Introduction to Data Structures

Two ways of storing data in memory

Part I. Arrays

Lecture 5 by *Marina Barsky*

Outline

- Discuss two alternative ways of storing a ***sequence*** of values:
    - Array
    - Linked List

- Functionality:
    - Get element by position (index)
    - Search for a position of a target element
    - Add new element at a given position
    - Remove an element at a given position

Arrays Revisited

```
long arr[] = new long[5];
```

```
Dog arr[] = new Dog[3];
```

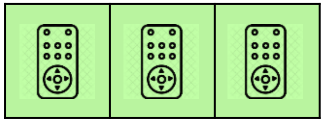
```
int arr[] = new int[2][5];
```

1	5	17	3	25

1D

1	5	17	3	25
8	2	36	5	3

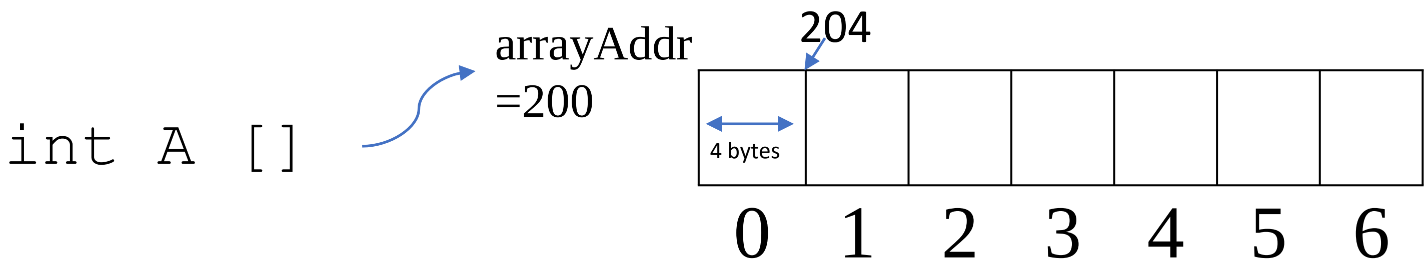
2D



References

Definition

Array is a **contiguous** area of memory containing **equal-size** elements indexed by contiguous integers



What we do with arrays

- Read operations:
  - get (index i)
  - find (Object o)

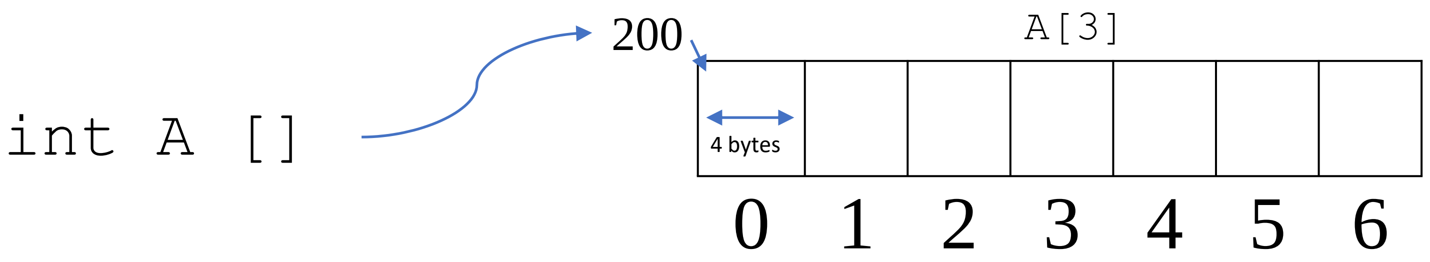
- Edit operations:
  - add()
  - remove()

What we do with arrays

- Read operations:
  - get (index i)
  - find (Object o)

- Edit operations:
  - add()
  - remove()

Get an element by index



- Because of contiguous arrangement we can directly access any element of the array by index *i*.

