Basic Data Structures: Dynamic Arrays and Amortized Analysis

Outline

1. Dynamic Arrays

2. Amortized Analysis—Aggregate Method

Problem: static arrays are static!

```
int my_array[100];
```

Problem: static arrays are static!

```
int my_array[100];
```

Semi-solution: dynamically-allocated arrays:

```
int *my_array = new int[size];
```

cin>>size;
new int[size]

Problem: might not know max size when allocating an array

> *All problems in computer science can be solved by another level of indirection.*

Solution: *dynamic arrays* (also known as *resizable arrays*)

Idea: store a pointer to a dynamically allocated array, and replace it with a newly-allocated array as needed.

Definition

Dynamic Array:

An abstract data type is defined by its behavior from the point of view of a user

Abstract data type with the following operations (at a minimum):

- `Get`(i): returns element at location i^*
- `Set`(i, val): Sets element i to val^*
- PushBack(val): Adds val to the end
- `Remove`(i): Removes element at location i
- `Size`(): the number of elements

---

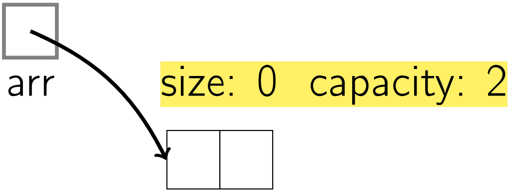
*must be constant time

Implementation

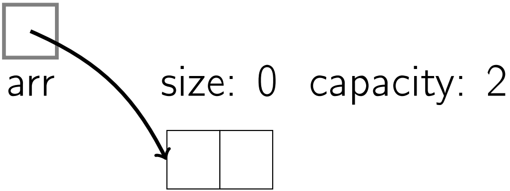
Store:

- `arr`: dynamically-allocated array
- `capacity`: size of the dynamically-allocated array
- `size`: number of elements currently in the array

