (assume k is even), then we can compute the product of A[1] × . . . × A[k] in O($n^{1.6}$ $k^{1.6}$) time.
(b) **(10 points)** Using the conclusion of part (a), or any other method of your choice, design an algorithm for computing the overall product of these k n-digit numbers in O($n^{1.6}$ $k^{1.6}$) time or better and give a brief justification of its running time.

Q2 - **(20 points) Finding the Missing Number** Suppose you are given an unsorted array of all integers in the range 0 to n except for one integer, denoted the missing number. Assume n = $2^k - 1$

(a) **(12 points)** Design a O(n) Divide and Conquer algorithm to find the missing number. Partial credit will be given for non Divide and Conquer algorithms. Argue (informally) that your algorithm is correct and analyze its running time.
Suppose the integers in

(b) **(8 points)** Suppose the integers in A are stored as k -bit binary numbers, i.e., each bit is 0 or 1. For example, if k=2 and the array A= [01,00,11], then the missing number is 10. Now the only operation to examine the integers is BIT-LOOKUP (i,j), which returns the jth bit of number A[i] and costs unit time. Design an O(n) algorithm to find the missing number. Argue (informally) that your algorithm is correct and analyze its running time.

Q3 - **(15 points) Typesetting**

Suppose we would like to neatly typeset a test. The input is a sequence of n words of lengths l $l_1, l_2, \ldots, l_n$ (measured in the number of fixed-size characters they take). Each-line can hold at most P characters, the text is left-aligned, and words cannot be split between lines. If a line contains words from i to j (inclusive) then the number of spaces at the end of the line is s= P - $\sum_{k-i}^{j} l_k$ -(j-i). We would like to typeset the text so as to avoid large white spaces at the end of lines; formally, we would like to minimize the sum over all lines of the square of the white spaces at the end of the line. Give an efficient algorithm for this problem. What is its running time?

Q4. **(20 points)** You are given an exam with questions numbered 1, 2, 3, . . . , n. Each question *i* is worth p_i points. You must answer the questions in order, but you may choose to skip some questions. The reason you might choose to do this is that even though you can solve any individual question *i* and obtain the p_i points, some questions are so frustrating that after solving them you will be unable to solve any of the following f_i questions.

Suppose that you are given the p_i and f_i values for all the questions as input. Devise the most efficient algorithm you can for choosing set of questions to answer that maximizes your total points, and compute its asymptotic worst-case running time as a function of n.

Q5 - **(30 points)** Before the invention of book-printing, it was very hard to make a copy of a book. All the contents had to be re-written by hand by so-called scribers. The scriber had been given a book and after several months he finished its copy. One of the most famous scribers lived in the 15th century and his name was Xaverius Endricus Remius Ontius Xendrianus (Xerox). Anyway, the work was very annoying and boring. And the only way to speed it up was to hire more scribers.

Once upon a time, there was a theater ensemble that wanted to play famous Antique Tragedies. The scripts of these plays were divided into many books and actors needed more copies of them, of course. So they hired many scribers to make copies of these books. Imagine you have m books (numbered 1, 2, . . . , m) that may have the different number of pages ($p_1, p_2, \ldots, p_m$) and you want to make one copy of each of them. Your task is to divide these books among k scribes, k ≤ m. Each book can be assigned to a single scriber only, and every scriber must get a continuous sequence of books. That means, there exists an increasing succession of numbers 0 = $b_0 \leq b_1 \leq b_2 \leq \cdots \leq b_{k-1} \leq b_k$ = m such that i-th scriber gets a sequence of books with numbers between b_{i-1} + 1 and b_i. The time needed to make a copy of all the books is determined by the scriber who was assigned the most work. Therefore, our goal is to minimize the maximum number of pages assigned to single scriber. Your task is to find the optimal assignment.

Answer the following related questions:

1. **(5 points)** Define the sub-problem so that we can solve the problem with dynamic programming. Please clearly mark the parameters of the sub-problems and what each parameter represents.

2. **(5 points)** In this problem, the cost function that we need to optimize is the maximum number of pages assigned to a single scriber. Using your sub-problem definition, write down the recurrences that determine the value of the cost function based on the ones of smaller related sub-problems. Please also write down the boundary condition(s) of the cost function.

3. **(5 points)** Using a bottom-up style, please write down the pseudo code or the C code of your algorithm to solve the problem in O(mk)-time.

4. **(5 points)** Please analyze the time complexity of your algorithm and show that it is indeed O(mk).

5. **(10 points)** Please prove that this problem exhibits optimal substructure. Note that the proof will be in a slightly different format than the ones that we talked about in the lectures. The proof should state that optimal solution(s) to sub-problem(s) can be used to put together the optimal solution to the problem (the big problem). You should start by assuming that you can find the optimal solution(s) to the sub-problem(s), and use it (them) to lead to a solution to the big problem (we usually do it the opposite way). Then, assume that you can find a better solution than this solution to the big problem, and try to find a contradiction which proves that this assumption is false. Then the proof is completed.

.