# a4 CSC263

Tony and Andrew

March 2024

# 1

Let *Append* have a cost of 4 and *Delete* have a cost of 1.

**Credit invariant (CI)**: Before and after each operation, each element in the first quarter of the array has at least 2 credits and each element in the second half has at least 3 credits.

We prove the credit invariant by induction on n, the number of operations called. In conjunction with the proof of the credit invariant, we will simultaneously prove that the number of credits never depletes below zero.

## 1.1 Base Case: First Operation

Appending one to the array makes it so all elements currently in the array have $4 - 1 = 3$ credits. Thus, the credit invariant holds.

Deleting 0 elements leaves 0 elements so the invariant holds vacuously. We pay 1 to delete, but since no items are deleted, no charge is applied.

The number of credits does not drop below zero in either case.

## 1.2 Inductive Step

Assume for any set of $n-1$ operations, the credit invariant holds. Suppose $n-1$ operations have occurred (so the C.I holds for these elements). We want to show the credit invariant holds for the $n^{th}$ operation. There are multiple cases for both append and delete. Let $L$ be the current array after $n-1$ operations. It has length $m$, where $m$ is a power of 2.

**Append and no list expansion**

$L$ still has length $m$. Appending an element charges 1, but as we pay 4, the element ends up with a credit of 3. Regardless of where in L we are appending to (first quarter, second half, etc.), as the new element as 3 credits, C.I is maintained. The number of credits does not drop below zero in this case.

**Append and list expansion.**

Following the proof done in class, 2 credits from each element in the original second half of L (where there are $\frac{m}{2}$ of them) are used to move themselves and an element in the original first half to the expanded list of length $2m$. As they each originally had 3 by the C.I, they are now each left with 1 credit. We pool these combined $\frac{m}{2}$ credits, and set them aside for now. After list expansion, the second quarter of the original list becomes part of the first quarter of the new list. These elements currently hold an ambiguous number of credits, since the C.I doesn't cover them, and we need to ensure that they each have a minimum of 2 credits for the new C.I to be maintained. There are a total of $\frac{m}{4}$ of them, so we can use the pool of $\frac{m}{2}$ credits we set aside earlier to "allocate" them the necessary credits. Now, each element in the new list's first quarter has at least 2 credits.

Lastly, the newly appended element which caused the list expansion is added to be the first element in the second half of the new list with a credit of 3. Therefore, the CI holds. Also, the number of credits does not drop below zero for this entire case.

**Delete and no List Shrinkage**

Assuming no list shrinkage, we pay one to delete one element from $L$. No matter which position the element is at, this operation charges 1 and leaves 0 credit. The remaining elements stay the same in terms of credit and position. So the CI holds. Also, the number of credits does not drop below zero for this case.

**Delete and List Shrinkage**

A deletion followed by a list shrinkage implies that the old number of elements in the list prior to deletion is equivalent to the smallest integer $s$ greater than or equal to $\frac{m}{4}$ such that subtracting $s$ by 1 results in it being less than $\frac{m}{4}$.

Following this, we divide the proof into two cases:

Case 1: $m$ has size less than or equal to 4. Then the number of elements it holds prior to deletion is 1. The 1 credit we pay for the deletion covers the 1 credit cost it directly charges. Since there are no items to move to the new array after the deletion, the C.I vacuously holds.

Case 2: $m$ has size greater than or equal to 8. Then the number of elements it holds prior to deletion is exactly $\frac{m}{4}$. Therefore, only the first quarter of the array is occupied, and by the C.I., each of the elements have 2 credits. We pool these combined $\frac{m}{2}$ credits, and set them aside for now. The direct deletion of the last element charges 1 credit, which is covered by the 1 credit that we pay. It costs $\frac{m}{4} - 1$ credits to move the remaining elements into a new array half the size. We finance this via the credits we pooled earlier, leaving us with $\frac{m}{4} + 1$ credits. The elements from the second half of the first quarter of the old array are now in the second quarter of the new array; they do not need to be allocated credits to meet the C.I. As for the first half of the first quarter of the old array (now the first quarter in the new array), there are $\frac{m}{8}$ of them, and we can allocate two credits to each of them using the remaining pool of credits from earlier. There are no elements in the second half of the new array, and each of the elements in the first quarter of the new array have two credits, so the C.I is maintained. Also, the number of credits never depletes below zero.

