CS3230 Tutorial 9

Amortized Analysis

(AY 25/26 Semester 2)

March 23, 2026

(Prepared by Benson and Hua Jun)

Contents

Recap: Amortized Analysis

Tutorial Questions: Accounting Method
- Q1. MCQ
- Q2. Dynamic Table Insert
- Q3. A queue with add and delete

Tutorial Questions: Potential Method
- Q4. A queue with add and delete
- Q5. Dynamic Table with only Deletions
- Q6. Dynamic Table with Insertions and Deletions

Bonus. Tree Puzzle (Part 2)

Recap: Amortized Analysis

- A strategy for analyzing a **sequence of operations** to show that the average cost per operation is small, even though a single operation within the sequence may be expensive. Focuses on the **worst case** (there is a guarantee).
- **NOT** calculating the expected value of the algorithm's runtime.

Recap: Amortized Analysis

A toy example:

Algorithm A toy stack with two operations

```
1:  S ← An empty stack
2:  procedure PUSH(x)                          ▷ O(1) time
3:      S.push(x)
4:  procedure MULTIPOPPUSH(N)                  ▷ O(N) time
5:      T ← An empty stack
6:      for i ← 1 to N do
7:          if S is empty then
8:              break
9:          T.push(S.pop())
10:     while T is not empty do
11:         S.push(T.pop())
```

Poll: We execute Q operations. What is the **average** runtime **per operation in the worst case**?

A. $O(1)$

B. $O(\log N)$

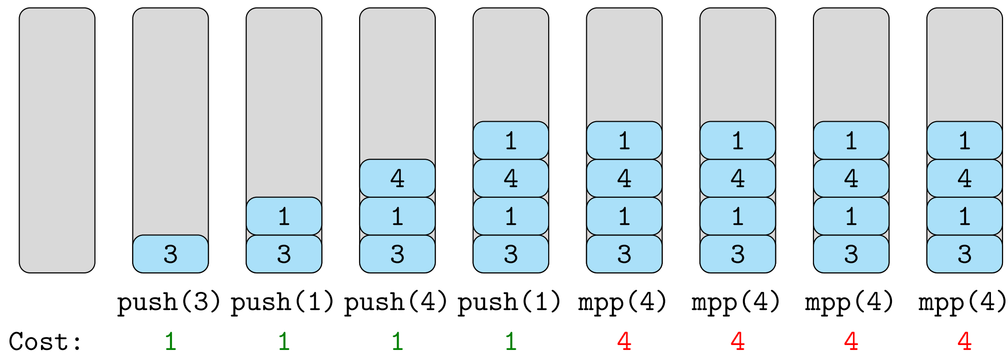
C. $O(N)$

D. $O(Q)$

E. $O(QN)$

Solution. If we call PUSH $\frac{Q}{2}$ times and then call MULTIPOPPUSH $\frac{Q}{2}$ times, the total runtime is $O(Q^2)$.

Recap: Amortized Analysis



Recap: Amortized Analysis

Another toy example:

Algorithm A toy stack with two operations

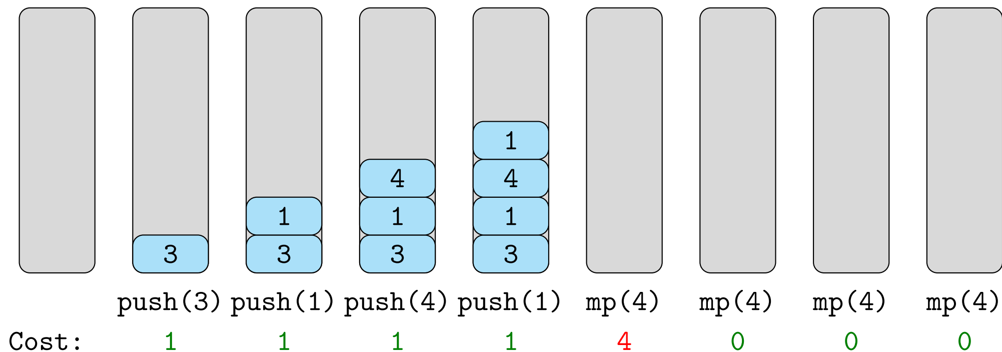
```
1: S ← An empty stack
2: procedure PUSH(x)                    ▷ O(1) time
3:     S.push(x)
4: procedure MULTIPOP(N)                ▷ O(N) time
5:     for i ← 1 to N do
6:         if S is empty then
7:             break
8:         S.pop()
```

Poll: We execute Q operations. What is the **average** runtime **per operation in the worst case**?