- The address of `A[i]` is computed as:

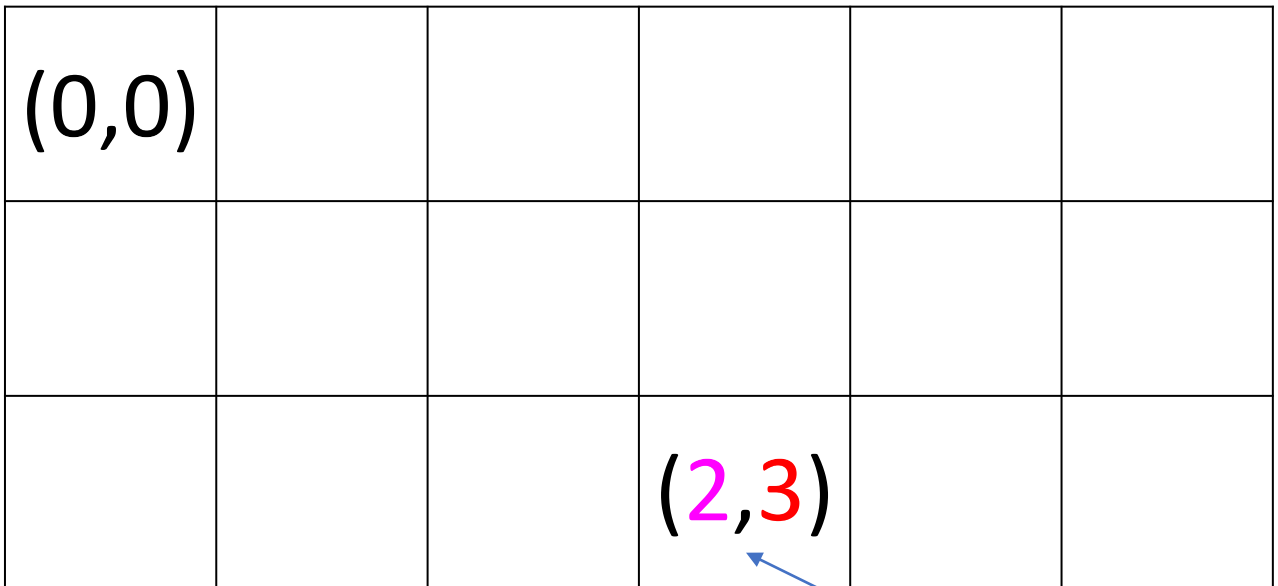
  $$\text{arrayAddr} + \text{elemSize} \times (i\ )$$

  and we can jump directly to this address

- For example, address of A[3] = 200+3*4 = 212

Same for Multi-Dimensional Arrays

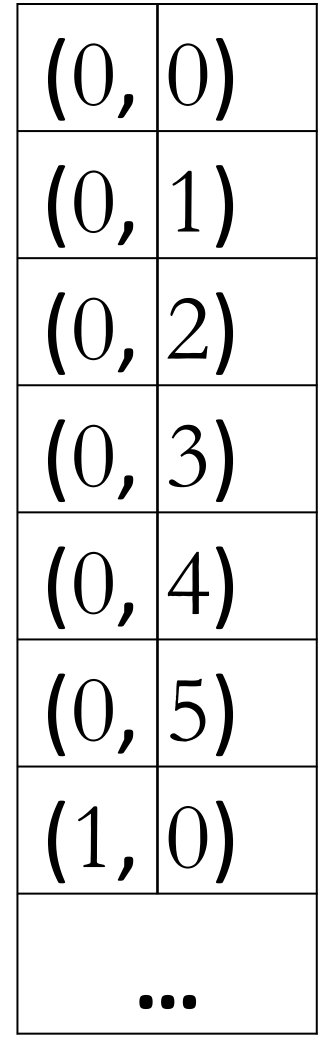
```
int arr [3][6];
```



$arrayAddr + elemSize \times (2 \times 6 + 3)$

The position of element `A[i][j]` in 2D array `A[rows][cols]` is computed as:

arrayAddr + *elemSize* × (`i` × `rows` + `j`)

