Consider a binary search tree. For any element \( x \), define \( \text{Successor}(x) \) to be the successor of \( x \) in the inorder traversal sequence. In class we saw how to find \( \text{Successor}(x) \) in time \( O(h) \), where \( h \) is the height of the tree. In this problem we augment the binary search tree such that each node now stores an additional attribute \( x.succ \) with a pointer to \( \text{Successor}(x) \). This will allow us to find \( \text{Successor}(x) \) in time \( O(1) \).

In the following subproblems you can use the procedures covered in class: \texttt{Minimum}, \texttt{Maximum}, \texttt{Predecessor}, \texttt{Successor}, \texttt{Search}, each of which runs in time \( O(h) \).

For the augmented data structure above, show how the additional information can be maintained

(a) (4 points) under the \texttt{Insert} operation; write pseudo code and argue that the asymptotic complexity of the operation is preserved;

\textbf{Solution:} INSERT YOUR SOLUTION HERE

(b) (4 points) under the \texttt{Delete} operation; write pseudo code and argue that the asymptotic complexity of the operation is preserved.

\textbf{Solution:} INSERT YOUR SOLUTION HERE

(c) (4 points) \textbf{[Extra credit]} Now consider a binary search tree, where each node stores two attributes \( x.succ \) and \( x.pred \), which point to the successor and predecessor in the tree. Argue that with this modification, deletion of an element \( x \) can be done in constant time \( O(1) \) while maintaining \( succ \) and \( pred \) attributes. \( (x \) is given as a pointer so you can access it in \( O(1) \) time, and need not expend \( O(h) \) time to find \( x \) itself.

\textbf{Solution:} INSERT YOUR SOLUTION HERE
Assume that you attend a boring science convention that takes place in a conference center that is arranged as an $n \times m$ grid with a mini-exhibition at each node. To attract customers, each exhibition offers a certain number of free pizza slices to passing customers. You want to move from the north-west corner to the south-east corner of the conference center. Since you do not want to spend too much time there, in each step you may either move south or east. Your goal is to maximize the total number of free pizza slices you can get.

More formally, if you pass by the exhibition located at node $(i, j)$ where $0 \leq i \leq n$ and $0 \leq j \leq m$, you get $p(i, j)$ slices of pizza. You are told all of the $p(x, y)$ a priori. Your goal is to design a path from $(0, 0)$ to $(n, m)$ allowing you to consume as much of pizza as you can!

Give an efficient (i.e., polynomial in $n$ and $m$) dynamic programming algorithm for this problem, and analyze its running time.

\textbf{Solution: } INSERT YOUR SOLUTION HERE
You may or may not be old enough to remember how people used to write text messages a while back: on a standard issue phone, the $N = 26$ letters would be assigned in groups of three or four to the $K = 8$ keys $2$ to $9$ on the dial pad, and in order to type the $l$th letter on a particular key, one would tap that key $l$ times in rapid succession.

Consider a text for which the frequency $f_i \in \mathbb{N}$ of each letter $i = 1, \ldots, N$ is known. Typing this text on a phone would require $T = \sum_{i=1}^{N} p_i \cdot f_i$ taps, where $p_i$ is the position of letter $i$ on its key. Clearly, if one wishes to minimize $T$, the optimal assignment of letters to keys—in alphabetical order—depends on the text in question; more precisely, on the frequencies $f_i$. In particular, having groups of just three or four letters may be suboptimal.

In this task you will develop an algorithm that given (general) $K$, $N$, and frequencies $f_1, \ldots, f_N$ finds an assignment of the letters $1, \ldots, N$ in alphabetical order to the keys $1, \ldots, K$ such that $T$ is minimized.

(a) (4 points) For $1 \leq i \leq j \leq N$, let $C[i, j] := \sum_{k=i}^{j} (k - i + 1) \cdot f_k$, which is the number of taps required to type only the letters $i, \ldots, j$ in the text if they are all assigned to a single key (in alphabetical order).

Provide pseudocode of an $O(N^2)$ algorithm that fills the entire array $C[\cdot, \cdot]$. Ignore cells $C[i, j]$ where $i > j$.

**Solution:** INSERT YOUR SOLUTION HERE

For $1 \leq n \leq N$ and $1 \leq k \leq K$, define $T[n, k]$ to be the smallest possible number of taps required to type only the letters $1, \ldots, n$ in the text if they are assigned to keys $1, \ldots, k$.

(b) (2 points) For $i = 1, \ldots, n$, what is $T[i, 1]$ (in terms of $C$)?

**Solution:** INSERT YOUR SOLUTION HERE

(c) (5 points) Derive a recurrence relation for $T[n, k]$ and justify it. Note that in (b) you considered the base case of this recurrence.

**Solution:** INSERT YOUR SOLUTION HERE

In the following you may assume that you have direct access to the filled array $C[\cdot, \cdot]$.

---

1 For simplicity letters are denoted by numbers in the remainder of this task.
(d) (5 points) Devise a polynomial-time (in $N$ and $K$) algorithm PHONE-TD($N, K$) (write pseudocode) that fills table $T[\cdot, \cdot]$ in a top-down, recursive fashion with memoization.

Solution: INSERT YOUR SOLUTION HERE

(e) (5 points) Provide pseudocode for an algorithm PHONE-BU($N, K$) that fills $T[\cdot, \cdot]$ in a bottom-up manner. Analyze the running time of your algorithm.

(Hint: Fill the table in column-major order, i.e., fill the first column, then the second, etc.)

Solution: INSERT YOUR SOLUTION HERE

(f) (6 points) [Extra credit] Finally, develop a procedure PHONE-FIND($N, K$) (write pseudocode) that uses the filled $T[\cdot, \cdot]$ to output an assignment of the letters to the keys such that the text can be typed with $T[N, K]$ taps. Write pseudocode and analyze the running time of your algorithm. If it leads to a faster PHONE-FIND, you may modify your algorithm PHONE-BU to make it store auxiliary values (accessible to PHONE-FIND).

Solution: INSERT YOUR SOLUTION HERE