A. $O(1)$
B. $O(\log N)$
C. $O(N)$
D. $O(Q)$
E. $O(QN)$

Solution. We need to push some elements before we can pop them. So the total runtime is always $O(Q)$.

Recap: Amortized Analysis



Recap: Amortized Analysis

We usually have some **cheap** operations and some **expensive** operations as we do Amortized Analysis.

Operation	Actual Cost		Amortized Cost
PUSH	1		◎ $O(1)$
MULTIPOP	$\min(\|S\|, N)$		◎ $O(1)$

Actual cost is already $O(1)$
$\rightarrow$ **cheap operation**

Actual cost is higher than $O(1)$
$\rightarrow$ **expensive operation**

Recap: Aggregate Method

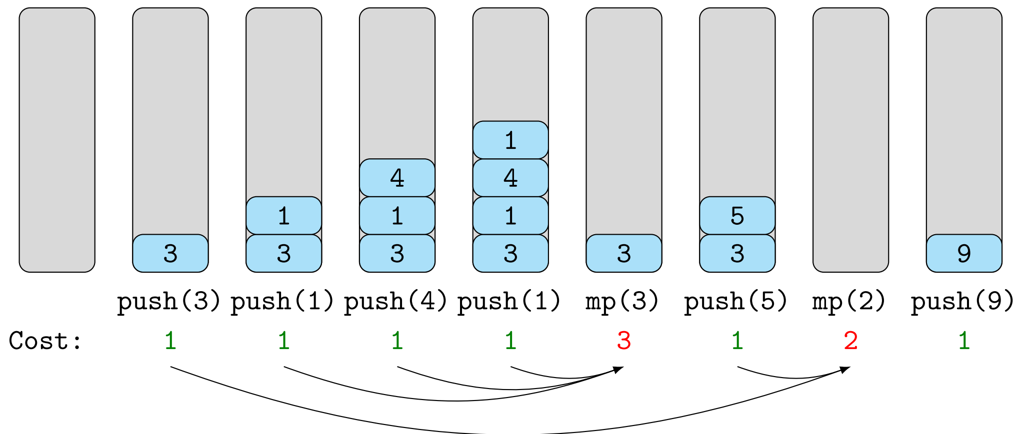
Aggregate method: Calculate the sum of costs of the Q operations, divided by Q.

- What exactly are the Q operations? **We must find the worst case**.
- Hard to calculate and prove...
- **Note**: This method is the best if the sequence is already predefined.

Operation	Actual Cost		
PUSH	1		
MULTIPOP	$\min(\lvert S\rvert, N)$		

Recap: Accounting Method

Accounting method (Intuition):



Recap: Accounting Method

Accounting method: "Charge" the cheap operation an extra cost to be stored in a bank. Use the funding to subsidize the expensive operations.

- Need to ensure: The balance in the bank **never goes negative**.

Operation	**Actual Cost** c_i	**Bank**	**Charge** $\hat{c}_i$
PUSH	1	1	2
MULTIPOP	$\min(\lvert S\rvert, N)$	$-\min(\lvert S\rvert, N)$	0

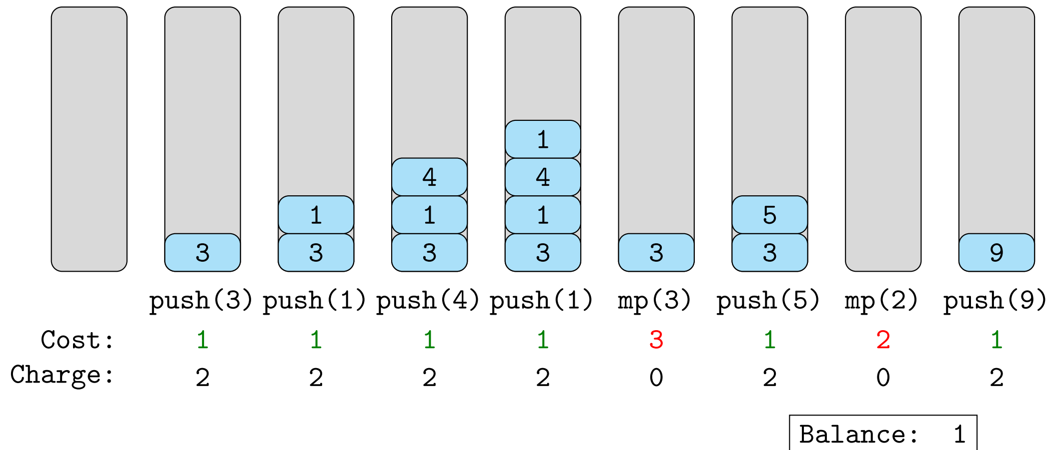
We need some funding from the bank to subsidize this expensive operation.