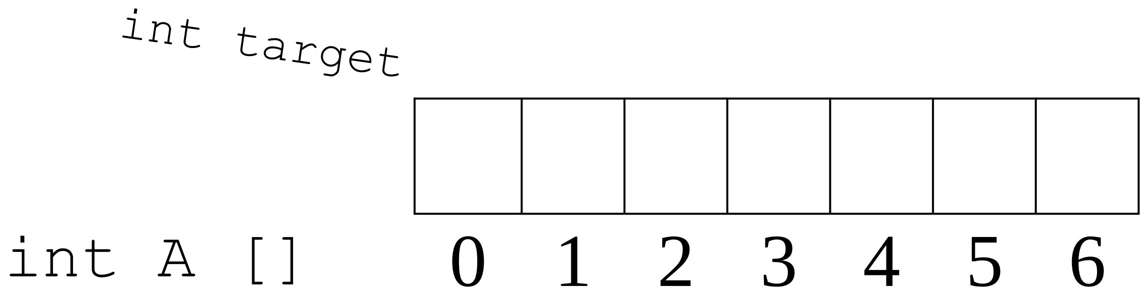


What we do with arrays

- Read operations:
  - get (index i)
  - **find** (Object o)

- Dynamic edit operations:
  - add()
  - remove()

Find an element: Linear Search



1. we iterate changing *i* from 0 to *length* - 1
2. if `A[i] == target` : found, return i
3. finished the loop : not found, return -1

```
static int find (int [] A, int target) {
    for (int i=0; i< A.length; i++) {
        if (A[i] == target)
            return i;
    }
    return -1;
}
```

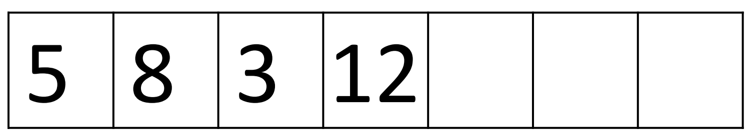
What we do with arrays

- Read operations:
  - get (index i)
  - find (Object o)

➢ Edit operations:
  - add()
  - remove()

Edit operations: add/remove

- We can use space allocated for the array to store a variable number of elements
- We just need to distinguish between the array capacity (length) and the actual number of elements in the array (we will call it size)
- This is especially useful if we have array of references – we can keep track of the number of actual objects in the array

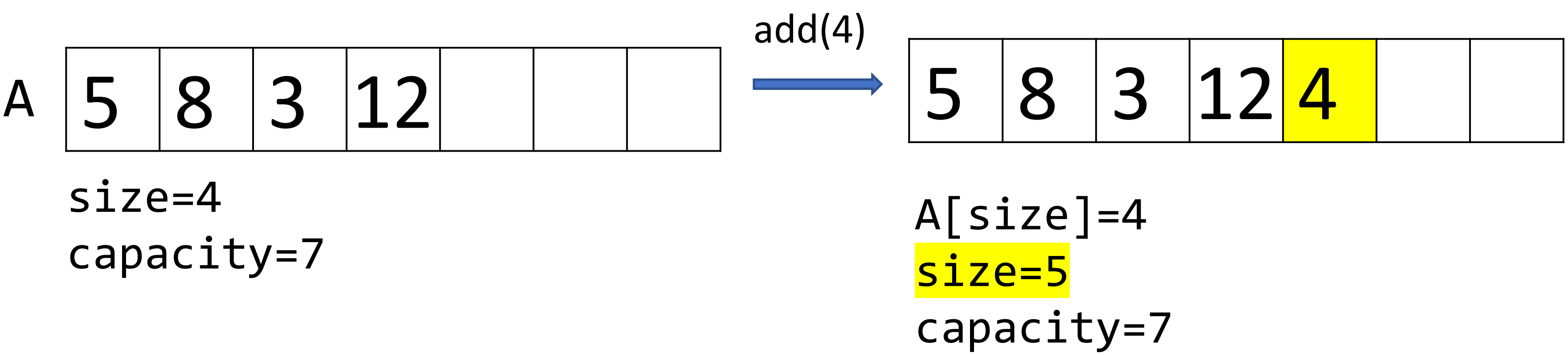



```
size=4
capacity=7
```

We can store the actual number of the elements added to the array in a variable `size`

