CALIFORNIA STATE UNIVERSITY
FULLERTON

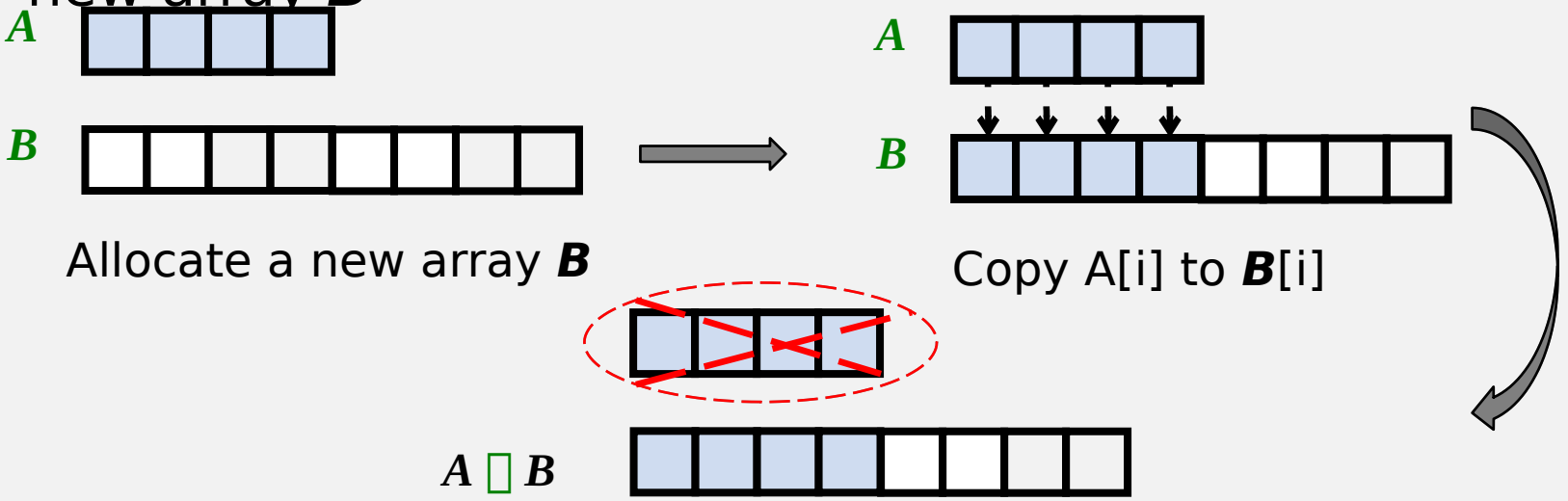
CPSC 131

Vectors & Amortized Analysis

Extendable Array-Based Vectors

Extendable Array

- In an ***insert*** operation, when the array ***A*** is full, instead of throwing an exception, the array can be replaced with a larger one
  - Allocate a new array ***B*** of a bigger capacity
  - Copy ***A***[i] to ***B***[i] for i = 0, …, N – 1
  - Delete ***A***'s current array and reassign ***A*** to point to the new array ***B***



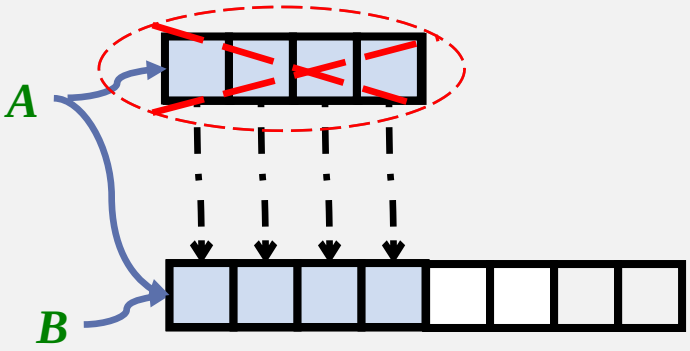
Extendable Array-Based Vectors Implementation

https://github.com/apanangadan/CSUF-CPSC_131/blob/master/ExtendableVector.h

Extendable array-based stack

- Similar to array-based stack
- But no need to worry about capacity!
- Can also make extendable array-based queues

Extendable Stack



Push to back of A until it becomes full

1. Allocate a new array ***B***
2. Copy A[i] to ***B***[i]
3. Delete (memory pointed to by) ***A***
4. ***A*** = ***B;***
5. Function ends, local variable B disappears

Cost analysis of stack implementations

	Linked list-based	Array-based	Extendable array-based
Create (constructor)	O(1)	O(1)	
Push(e)	O(1)	O(1)	
Pop()	O(1)	O(1)	
Top()	O(1)	O(1)	
Size()	O(1)	O(1)	
Empty()	O(1)	O(1)	

Cost analysis of stack implementations

	Linked list-based	Array-based	Extendable array-based
Create (constructor)	O(1)	O(1)	O(1)
Push(e)	O(1)	O(1)	
Pop()	O(1)	O(1)	
Top()	O(1)	O(1)	
Size()	O(1)	O(1)	
Empty()	O(1)	O(1)	

Cost analysis of stack implementations

	Linked list-based	Array-based	Extendable array-based
Create (constructor)	O(1)	O(1)	O(1)
Push(e)	O(1)	O(1)	**O(n) worst case**
Pop()	O(1)	O(1)	
Top()	O(1)	O(1)	
Size()	O(1)	O(1)	
Empty()	O(1)	O(1)	

Cost analysis of stack implementations

	Linked list-based	Array-based	Extendable array-based
Create (constructor)	O(1)	O(1)	O(1)
Push(e)	O(1)	O(1)	**O(n) worst case**
Pop()	O(1)	O(1)	O(1)
Top()	O(1)	O(1)	O(1)
Size()	O(1)	O(1)	O(1)
Empty()	O(1)	O(1)	O(1)

Efficiency Analysis

- When the array is full, we replace the array with a larger one
  - Performing just one array replacement required by one element insertion takes ***O***(***n***) time
  - But after performing the array replacement, the new vector allows ***n*** new elements to be added to the array before the vector must be resized again
  - ➢ **Not as slow as one thinks!**
- **Amortized analysis** – average time for a series of operations
  - Instead of looking at the individual cost of each operation independently
  - Looks at the cost of the entire series and “charges” each individual operation with a share of the total cost

Cost analysis of stack implementations

	Linked list-based	Array-based	Extendable array-based
Create (constructor)	O(1)	O(1)	O(1)
Push(e)	O(1)	O(1)	**O(n) worst case O(1) amortized (on average)**
Pop()	O(1)	O(1)	O(1)
Top()	O(1)	O(1)	O(1)
Size()	O(1)	O(1)	O(1)
Empty()	O(1)	O(1)	O(1)

std::vector

- Part of the C++ Standard Library
  - Extends basic C++ with useful classes
- Different compilers (Visual Studio, Xcode, clang++) have different implementations but the same methods
- std::vector - one of the most commonly used containers
- Based on extendable arrays
- http://www.cplusplus.com/reference/vector/vector/