How much do we have to deposit?

We need $1 to pop it later

Recap: Accounting Method

Accounting method in action:



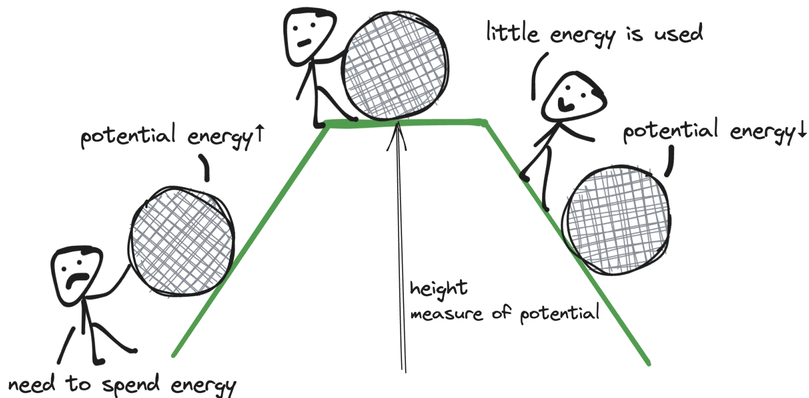
Recap: Accounting Method

How to figure out the amount to give/take from bank?

- Try to deal with expensive operation first, try to take as much from the bank as you need to get the intended amoritsed complexity.
- E.g. Actual Cost: $n + 1$, Expected amortized cost: $O(1)$, we can take n from the bank to make it constant.
- Then try to make cheap operations pay to the bank such that the bank is never empty. ($+1$, $+2$, $+\log n$, etc.)

Recap: Potential Method

The idea of Potential Analysis is the same as Physics' potential energy.



Recap: Potential Method

Potential method: As we process the cheap operations, we accumulate potential. As we process the expensive operations, we release potential.

- The potential function is usually formed by some **internal states** of the data structure. This avoids the "artificial" charges.
- Need to ensure: (1) Initially, the potential is 0. (2) The potential is always non-negative at any instant.

Main Idea: Find something which **decreases a lot** after performing the expensive operation!

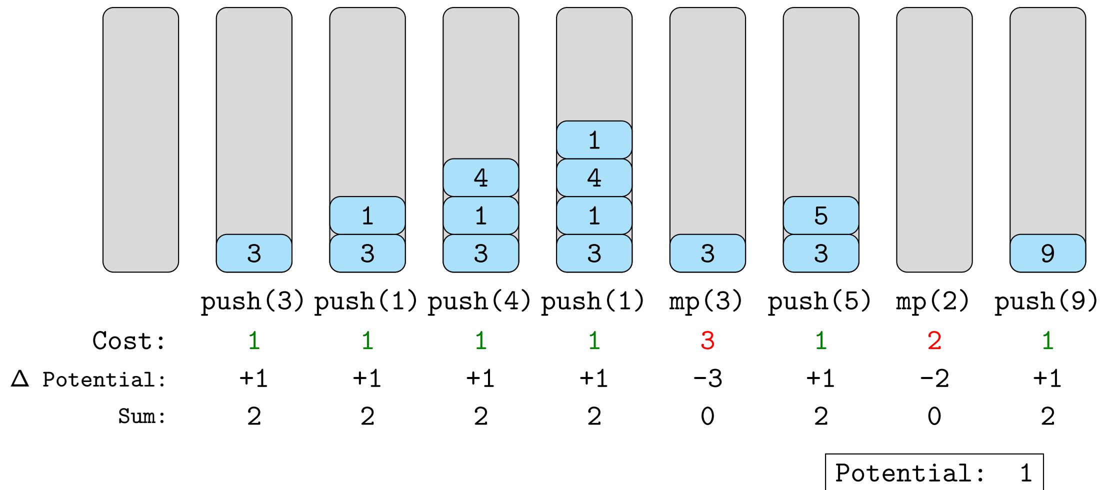
Potential function: $\phi(i) = |S|$.