Add to the end of A

1. As long as capacity permits, add new element to the empty slot at position *`size`*
2. Increment *`size`* by 1



Add in the middle of A

- We must keep elements consecutive: only contiguous sequence in memory lets us fast retrieval by position
- If we want to insert an element at some position `j` of A, we must shift all the elements from *j* to `size-1` to the right

5	8	3	12	4		

```
A[size]=4
size=5
capacity=7
```

We need to check that *j* is a valid position: *j<=size*

add(9 at j=2)

5	8		3	12	4	

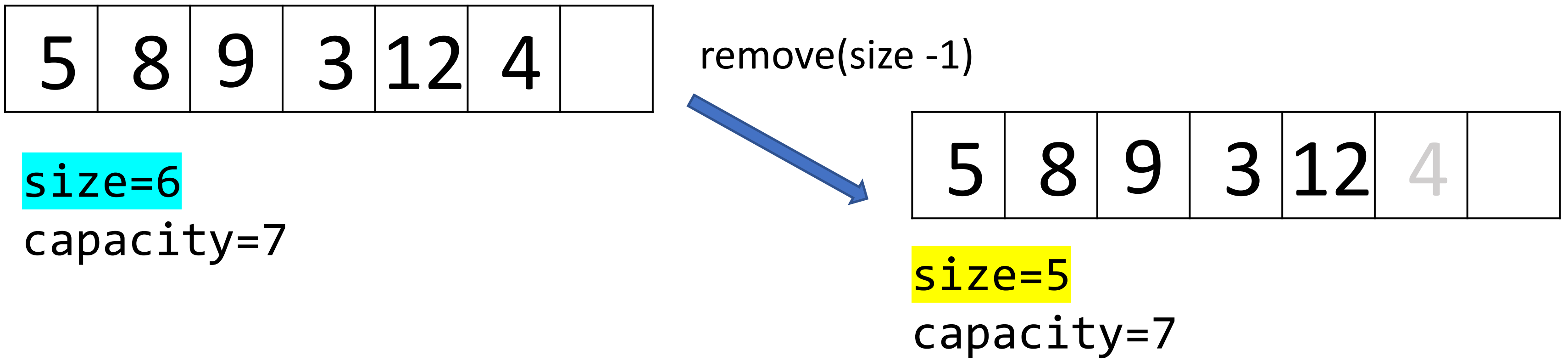
5	8	9	3	12	4	

```
for (int i=size; i>j; i--)
    A[i] = A[i-1]
```

```
A[j]=9
size=6
capacity=7
```

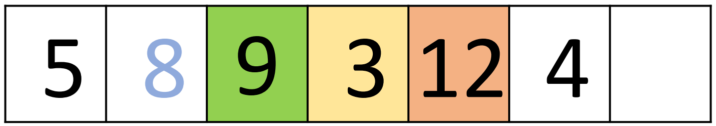
Remove from the end

- Simply decrement `size`



Remove in the middle

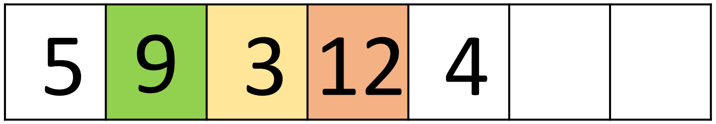
- To remove element at position j, shift all elements from *j*+1 to `size` to the left and decrement `size`




```
size=6
capacity=7
```

Also need to check that *j* is a valid position: *j*<*size*

remove(j=1)




```
for (int i=j+1; i<size; i++)
    A[i-1] = A[i]
```

```
size=5
capacity=7
```

If we try to add an element past the capacity of the array:

Bad things happen:

- Java: Array index out of bound
- Python: List index out of range
- C: No warnings, total corruption of program memory

But we cannot always know in advance how many elements we are going to store in the Array!