Dynamic Array Resizing

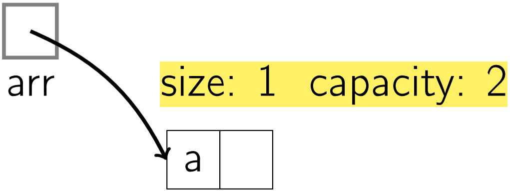


Dynamic Array Resizing



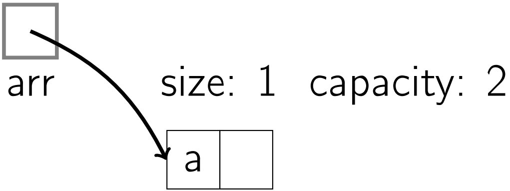
`PushBack(a)`

Dynamic Array Resizing

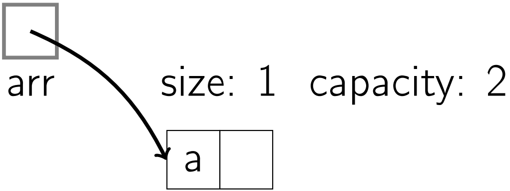


`PushBack(a)`

Dynamic Array Resizing

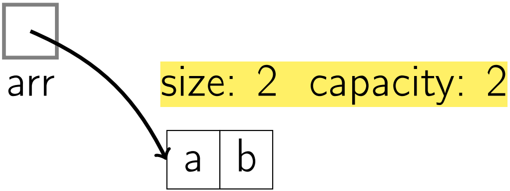


Dynamic Array Resizing



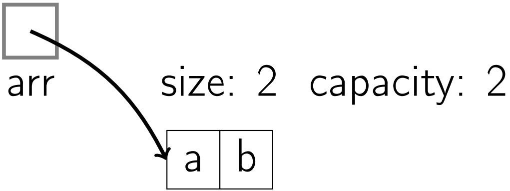
`PushBack(b)`

Dynamic Array Resizing

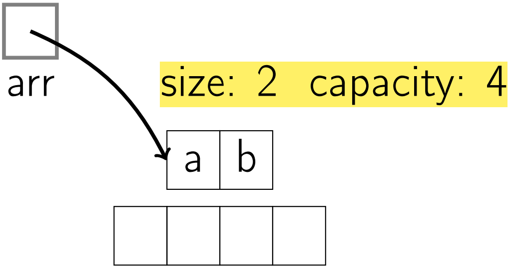


`PushBack(b)`

Dynamic Array Resizing

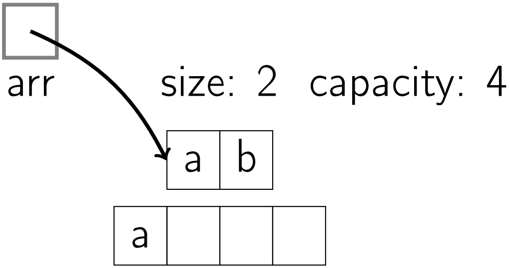


Dynamic Array Resizing



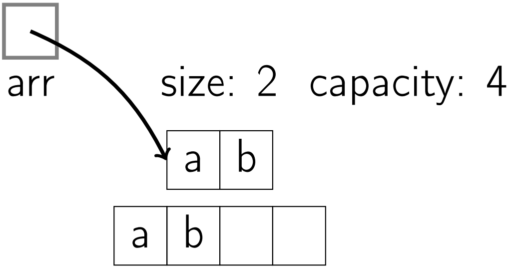
`PushBack(c)`

Dynamic Array Resizing



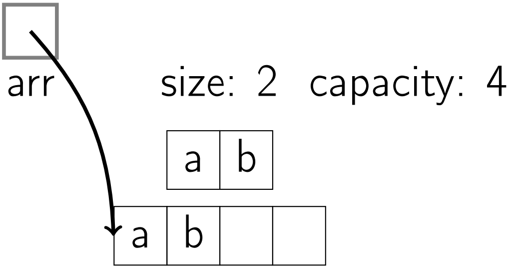
`PushBack(c)`

Dynamic Array Resizing



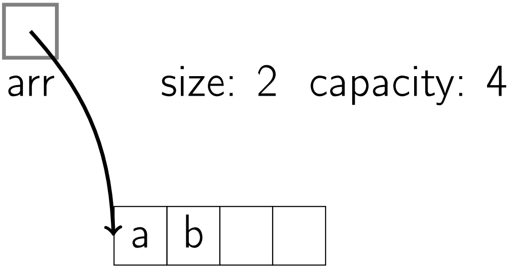
`PushBack(c)`

Dynamic Array Resizing



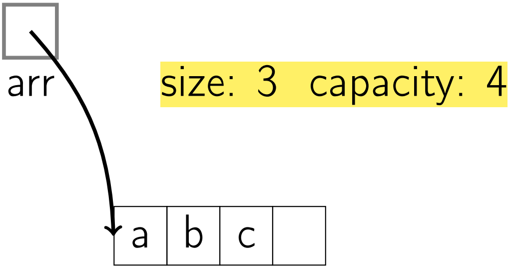
`PushBack(c)`

Dynamic Array Resizing



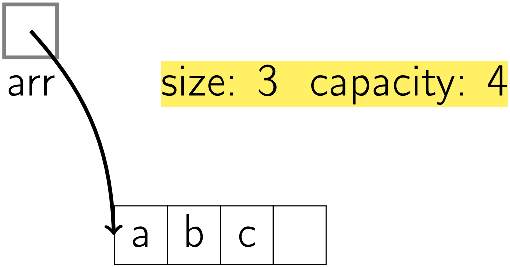
`PushBack(c)`

Dynamic Array Resizing

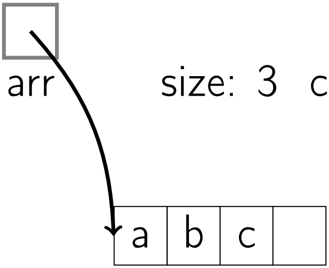


`PushBack(c)`

Dynamic Array Resizing



