1. Solution:

After we add the DECREMENT operation, we could define its psetedcode, as following:

```
DECREMENT(A)
i = 0
while i < A.length and A[i] = 0          // this loop keeps substracting 1 until reaches 1 or boundar
        A[i] = 1
        i = i + 1          // means minus 1, changing from 0 to 1

if i < A.length
        A[i] = 0
```

So, we could see the operation of DECREMENT is just the opposite of INCREMENT, and the maximum cost of one DECREMENT operation is k. In this case, we could see that all bits are changed, aka, 000000…00000 ——> 111111…11111

Now that we have both the INCREMENT and DECREMENT operations, we should conclude that the overall operations are the mixture of them, then it is a sequence of increase and decrease.

We could see that when changing from 00000…00000 —> 11111…11111, the DECREMENT cost is maximized and the cost for this single operation is k. And in this case, when we perform the INCREMENT, changing from 11111…11111 -> 00000…00000, the INCREMENT cost is also maximized, and the cost for this single operation is k.

Now we can see that for a single operation the maximum cost is k, and it is possible to perform INCREMENT and DECREMENT one by one, as the following sequence:

INC, DEC, INC, DEC…. INC, DEC

Then in this extreme situation, each step the cost is k, and it is maximized, so the total of n operations is O(nk).

We could easily know that this prove is general, and conclude that when changing from all 1 to all 0 by adding 1 at the last bit it is maximized cost. And after that when subtracting by 1 at the last bit it is also maximized cost. So in any k bits and n operations, the assumption is proved.

2. Solution

Assume that the stack operation has the following three basic sub-operation, 1) push ; 2) pop; 3) copy, then multipop is very a combination of multiple times of pop operations, we could use that similar to 2).

operation	amortized cost
push	3
pop	0
copy	0

Now that we could assign the values as the above shows, so we give 3 to the operation, 1 for its actual cost, which is 1, another one for pop operation, as the credit for prepaid cost of poping, and another one for the copy operation, for the credit of future copying, because each element should have the chance or probability of copying to back up.

Since then, we can guarantee that each time we do the push operation, we could give enough credit for the future pop and copy operation. Our solution promise that that the amortized cost is never a negative value. And the total cost of n operations is thus 3n, using other format to represent is that is is linear O(n), the cost of per operation is O(n)/n = O(1).

Since amortized cost >= actual cost according the equation in 17.1 in the book, the upper bound is O(n) for actual cost of the operations in a n sequence operation.

So the conclusion is reached from the above statement.

3. Solution

a. From the requirement of question we could know that H doesn't know the sequence in advance, so we could only traverse the LinkList one by one, and know that the worst case for all these traverse is that each step we doesn't find the element. For example, the first element delta1, if the element is in the last position of the n element, then the cost is n; then the second element delta2, if the element is in the last second position of n element, then the cost is (n-1). So in worst-case, the total cost is n + (n-1) + … , as there are m elements in the access sequence, the number of adding is m, so the total's exact value is (n+n-m +1)m/2, as the m is const value we could ignore it comparing with n, so it is bounded to Omega(mn). So the conclusion is reached.

b. This question is to calculate the cost of visiting and transferring of element x, we could see that for delta(i), when we want to visit the element x, the only way is to visit from the beginning to the end until we find it, so the cost is rankL(x), because this value is the order of x in the list L. After that, we use **move-to-front** method, so we need to move x to the first position. By exchanging the neighborhood element, we could move it one step forward for each exchanges, and since it ranks rankL(x), there are rankL(x)-1 elements in the sequence, and it need to exchanges this times, the cost is rankL(x)-1. The total cost is:
   rankL(x) + rankL(x) -1 = 2 rankL(x) -1
   The conclusion is then proven.

c. According to b) we have the conclusion that ci=2 * rank(L-1)(x), then using another H instead of move-to-front, we know that this cost is the sum of visiting delta(i) and the

transposition during access it. It is known that in move-to-front situation t*i = rank(L-1)(x) - 1, and in this situation H, the new is rank(L*-1)(x), so the overall cost is the sum of accessing and transposition, and ci* = rank(L*-1)(x) + ti*.

d. Let's assume that the transposition is between element x and element y, then their order is changed after the transposition. We could know that there are two situations:
   a. x is before y —-> y is before x
   b. x is after y —-> y is after x

   No matter in what situation, the pi represents the order that inverse. Then if pair(x,y) before is one of the pair counting in pi, then after switch the pair number decrease by 1, and the potential decrease by 2; In another case the potential increase by 2, which adds a new pair in the inversion. So the conclusion is reached.

e. From the definition of A, B, C, D, we could know that in L(i-1), it constitute the AUB, x, CUD, since the definition of L(i-1), the total sequence is like:
   L(i-1) element order: AUB, x, CUD
   so the rankL(i-1)(x) = |A| + |B| + 1 (including the element x iteself, so adding one)
   similarly, for L*(i-1) sequence, the order is like:
   L*(i-1) order: AUC, x, BUD
   so the rankL*(i-1)(x) = |A| + |C| + 1
   And the conclusion is reached.

f. According to the above conclusion, we have:
   r = |A| + |B| + 1, r* = |A| + |C| + 1
   1) the MTF moves x to fronts, it creates |A| inversion, and destroy |B| inversion.
   2) each transposition by optimal create less than or equal to 1 inversion, thus
   f(Li) - f(Li-1) <= 2 * (|A| - |B| + ti)

g. According to last statement, the equation of amortize cost and actual cost is:
   c_amortized(i) = c(i) + f(Li) - f(Li-1)
   <= 2r + 2(|A| + |B| +ti)
   by using r = |A| + |B| + 1, we could change it to:
   = 2r + 2(|A| - (r-1-|A|) + ti)
   = 2r + 4|A| - 2r + 2 + 2ti
   = 4|A| + 2 + 2ti
   by using r* = |A| + |C| + 1 >= |A| + 1,
   <= 4(r* + ti)
   = 4Ci*

h. According to the above conclusion, we could know that
   C_mtf(s) = sum(Ci) <— from i to |s| calculate the Ci…
   = sum(Ci~ + f(Li-1) - f(Li))
   <= sum(4* Ci* + f(L0) - f(L|s|))
   <= 4 Copt(s)
   So the conclusion is reached.