A new data structure

Dynamic Array (also known as *Resizable Array*)

Idea: store in a variable a reference to an array and when needed replace it with a new reference to a new array, double size

Definition

Dynamic Array:
data structure that supports the same operations as a regular array, but does not limit the number of elements that it can hold

Dynamic allocation of space

- We keep track of the number of elements in the array using variable *size*
- If *size* reaches *capacity*, then we need more space
- We allocate a new larger array and transfer data from an old array to the new one

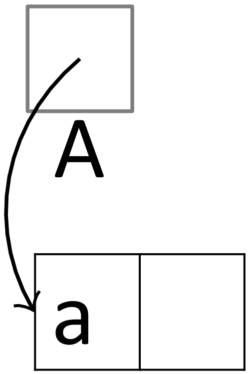
```
int myArray[100];
//Adding data to myArray…
int[] newArray = new int[200];
System.arraycopy(myArray, 0, newArray, 0, 100);
```

Dynamic array

We need keep track of 3 variables:

- `A`: reference to the beginning of the array
- `capacity`: current length of the dynamically-allocated array
- `size`: number of elements currently in the array

Dynamic Array: Resizing

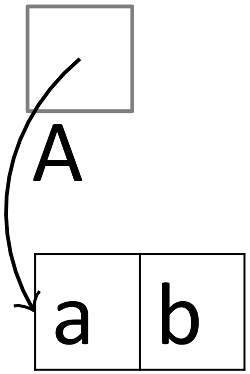


size: 1 capacity: 2

add(a)

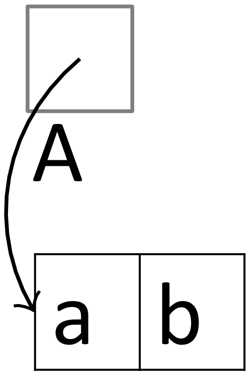
Dynamic Array: Resizing

size: 2 capacity: 2



add(b)

Dynamic Array: Resizing



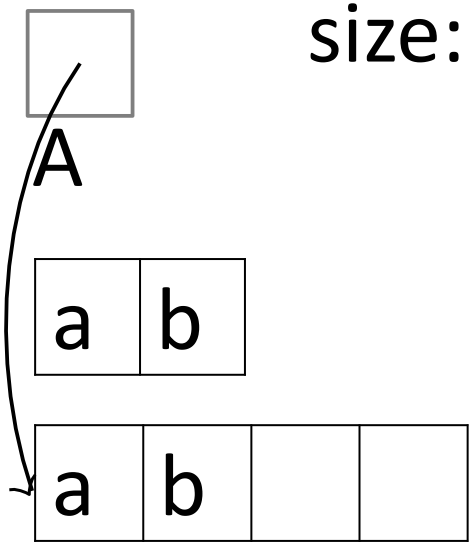
size: 2 capacity: 2

`add(c)`

Cannot add c: need to resize

Dynamic Array: Resizing

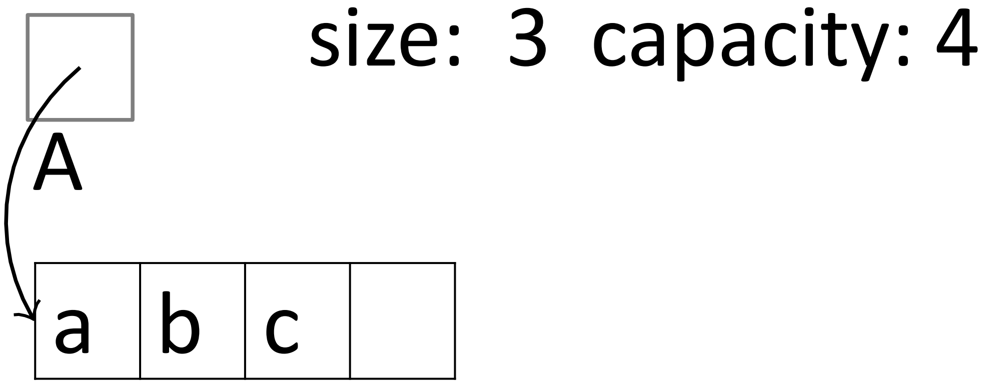
size: 2 capacity: 4



add(c)

