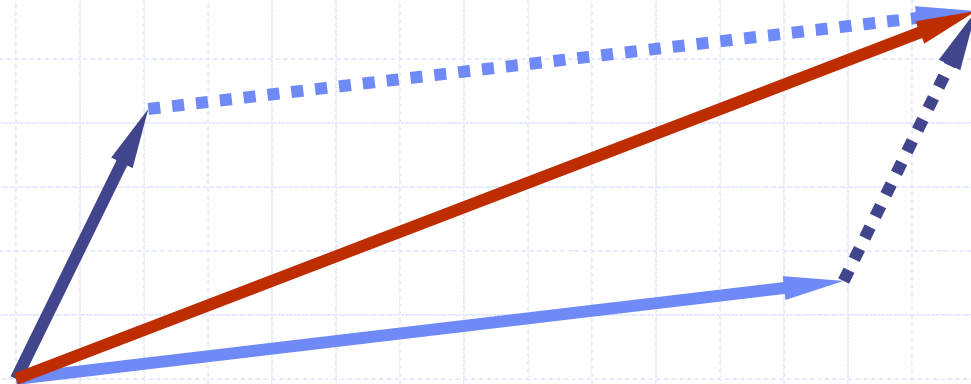


Array List



Outline and Reading

- ◆ The Array List ADT (§6.1.1)
- ◆ Array-based implementation (§6.1.3)

The Array List ADT

- ◆ The **Array List** ADT extends the notion of array by storing a sequence of arbitrary objects
- ◆ An element can be accessed, inserted or removed by specifying its index (number of elements preceding it)
- ◆ An exception is thrown if an incorrect index is specified (e.g., a negative index)
- ◆ Main array list operations:
 - Element **get**(int i): returns the element at index i without removing it
 - Element **set**(int i, Element e): replace the element at index i with e and return the old element
 - **add**(int i, Element e): insert a new element e to have index i
 - Element **remove**(int i): removes and returns the element at index i
- ◆ Additional operations: int **size**() and boolean **isEmpty**()

Applications of Array Lists

◆ Direct applications

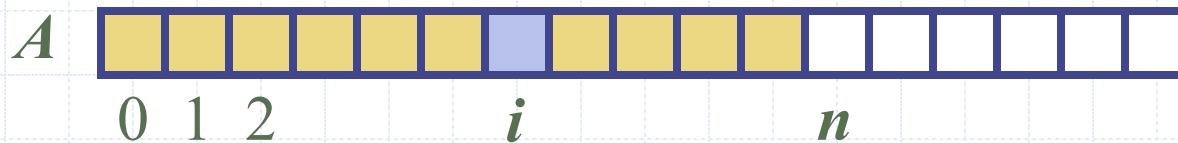
- Sorted collection of objects (elementary database)

