(a)

To formulate this problem recursively, we need to compare every single possibility with the word c by check if ($c[1 \cdots i]$ equals one of $\begin{cases} \text{if Blend}(s_1[1 \cdots i], s_2[0 \cdots i]) \\ \text{if Blend}(s_1[0 \cdots i], s_2[1 \cdots i]) \end{cases}$)

Without the repetitive recursive steps in the recursive operation above, it is basically checking every possible combination of s_1 and s_2, which is the number of distinct subproblems ($n \cdot k$) → (categorized subproblem)

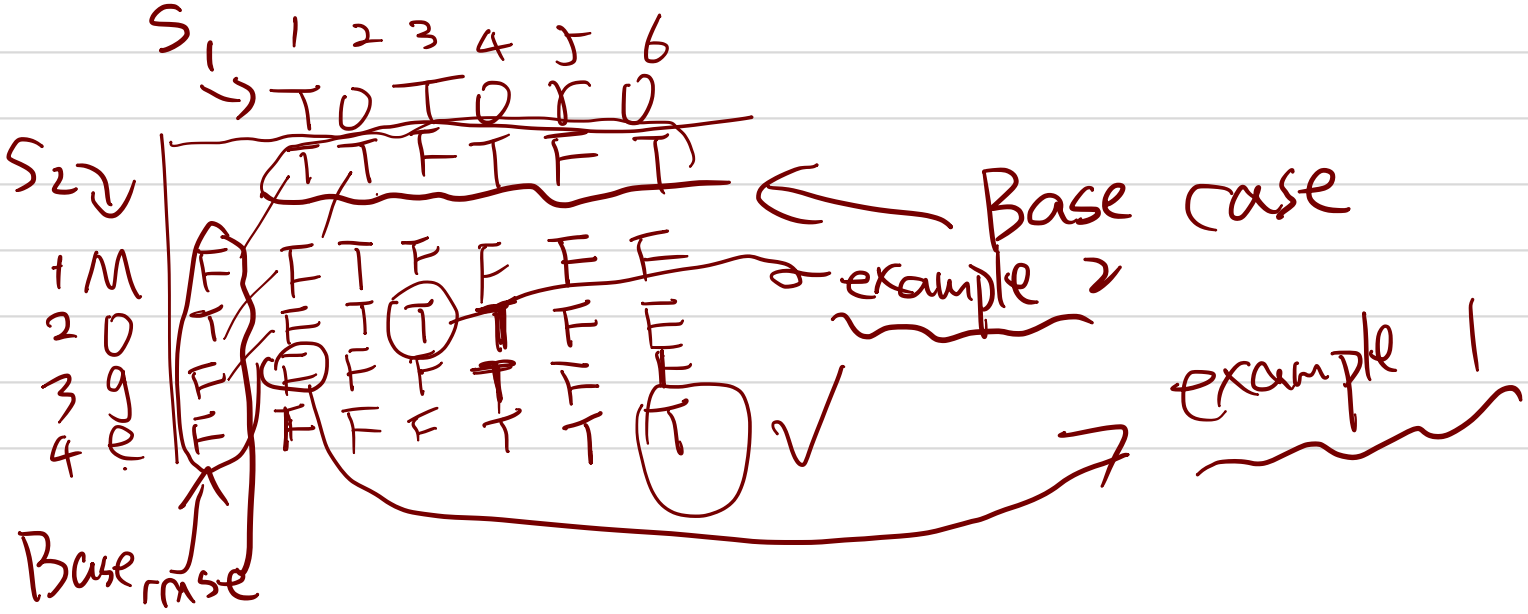
The table will be like below:

1 2 3 4 5 6 7 8 9 10

TOMOTOgeRO

s_1: TOTORO

s_2: Muge



There are several rules for this table:

① Base case: which are located in the first row and the first column of the table, row is respond to if the i^{th} number in S_1 match the i^{th} number in C. So the j^{th} item in S_1 is r, not matching the j^{th} item in C (which is a T), so an F is given. For the column, same thing, it checks if the i^{th} number in S_2 match the i^{th} number in C. By this rule we fill the base case.

② For the rest of the table, the rule is pretty straight forward:

we at least need one T from either top or left to go on filling the table.
If both top and left is F, which means there is no way for the combination of element before it satisfied C.
For example, in column 1, row 3, the top and left both F, that means I MOG or MOG I satisfied the first four elems in C (which is TOMD)

(3) When there is at least one T from left or right, it means that at least the element from Top and before, or left and before satisfied the C subsequence. then what to do is sum up its

coordinate and goes to C[row+col] to see if elem from top or left equals to that. For example (example 2) row 2 column 3, one of the top and left is T. Then we sum up row and column. We get 2+3 = 5. S[5] = T, the top above element is T, then it matches, so at last the result is a T. otherwise if it doesn't match top element or left elem, it is an F.

The size of table is n·k

2.

(a) I will use a linked list to handle these two operations. For insert, I will just need to link the new data to the head or to the tail of the linked-list. Which takes O(1) constant time to do that. For delete half operation, I will just use the deterministic algorithm for median finding to find the median of the unsorted linked list first (because we are inserting to the head and tail arbitrarily) This takes O(k). After get the median, it takes another O(k) to partition the list using the median as a pivot to link every single element that larger than median after it. At last, to delete the element larger than median, it just takes O(1) to unlink every element after median. In total, delete half takes O(k)+O(k)+1, which is O(k).

median-finding partition delete

Worst time complexity for one and n operation will be include after analyse

(b) For accounting method, we first look at how much it cost for one operation

Insert	Delete
1	Avarage of delete : for delete $\frac{n}{2}$ $cn / (\frac{n}{2}) = 2c$ ← constant

For the saving to the bank, we save $\underbrace{1+2c}_{f(n)}$ (pretend cost) each time.

so for a insert, we spend 1 and save 2c

current bank : 2c

if we insert another, it takes 1 and save another 2c, current bank : 4c

After inserting k elements, we got the saving of 2ck. And we decide to operate delete half, which cost $2c \cdot \frac{k}{2} = ck$, we paid that using half of our saving, remains $\underbrace{ck}$ for the next round of operation. even operate delete half again, it will still remains half of original saving.

(C) $\Delta\Phi$ = Δ size

$\hat{c}_i = c_i + \Delta\Phi_i$

	Insert	Delete half
$\Delta\Phi$	1	$-\frac{n}{2}$
$\hat{c}_i$	$1+1$	$cn - \frac{n}{2}$ ← bad, still linear

change $\Delta\Phi = \Delta\text{size} \cdot 2c$

	Insert	Delete half
$\Delta\Phi$	1	$-\frac{n}{2} \cdot 2c = cn$
$\hat{c}_i$	$1 + 2c$	$cn - cn = 0$

average sum of one (Insert/delete) operation

both constant done

In conclusion, the worst time complexity for one operation is one operation of delete half max, which is c·n (O(n)), the naive worst time complexity for n operations is n · cn (O(n²)), but depends on the amortize result, the worst time complexity should be n · (1+2c), which is basically O(n)