$|S|$ accumulates during PUSH and decreases a lot after MULTIPOP.

Operation	**Actual Cost**	$\phi(i) - \phi(i-1)$	**Amortized Cost**
PUSH	1	1	2
MULTIPOP	$\min(\lvert S\rvert, N)$	$-\min(\lvert S\rvert, N)$	0

Recap: Potential Method

Potential method in action:



Q1. MCQ

Which of the following statements is **false**?

A. The amortized cost for insert in dynamic tables is $\Theta(1)$.

B. In the accounting method, the amortized cost $\hat{c}_i$ is always greater than the actual cost c_i of an operation.

C. $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i \geq 0$ where $\hat{c}_i$ and c_i are the amortized and actual costs of the i-th operation respectively.

Solution.

A. See Q2.

B. The amoritized cost $\hat{c}_i$ is smaller than the actual cost c_i if we take money from the bank.

C. $\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i +$ final bank balance. The bank balance is never negative.

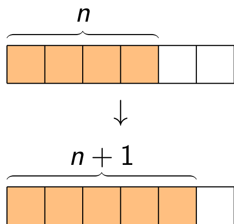
Q2. Dynamic Table Insert

Prove that insertion in Dynamic Table is amortized $O(1)$ using Accounting Method!

Q2. Dynamic Table Insert

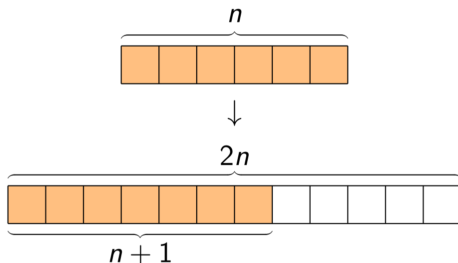
When inserting in dynamic tables, there are two cases:

INSERT (when table is not full):



Actual cost: 1 (cheap)

INSERT (when table is already full):



Actual cost: $n+1$ (expensive)

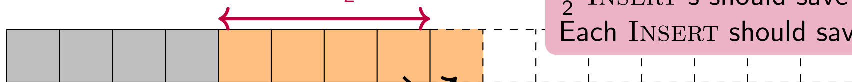
Q2. Dynamic Table Insert

Amortized cost for insert in dynamic tables:

Operation	Actual Cost	Bank	Nett
INSERT (when table is not full)	1	2	3
INSERT (when table is already full)	$n+1$	$-n$	1

We need some funding from the bank to turn the cost $n+1$ into $O(1)$.

How much do we have to deposit?



Q3. A queue with add and delete

Consider a data structure that is based on a queue with four operations:

- $\textsc{Enqueue}(a)$: Add the element a into the queue
- $\textsc{Dequeue}()$: Dequeue a single element from the queue
- $\textsc{Delete}(k)$: Dequeue k elements from the queue
- $\textsc{Add}(A)$: Enqueue all elements in A

Using **accounting method**, show that $\textsc{Enqueue}$, $\textsc{Dequeue}$ and $\textsc{Delete}$ run in amortized $O(1)$ time while $\textsc{Add}$ runs in amortized $O(|A|)$ time.

(Please state the charge for each operation.)

Q3. A queue with add and delete

Actual cost is already $O(|A|)$
$\rightarrow$ **cheap operation**

Actual cost is higher than $O(1)$
$\rightarrow$ **expensive operation**

Operation	**Actual Cost**	**Bank**	**Nett**
$\textsc{Enqueue}(a)$	1	1	2
$\textsc{Dequeue}()$	1	-1	0
$\textsc{Delete}(k)$	k	$-k$	0
$\textsc{Add}(A)$	$\|A\|$	$\|A\|$	$2\|A\|$

Q3. A queue with add and delete

◎ The balance in the bank never goes negative.