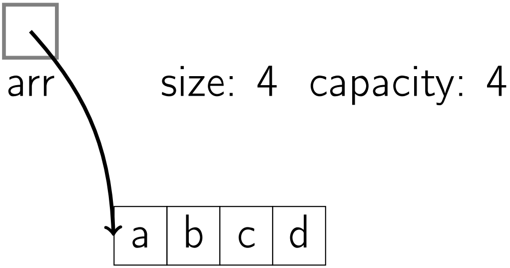
Dynamic Array Resizing



size: 3 capacity: 4

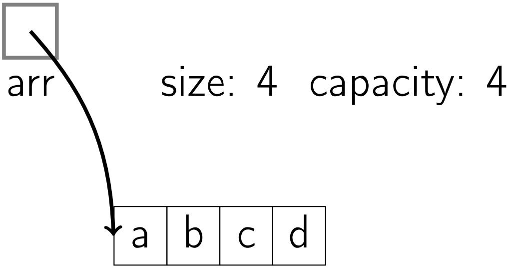
`PushBack(d)`

Dynamic Array Resizing

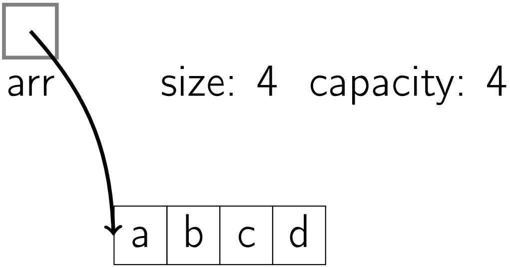


`PushBack(d)`

Dynamic Array Resizing

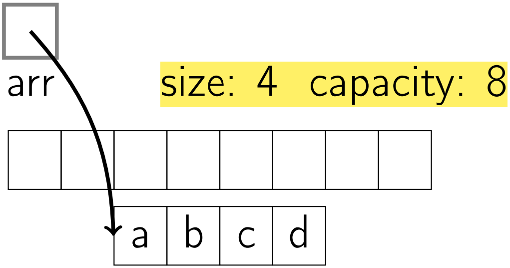


Dynamic Array Resizing



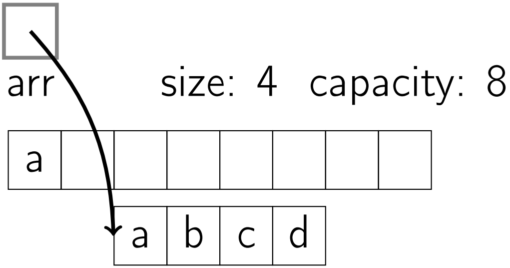
`PushBack(e)`

Dynamic Array Resizing



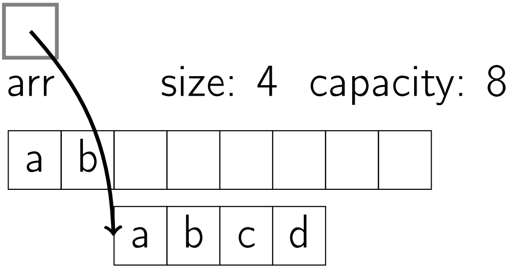
`PushBack(e)`

Dynamic Array Resizing



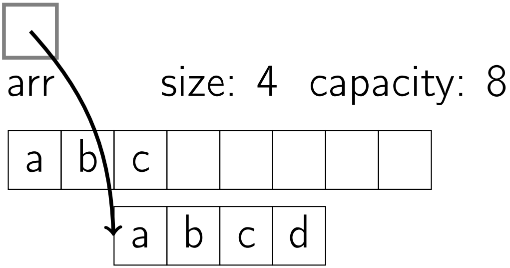
`PushBack(e)`

Dynamic Array Resizing



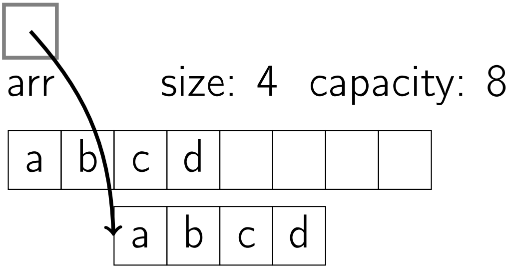
`PushBack(e)`

Dynamic Array Resizing



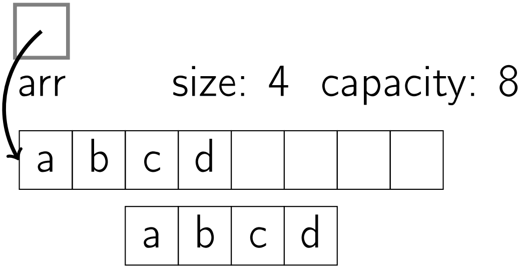
`PushBack(e)`

Dynamic Array Resizing



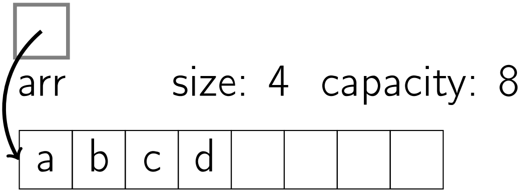
`PushBack(e)`

Dynamic Array Resizing



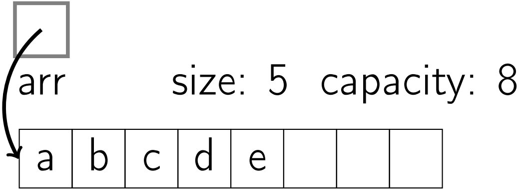
`PushBack(e)`

Dynamic Array Resizing



`PushBack(e)`

Dynamic Array Resizing



`PushBack(e)`

```
Get(i)

if i < 0 or i ≥ size:
  ERROR: index out of range
return arr[i]
```