◆ Indirect applications

- Auxiliary data structure for algorithms
- Component of other data structures

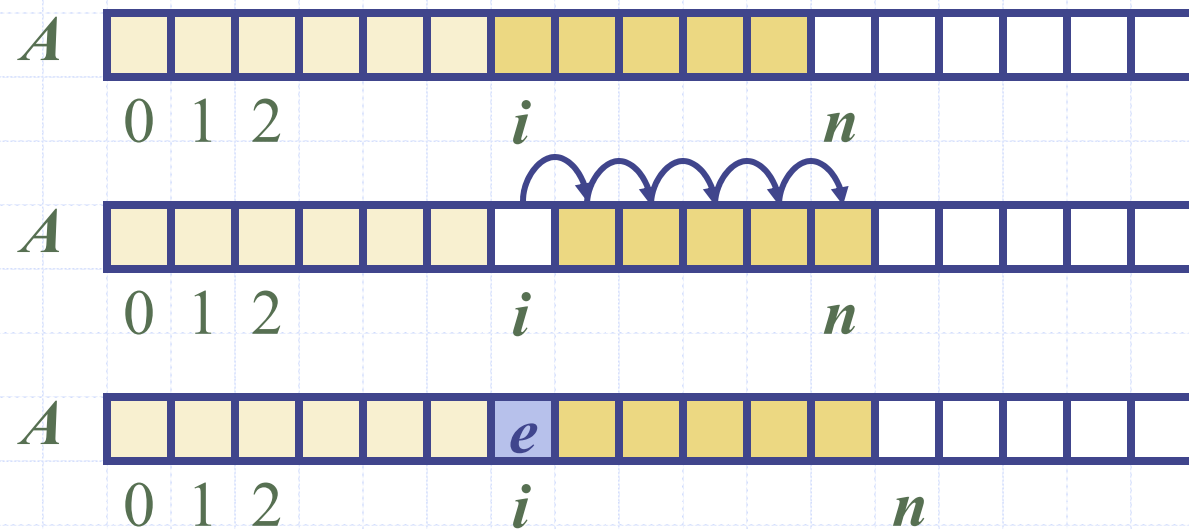
Array-based Array List

- ◆ Use an array A of size N
- ◆ A variable n keeps track of the location in the array after the last element in the Array List (the size of the Array List)
- ◆ Operation *get*(i) is implemented in $O(1)$ time by returning $A[i]$



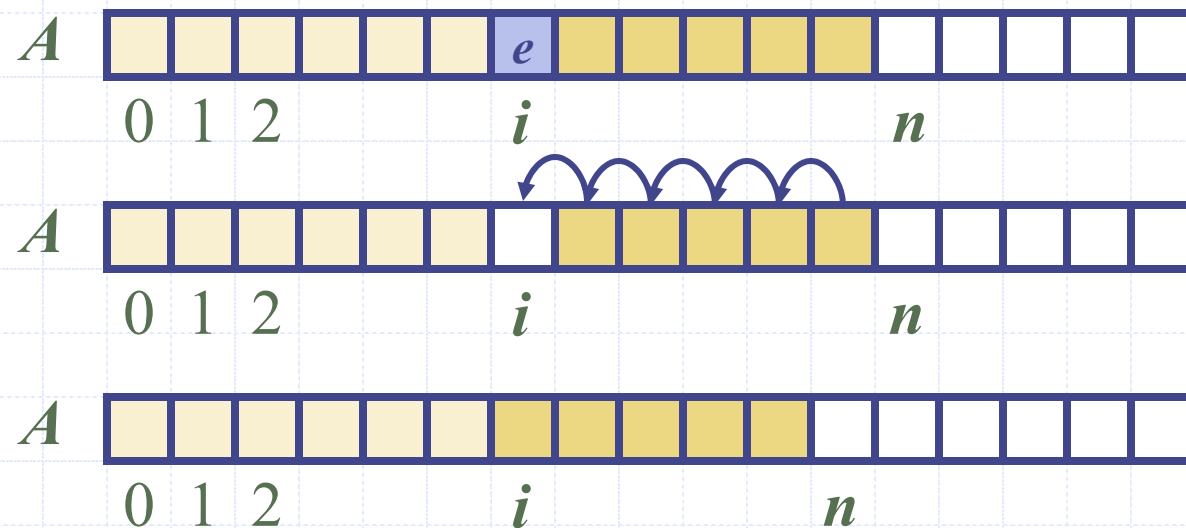
Insertion

- ◆ In operation *add*(i, e), we need to make room for the new element by shifting forward the $n - r$ elements $A[0], \dots, A[n - 1]$
- ◆ In the worst case ($i = 0$), and we must shift all elements, this takes $O(n)$ time



Deletion

- ◆ In operation *remove*(i), we need to fill the hole left by the removed element by shifting backward the $n - i - 1$ elements $A[r + 1], \dots, A[n - 1]$
- ◆ In the worst case ($i = 0$), this takes $O(n)$ time



Performance

- ◆ In the array based implementation of a Array List
 - The space used by the data structure is $O(n)$
 - *size*, *isEmpty*, *get* and *set* run in $O(1)$ time
 - *add* and *remove* run in $O(n)$ time
- ◆ If we use the array in a circular fashion, two common cases, *add(0)* and *remove(0)* run in $O(1)$ time
- ◆ In an *add* operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one