Operation	Actual Cost	Bank	Nett
ENQUEUE(a)	1	1	2
DEQUEUE()	1	-1	0
DELETE(k)	k	$-k$	0
ADD(A)	$\|A\|$	$\|A\|$	$2\|A\|$

- After the insertion of element x (in ENQUEUE and ADD operations), \$1 is associated to x in the bank.
- When we dequeue the element x (in DEQUEUE and DELETE operations), we can use \$1 from the bank for dequeue of x.
- Hence, the balance in the bank never goes negative.

Q4. A queue with add and delete

Consider a data structure that is based on a queue with four operations:

- $\textsc{Enqueue}(a)$: Add the element a into the queue
- $\textsc{Dequeue}()$: Dequeue a single element from the queue
- $\textsc{Delete}(k)$: Dequeue k elements from the queue
- $\textsc{Add}(A)$: Enqueue all elements in A

Using **potential method**, show that $\textsc{Enqueue}$, $\textsc{Dequeue}$ and $\textsc{Delete}$ run in amortized $O(1)$ time while $\textsc{Add}$ runs in amortized $O(|A|)$ time.

(Please state your potential function.)

Q4. A queue with add and delete

Let $\phi(i)$ be the number of elements in the queue after the i-th operation.

$\textsc{Delete}(k)$ is the expensive operation, after which the number of elements in the queue decreases a lot.

◎ Check that $\phi(0) = 0$ and $\phi(i) \geq 0$.

- The queue is initially empty, so $\phi(0) = 0$.
- The number of elements in the queue is always non-negative, so $\phi(i) \geq 0$.

Q4. A queue with add and delete

Let $\phi(i)$ be the number of elements in the queue after the i-th operation.

Operation	**Actual Cost**	$\phi(i) - \phi(i-1)$	**Amortized Cost**
$\textsc{Enqueue}(a)$	1	1	2
$\textsc{Dequeue}()$	1	-1	0
$\textsc{Delete}(k)$	k	$-k$	0
$\textsc{Add}(A)$	$\lvert A\rvert$	$\lvert A\rvert$	$2\lvert A\rvert$

Therefore, Enqueue, Dequeue and Delete run in amortized $O(1)$ time while Add runs in amortized $O(|A|)$ time.

Q5. Dynamic Table with only Deletions

Algorithm Deletion from a dynamic table

```
1: procedure Delete()                ▷ worst case O(n) time
2:     n ← n − 1
3:     if n = 0 then
4:         Free(T)
5:     else
6:         if n = Size(T) /2 then
7:             T' ← CreateTable(n)
8:             Copy(T, T')
9:             Free(T)
10:            T ← T'
```

T is the dynamic table that supports only deletions.

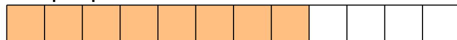
Use the potential method to show that the amortized cost of each DELETE() operation is $O(1)$.

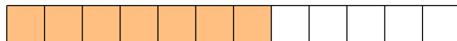
Q5. Dynamic Table with only Deletions

Algorithm Deletion from a dynamic table

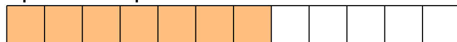
```
1:  procedure DELETE()                ▷ worst case O(n) time
2:      n ← n − 1
3:      if n = 0 then
4:          FREE(T)
5:      else
6:          if n = SIZE(T)/2 then
7:              T′ ← CREATETABLE(n)
8:              COPY(T, T′)
9:              FREE(T)
10:             T ← T′
```

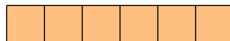
Cheap operation:





Expensive operation:





Q5. Dynamic Table with only Deletions

Let $\phi(i)$ be the number of empty slots in the table ($\phi(i) = \text{Size}(T) - n$).

After $\text{Delete}(k)$ where the table shrinks to half (the expensive operation), the number of empty slots decreases drastically to 0.

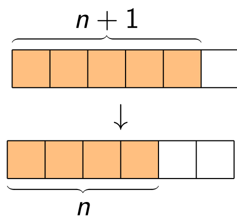
◎ Check that $\phi(0) = 0$ and $\phi(i) \geq 0$.

