Instructions. Limited collaboration is allowed while solving problems, but you must write solutions yourself. List collaborators on your submission.

You can choose which problems to complete, but must submit at least one problem per assignment. See the course page for information about how challenge problems are graded and contribute to your homework grade. Since you don’t need to complete every problem, you are encouraged to focus your efforts on producing high-quality solutions to the problems you feel confident about. There is no benefit to guessing or writing vague answers.

If you are asked to design an algorithm, please (a) give a precise description of your algorithm using either pseudocode or language, (b) explain the intuition of the algorithm, (c) justify the correctness of the algorithm; give a proof if needed, (d) state the running time of your algorithm, (e) justify the running-time analysis.

Submissions. Please submit a PDF file. You may submit a scanned handwritten document, but a typed submission is preferred. Please assign pages to questions in Gradescope.

1 Challenge Problems

Problem 1. Database medians. You are working as a programmer for UMass administration, and they ask you to determine the median GPA for all students. However, student GPAs are stored in two different databases, one for in-state students and one for out-of-state students. Assume there are $n$ students of each type, so there are $2n$ students total. You’d like to determine the median of this set of $2n$ values, which we will define here to be the $n$th smallest value.

However, security is very tight, so the only way you can access these values is through queries to the databases. In a single query, you can specify a value $k$ to one of the two databases, and the chosen database will return the $k$th smallest value that it contains. Since queries are expensive, you would like to compute the median using as few queries as possible.

Give an algorithm that finds the median value using at most $O(\log n)$ queries.

Problem 2. Decimal to Binary. Recall that in class we designed a divide-and-conquer algorithm to multiply two $n$-digit decimal numbers in time $O(n \log_2(3))$. In this problem, we’ll assume the existence of a subroutine fastmultiply$(x,y)$ that takes two $n$-bit binary numbers $x$ and $y$, and returns the binary representation of their product $xy$ in time $O(n \log_2(3))$. We’ll then use this subroutine to convert numbers from decimal to binary.

Decimal numbers will be represented as arrays of digits, so you can index them to access the $i$th digit, but are unable to directly multiply them without first converting to binary.

1. We’ll first design an algorithm pwr2bin to convert a decimal number that is a power of 10 to binary. Specifically, pwr2bin$(n)$ computes the binary representation for $10^n$. Assume that $n$ is a power of 2.

   def pwr2bin$(n)$:
   
   If $n = 1$: return 1010 (decimal 10 in binary)
   Else:
       $z = /*$ FILL ME IN $*/$
       
       Return fastmultiply$(z,z)$.

What is the appropriate value for $z$? What is the running time of the algorithm?
2. Next, we will design an algorithm \texttt{dec2bin}(x), which accepts as input an \(n\)-digit decimal number \(x\), and returns the binary representation of \(x\). Assume that \(n\) is a power of 2.

\begin{verbatim}
def dec2bin(x):
    if length(x) = 1: return binary(x)
    else:
        split x into \(x_L\) and \(x_R\), where \(x_L\) contains the first \(n/2\) digits, and \(x_R\) contains the last \(n/2\) digits.
        return /* FILL ME IN */.
\end{verbatim}

The subroutine \texttt{binary}(x) performs a lookup into a table containing the binary value of all decimal numbers 0, \ldots, 9. What are we supposed to return? What is the running time of this algorithm?

**Problem 3.** Maximum Subsequence Sum. Recall the maximum subsequence sum (MSS) problem, for which we gave a \(\Theta(n \log n)\) divide-and-conquer algorithm. In this problem you will develop a dynamic programming algorithm with running time \(\Theta(n)\) to solve the problem.

The input is an array \(A\) containing \(n\) numbers, and the goal is to find a starting index \(s\) and ending index \(t\) (where \(s \leq t\)) so the following sum is as large as possible:

\[
\]

For example, in the array
\[
A = \{31, -41, 59, 26, -53, 58, 97, -93, -23, 84\}
\]
the maximum is achieved by summing the third through seventh elements, to get 59 + 26 + (−53) + 58 + 97 = 187, so the optimal solution is \(s = 3\) and \(t = 7\). When all entries are positive, the entire array is the answer (\(s = 1\) and \(t = n\)). In general, we will allow the case \(s = t\) (a sequence of only one element) but not allow empty subsequences.

Give a dynamic algorithm to find the value of a maximum subsequence sum. For this problem, you only need to return the value, not the maximum subsequence itself.

**Hint:** Define \(\text{OPT}(j)\) to be the maximum sum of any subsequence that \textit{ends at index} \(j\).

**Problem 4.** Chicken Wings. The image to the right is a real restaurant menu. The goal of this problem is to find the cheapest way to buy \(V\) chicken wings for some integer \(V\) given a menu like this one. Assume the menu is given as a list of \(n\) menu items \((v_1, w_1), (v_2, w_2), \ldots, (v_n, w_n)\), where \(w_i\) is the price to buy \(v_i\) wings and \(v_i\) and \(w_i\) are both integers. In the example, assuming costs are computed in cents, we would have:

\[
(v_1, w_1) = (4, 455)
\]
\[
(v_2, w_2) = (5, 570)
\]
\[
(v_3, w_3) = (6, 680)
\]

You are allowed to choose any combination of orders whose quantities add up to \(V\), including ordering the same quantity multiple times. You can assume there is always \textit{some} combination of orders to buy exactly \(V\) wings.

(a) There is a natural greedy algorithm where you first buy the largest quantity \(v_i\) such that \(v_i \leq V\), and then repeat on the remaining \(V - v_i\) wings.

Show that this algorithm is not optimal for the menu shown to the right.

(b) Write a dynamic programming algorithm to find the cost of the cheapest set of orders to buy exactly \(V\) wings.

(c) Modify your algorithm to also return the set of orders you could make to achieve the smallest cost.