Since m is a power of 2, there are no other cases we need to consider.

### 1.3 Finding the Upper Bound

For each of the cases in the inductive step, the credit invariant holds and the credit never depletes below zero. So, charging these costs on an arbitrary sequence of append and delete operations provides an upper bound. We demonstrated that the actual cost for each operation in all cases is less than the amortized charge. So, for any sequence of $m$ operations, the sum of the actual cost of the ith operation (each operation) $c_i$,where i takes on natural values between 1 and m inclusive, will be lower than the cost for the sum of each amortized operation $a_i$. Thus, for the representation of the actual cost $T_m^{sq}$,

$$T_m^{sq} = \sum_{i=1}^{m} c_i < \sum_{i=1}^{m} a_i < \sum_{i=1}^{m} (4) = 4m$$

To find the upper bound on the amortized sequence complexity we find the upper bound on $T_m/m$.
Thus, $\frac{T_m}{m} = \frac{4m}{m} = 4 \in O(1)$
Therefore, an upper bound on the amortized complexity is $O(1)$.

# 2

On initial observation, we see that the total cost for $n$ push operations onto an initially empty ExpoStack can be divided into two subsets. The first is the direct cost of pushing the new items onto $S_0$. The second is the cost of all the pop-push calls from $S_i$ to $S_{i+1}$, where $i$ is a natural number, needed to make space for the new items. The direct cost of all the pushes of each new item onto $S_0$ (not considering the downstream pop/push relocation costs onto $S_1$, $S_2$, etc.) is $m$ (costs 1 per push).

Finding the cost of all the pop-push calls is a little more complicated. We first find the number of pop-push calls from $S_i$ to $S_{i+1}$ individually for each $i$. Via the definition of an ExpoStack, we know that pop-push relocations from $S_i$ to $S_{i+1}$ only occur when $S_i$ is full. Therefore, each instance of pop/push relocations from $S_i$ to $S_{i+1}$ results in the pop-pushes of $2^i$ elements. Additionally, after the first instance of pop-push relocations from a given $S_i$ to $S_{i+1}$, each subsequent instance of pop-push relocations for that stack will occur once every $2^i$ push operations (this can be proved fairly simply by induction). Another observation that can be proved from induction is that the first time that an instance of pop-push relocations occurs from a stack $S_i$ is when the $2^{i+1}$th element is pushed to the the beginning of the ExpoStack. These facts directly pertain to elements of the equation we will display below.

Considering all of the observations in the above paragraph, for n push operations, the stack $S_i$, where i $\in$ 0, ..., $\lfloor log_2 n \rfloor - 1$, will have $\lfloor \frac{n}{2^i} \rfloor - 1$ instances of pop-push relocations, where each relocation involves the movement of $2^i$ elements. Considering the fact that each pop-push relocation costs 2, we obtain the following subtotal cost $R$ by summing the pop-push relocation costs of each $S_i$:

$$R_n = 2 \sum_{i=0}^{\lfloor \log_2(n) \rfloor - 1} \left( \left\lfloor \frac{n}{2^i} \right\rfloor - 1 \right) \cdot 2^i$$

Considering the cost of $n$ needed for direct pushes introducing new elements to the beginning of the ExpoStack, we arrive at the final summation $T_n$:

$$T_n = n + 2 \sum_{i=0}^{\lfloor \log_2(n) \rfloor - 1} \left( \left\lfloor \frac{n}{2^i} \right\rfloor - 1 \right) \cdot 2^i$$