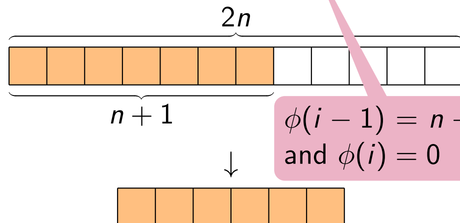
- The table is initially full, so $\phi(0) = 0$.
- The number of empty slots is always non-negative, so $\phi(i) \geq 0$.

Q5. Dynamic Table with only Deletions

Let $\phi(i)$ be the number of empty slots in the table ($\phi(i) = \text{SIZE}(T) - n$).

Operation	Actual Cost	$\phi(i) - \phi(i-1)$	Amortized Cost
DELETE (when table does not shrink)	1	1	2
DELETE (when table shrinks to half)	$n+1$	$1-n$	2





Q6. Dynamic Table with Insertions and Deletions

Extra Slide

Design a dynamic table that supports amortized $O(1)$ insertions and deletions.

Q6. Dynamic Table with Insertions and Deletions

Extra Slide

An attempt:

Algorithm Insertion to a dynamic table

```
1: procedure INSERT(x)                ▷ worst case O(n) time
2:     if n = SIZE(T) then
3:         T′ ← CREATETABLE(2n)
4:         COPY(T, T′)
5:         FREE(T)
6:         T ← T′
7:     n ← n + 1
8:     T[n] ← x
```

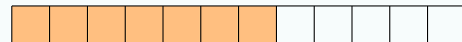
Algorithm Deletion from a dynamic table

```
1: procedure DELETE()                 ▷ worst case O(n) time
2:     n ← n − 1
3:     if n = 0 then
4:         FREE(T)
5:     else
6:         if n = SIZE(T) /2 then
7:             T′ ← CREATETABLE(n)
8:             COPY(T, T′)
9:             FREE(T)
10:            T ← T′
```

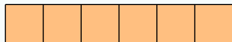
Poll: What are your thoughts on this?

Q6. Dynamic Table with Insertions and Deletions

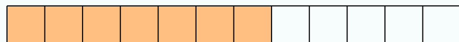
Extra Slide



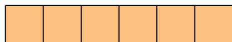
↓ (DELETE)



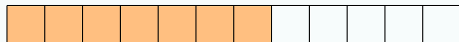
↓ (INSERT)



↓ (DELETE)



↓ (INSERT)



⋮

Each operation incurs a cost of $1 + n$.

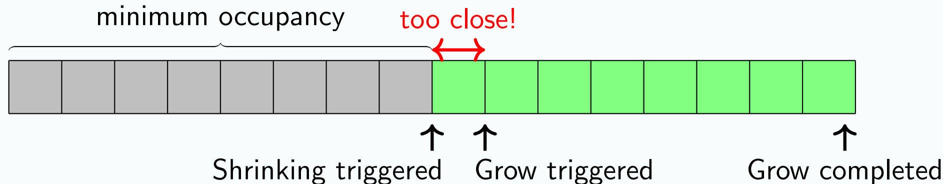
Remember: We're only interested in the **worst-case** scenario.

Q6. Dynamic Table with Insertions and Deletions

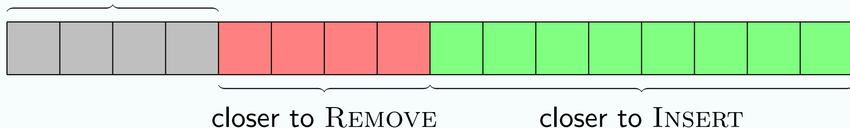
Extra Slide

What went wrong here?

- Observe that we're allowing DELETE and INSERT operations to happen close to each other.



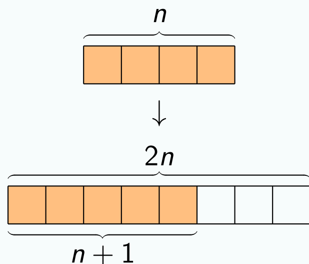
- **Solution**: Halve the table size **only when** $n \leq \frac{\textsc{Size}(T)}{4}$.



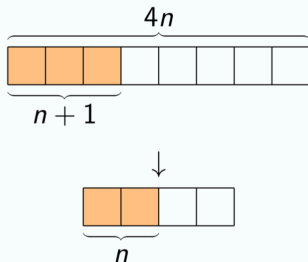
Q6. Dynamic Table with Insertions and Deletions

Halve the table size **only when** $n \leq \dfrac{\textsc{Size}(T)}{4}$, but the growing policy stays the same.