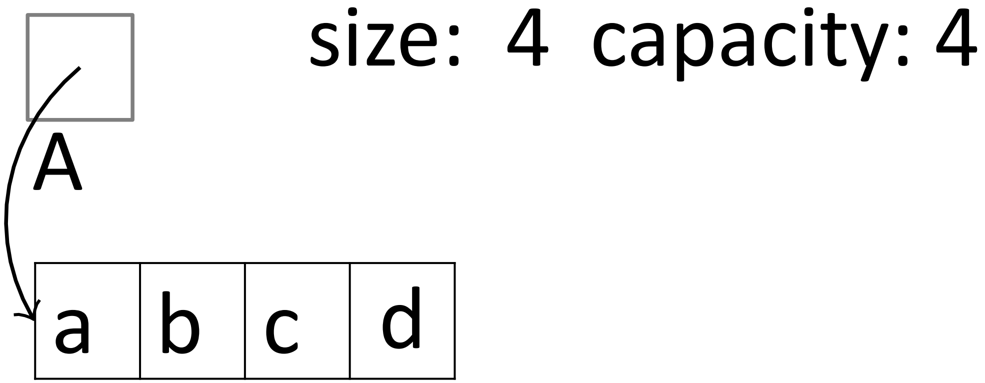
Resize array: copy old data

Dynamic Array: Resizing



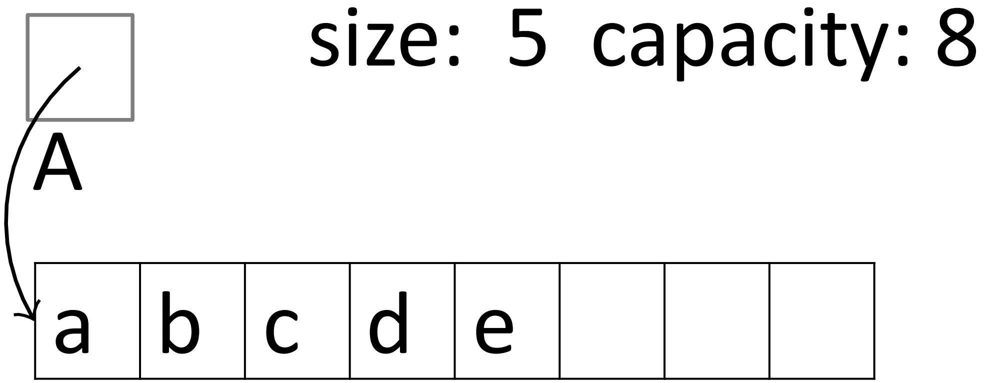
add(c)

Dynamic Array: Resizing



add(d)

Dynamic Array: Resizing



`add(e)`

Which method in Dynamic Array **always** requires only one operation?

A. Add to the end
B. Remove from the middle
C. Get element at position *i*
D. Find position of a given element
E. None of the above

Arrays: summary

- The discussion in this lecture relates to a **general concept of an Array as a way of storing a sequence of values**, not an Array in Java or in any other programming language
- The equal-sized sequence elements are placed consecutively in memory, and this allows direct access to the *i*-th element of the sequence in one operation
- To maintain this efficiency, we must make sure that there are no gaps and this makes adding/removing elements more expensive
- The array capacity can be adjusted when needed through doubling its size when it becomes full. The resizable array is called a dynamic array