The 1 accounts for the first expo push operation that does not cause a push-pop pair interaction. The sum accounts for the number and the cost of push-pop pair interactions that occur for each $S_i$. Multiplying by 2 accounts for there being 2 push-pop pairs that occur each time. $2^i$, the cost of moving all the elements from $S_i$ to $S_(i+1)$ is multiplied by

So, if the cost of $T_n$ is as stated, we can deduce:

$$T_n = n + 2 \sum_{i=0}^{\lfloor \log_2(n) \rfloor - 1} \left( \left\lfloor \frac{n}{2^i} \right\rfloor - 1 \right) \cdot 2^i$$

$$T_n \leq n + 2 \sum_{i=0}^{\lfloor \log_2(n) \rfloor - 1} \frac{n}{2^i} \cdot 2^i$$

$$\leq n + 2 \sum_{i=0}^{\lfloor \log_2(n) \rfloor} n$$

$$\leq n + 2n \cdot \log_2(n)$$

To find the upper bound on the amortized sequence complexity we find the upper bound on $T_n/n$.
$\frac{T_n}{n} \leq \frac{n+2n\log_2(n)}{n} = 1 + 2log_2(n)$.
So, $\frac{T_n}{n} \in O(\log_2(n))$
Therefore, an upper bound on the amortized complexity is $\log(n)$.

# 3

First, we establish the upper bound number of pairwise comparisons made for `Diminsh`$(S)$, where $|S| = n$ for some $n \in \mathbb{N}$. As stated in the handout, step 1 calls `Med`$(S)$, which makes at most $5n$ pairwise comparisons. Step 2 involves looping through all the elements in the list and checking to see if they are greater than the median; this involves exactly $n$ pairwise comparisons. Step 3 does not involve any pairwise comparisons, nor does step 4. So the upper bound number of pairwise comparisons made is $6n$.

## 3.1 Credit Invariant

Assuming we start from an empty linked list and charge 12 credits for insertion and 0 for diminish operations, we posit that after each operation, each element in the list has a credit of 12.

## 3.2 Proof by Induction

Now, we will prove the credit invariant by induction on $k$, the number of operations performed. In conjuction we will will also prove that the number of credits in the list never goes below 0.

**Base case:** Let $k = 1$.

**Case 1:** It is an insertion. We pay 12 credits for the insertion, and there are no charges (since no pairwise comparisons are made). So the only element in the list has 12 credits, and the C.I. is maintained. Also, the number of credits does not deplete below 0.

**Case 2:** It is a `Diminish` call. Since the list is empty (no previous calls) or pairwise comparisons are made, and since there are no elements, the C.I. is vacuously true. Also, the number of credits does not deplete below 0.

**Inductive step:** Let $k \in \mathbb{N}^{\geq 2}$, and assume for all series of less than $k$ operations, the C.I. is upheld. Consider an arbitrary series of $k-1$ operations, assuming we are performing the $k^{\text{th}}$ operation:

**Case 1:** It is an insertion. We pay 12 credits for the insertion, and there is no cost (no pairwise comparisons). By the I.H., all the rest of the elements have at least 12 credits. Since the newly inserted element also has 12 credits, the C.I. is maintained. Also, the number of credits does not deplete below 0.

**Case 2:** It is a Diminish Call. Let $n$ be the number of elements initially in the list. These elements store a total of $12n$ credits. The diminish call costs at most $6n$, so we are left with at least $6n$ credits at the end. This can be distributed among the remaining $\lfloor \frac{n}{2} \rfloor$ elements on the list such that each element has at least 12 credits. Thus, the C.I. holds. Also, the number of credits does not deplete below 0.

## 3.3 Finding the upper bound

Since the number of credits in the list never becomes negative, and the amortized cost for each operation $\leq 12$, we see:

$$T_{\text{k}}^{sq} = \sum_{i=1}^{k} C_i \leq \sum_{i=1}^{k} a_i \leq \sum_{i=1}^{k} 12 = 12k$$

Thus, the amortized complexity of each operation (i.e., $\frac{T_{\text{k}}^{sq}}{k}$) is constant.