Growing:



Shrinking:

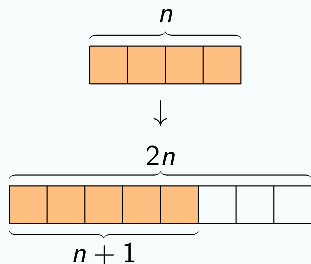


Q6. Dynamic Table with Insertions and Deletions

Extra Slide

Recap:

- If we only have $\textsc{Insert}$ operations, the only expensive operation is when the table grows, where $-$(number of empty spaces) decreases a lot.
- A candidate potential function is $\phi(i) = n - \textsc{Size}(T)$. Unfortunately, it doesn't satisfy the requirement $\phi(i) \geq 0$.
- **Idea**: Since the table is at least half-full, we can add $\frac{Size(T)}{2}$ to $\phi(i)$, i.e. $\phi(i) = n - \frac{Size(T)}{2}$. (Note that $\phi(i)$ still decreases a lot when growing.)
- $\phi(i) = n - \frac{Size(T)}{2}$ can be seen as the **number of entries from the half-way mark of T**.

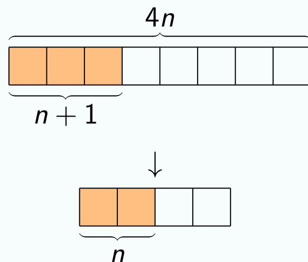


Q6. Dynamic Table with Insertions and Deletions

Extra Slide

Recap:

- If we only have $\textsc{Delete}$ operations, the only expensive operation is when the table shrinks, where the number of empty spaces decreases a lot.
- So we chose $\phi(i) = \textsc{Size}(T) - n$.
- Under the new shrinking policy, we only shrink at size $\frac{\textsc{Size}(T)}{4}$. Hence we can use $\phi(i) = \frac{\textsc{Size}(T)}{2} - n$ instead.

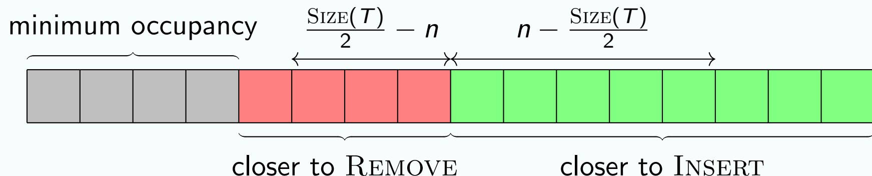


Q6. Dynamic Table with Insertions and Deletions

Extra Slide

Unfortunately, we now have **two** expensive operations: Insert (when table is already full) and Delete (when table shrinks to half).

- Insert: $\phi(i) = n - \frac{\textsc{Size}(T)}{2}$
- Remove: $\phi(i) = \frac{\textsc{Size}(T)}{2} - n$

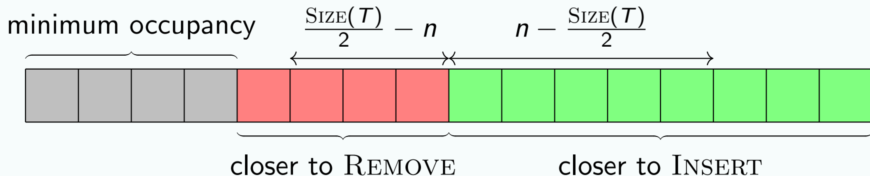


Q6. Dynamic Table with Insertions and Deletions

Extra Slide

Let's use $\phi(n) = |\textsc{Size}(T) - 2n|$.

- When $\frac{\textsc{Size}(T)}{4} \le n \le \frac{\textsc{Size}(T)}{2}$ (about to shrink), $\phi(n) = \textsc{Size}(T) - 2n$.
- When $n > \frac{\textsc{Size}(T)}{2}$ (about to grow), $\phi(n) = 2n - \textsc{Size}(T)$.
- Perfect!