$\texttt{Set}(i, val)$

```
if i < 0 or i ≥ size:
  ERROR: index out of range
arr[i] = val
```

PushBack(val)

```
if size = capacity:
    allocate new_arr[2 × capacity]
    for i from 0 to size − 1:
        new_arr[i] ← arr[i]
    free arr
    arr ← new_arr; capacity ← 2 × capacity
arr[size] ← val
size ← size + 1
```

Remove(i)

```
if i < 0 or i ≥ size:
  ERROR: index out of range
for j from i to size − 2:
  arr[j] ← arr[j + 1]
size ← size − 1
```

```
Size()
```

`return` *size*

Common Implementations

- **C++**: `vector`
- **Java**: `ArrayList`
- **Python**: `list` (the only kind of array)

Runtimes

$\texttt{Get}(i)$	$O(1)$
$\texttt{Set}(i, val)$	$O(1)$
$\texttt{PushBack}(val)$	$O(n)$
$\texttt{Remove}(i)$	$O(n)$
$\texttt{Size}()$	$O(1)$

Summary

- Unlike static arrays, dynamic arrays can be resized.
- Appending a new element to a dynamic array is often constant time, but can take $O(n)$.
- Some space is wasted—at most half.

Outline

1. Dynamic Arrays
2. Amortized Analysis—Aggregate Method

Sometimes, looking at the individual worst-case may be too severe. We may want to know the total worst-case cost for a sequence of operations.

Dynamic Array

We only resize every so often.

Many $O(1)$ operations are followed by an $O(n)$ operations.

What is the total cost of inserting many elements?

Definition

Amortized cost: Given a sequence of n operations, the amortized cost is:

$$\frac{\text{Cost}(n \text{ operations})}{n}$$

Aggregate Method

Dynamic array: n calls to PushBack

Aggregate Method

Dynamic array: n calls to PushBack

Let c_i = cost of i'th insertion.

Aggregate Method

Dynamic array: n calls to PushBack

Let c_i = cost of i'th insertion.

$$c_i = 1 + \left\{ \begin{array}{l} \\ \\ \end{array} \right.$$

Aggregate Method

Dynamic array: n calls to PushBack

Let c_i = cost of i'th insertion.

$$c_i = 1 + \begin{cases} i-1 & \text{if } i-1 \text{ is a power of } 2 \\ \end{cases}$$

Aggregate Method

Dynamic array: n calls to PushBack

Let c_i = cost of i'th insertion.

$$c_i = 1 + \begin{cases} i-1 & \text{if } i-1 \text{ is a power of 2} \\ 0 & \text{otherwise} \end{cases}$$

Aggregate Method

Dynamic array: n calls to PushBack

Let c_i = cost of i'th insertion.

$$c_i = 1 + \begin{cases} i-1 & \text{if } i-1 \text{ is a power of } 2 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\sum_{i=1}^{n} c_i}{n}$$

Aggregate Method

Dynamic array: n calls to PushBack

Let c_i = cost of i'th insertion.

$$c_i = 1 + \begin{cases} i-1 & \text{if } i-1 \text{ is a power of } 2 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\sum_{i=1}^{n} c_i}{n} = \frac{n + \sum_{j=1}^{\lfloor \log_2(n-1) \rfloor} 2^j}{n}$$

Aggregate Method

Dynamic array: n calls to PushBack

Let c_i = cost of i'th insertion.

$$c_i = 1 + \begin{cases} i-1 & \text{if } i-1 \text{ is a power of } 2 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\sum_{i=1}^{n} c_i}{n} = \frac{n + \sum_{j=1}^{\lfloor \log_2(n-1) \rfloor} 2^j}{n} = \frac{O(n)}{n}$$

Aggregate Method

Dynamic array: n calls to PushBack

Let c_i = cost of i'th insertion.

$$c_i = 1 + \begin{cases} i-1 & \text{if } i-1 \text{ is a power of } 2 \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\sum_{i=1}^{n} c_i}{n} = \frac{n + \sum_{j=1}^{\lfloor \log_2(n-1) \rfloor} 2^j}{n} = \frac{O(n)}{n} = O(1)$$