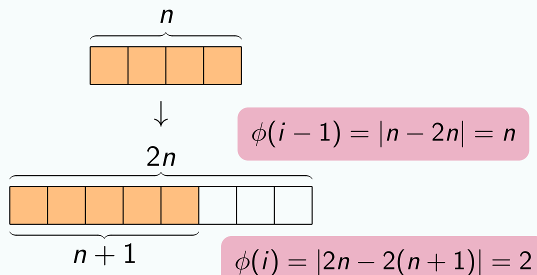
Q6. Dynamic Table with Insertions and Deletions

Extra Slide

Let $\phi(n) = |\textsc{Size}(T) - 2n|$.

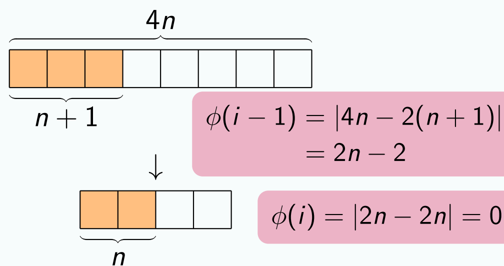
n changes by 1 so $|\textsc{Size}(T) - 2n|$ changes by at most 2.

Operation	Actual Cost	$\phi(i) - \phi(i-1)$	Amortized Cost
Insert (when table is not full)	1	≤ 2	≤ 3
Insert (when table is already full)	n	$2 - n$	2
Delete (when table does not shrink)	1	≤ 2	≤ 3
Delete (when table shrinks to half)	$n+1$	$2 - 2n$	≤ 3



$\phi(i-1) = |n - 2n| = n$

$\phi(i) = |2n - 2(n+1)| = 2$



$\phi(i-1) = |4n - 2(n+1)|$
$= 2n - 2$

$\phi(i) = |2n - 2n| = 0$

Bonus. Tree Puzzle (Part 2)

Extra Slide

There is an (unknown) labeled tree with N vertices numbered $1, 2, \ldots, N$. Your are tasked to find the structure of this tree. Luckily, you are given an oracle which gives you a Yes/No answer to the following question (you may specify any X and Y):

- Given two *disjoint* sets X and Y. Does there exist $x \in X$ and $y \in Y$ such that the vertices x and y are connected by an edge?

Assume you are now given one pre-order traversal of the tree to start with (for free!), what is the minimum number of questions (asymptotically) needed to complete this task? Prove both the lower and upper bounds.

(Hint: Read the solution to Part 1 first.)

Bonus. Tree Puzzle (Part 2)

Extra Slide

Solution: The minimum number of questions needed is $O(N)$.
Lower Bound:

- By Cayley's formula, there are N^{N-2} distinct labeled trees with N vertices.
- There are $N!$ possible pre-order traversals.
- Each tree has at least one pre-order traversal. Hence, on average, each pre-order traversal corresponds to $\geq \frac{N^{N-2}}{N!}$ trees.
- By a probabilistic argument, there exists at least one pre-order traversal corresponding to $\geq \frac{N^{N-2}}{N!}$ trees.
- Using the decision model, we need at least $\log_2 \frac{N^{N-2}}{N!} = \log_2 N^{N-2} - \log_2 N!$ $= (N-2)\log N - N \log N + N - O(\log N) = N - O(\log N) = \Omega(N)$ queries.

Bonus. Tree Puzzle (Part 2)

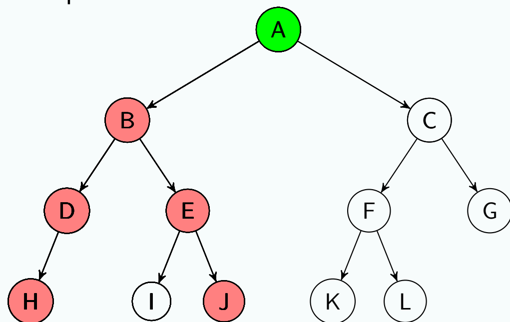
Algorithm (Upper Bound):

- We start with a tree with one vertex (first vertex in the pre-order traversal) and insert the remaining vertices one by one.
- When inserting a vertex, starting from the latest inserted vertex, we query up the tree hierarchy to find the parent of that vertex.
- Each insertion is amortized $O(1)$ – The total number of queries is bounded by
$$\underbrace{N-1}_{\text{negative queries}} + \underbrace{N-1}_{\text{positive queries}} = 2N-2.$$

Bonus. Tree Puzzle (Part 2)

Extra Slide

Example:



Pre-Order Traversal:
A, B, D, H, E, I, J, C, F, K, L, G