Array Data Structures

Data Structures for Computer Professionals

Patiwet Wuttisarnwattana, Ph.D.

patiwet@eng.cmu.ac.th

Computer Engineering, Chiang Mai University

Data Structures covered in this class

- Linear Data Structures
  - Linked Lists
  - Arrays
  - Queues
  - Stacks
- Trees
  - Binary Trees
  - Binary Search Trees
  - AVL Trees
  - *B-Trees*
  - *Splay Trees*
- Priority Queues
  - Binary Heaps
- Hash Tables
  - Hash Functions
  - Collision Resolutions
- Graphs
  - BFS
  - DFS

Array Data Structure

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

```
long arr[5];
```

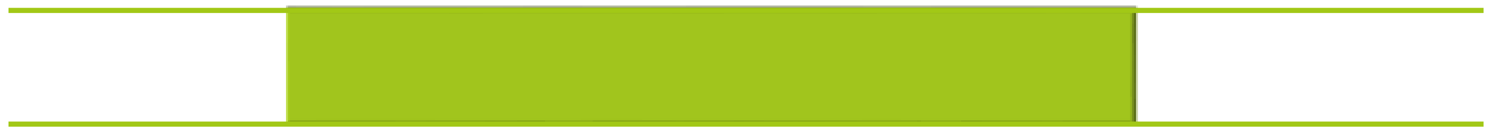
```
arr = [None] * 5
```

1	5	17	3	25

1	5	17	3	25
8	2	36	5	3

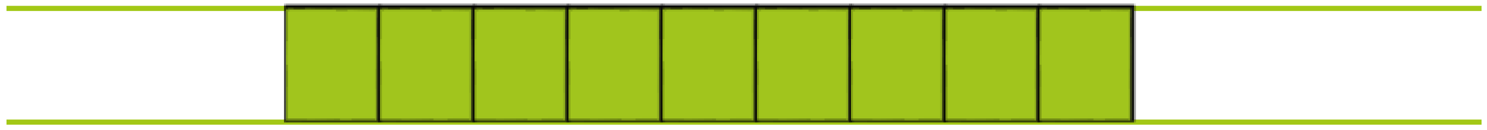
Array

- Definition
- Array:
  - Contiguous area of memory



Array

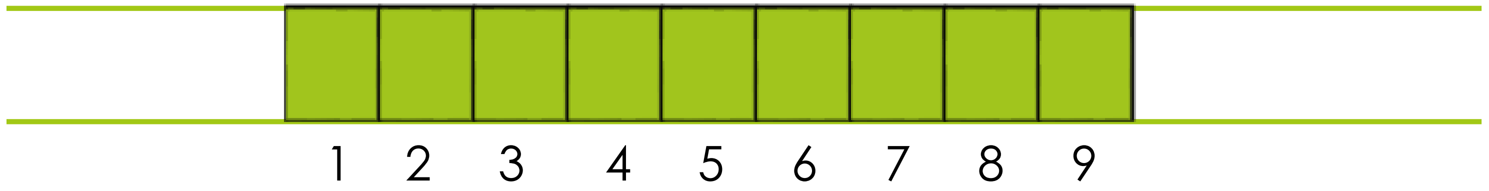
- Definition
- Array:
  - Contiguous area of memory consisting of equal-size elements



Array

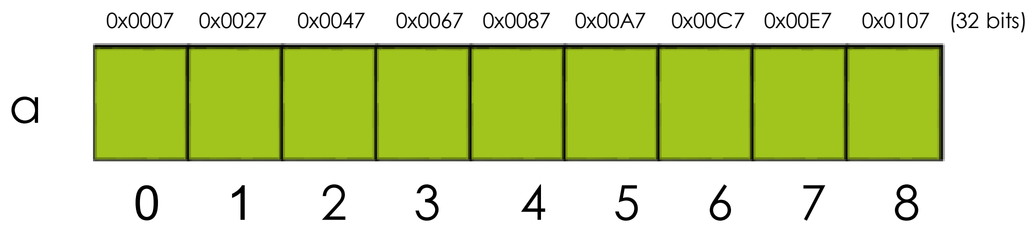
- Definition

- Array:
  - Contiguous area of memory consisting of equal-size elements indexed by contiguous integers.



What’s special about Arrays?

- Constant-time Access
- Given an index *i*, reading time and writing time are constant



a[0] = 0
0x0007 - 0x0026 ← 0

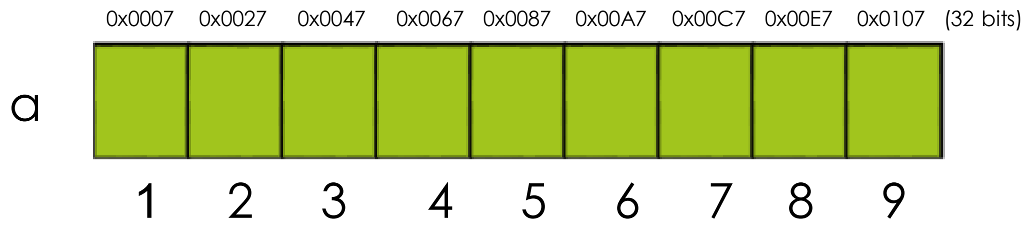
a[2] = 2.00000001
0x0047 - 0x0066 ← 2.00000001

a[4] = 55555.66666666
0x0087 - 0x00A6 ← 55555.66666666

a[7] = 1
0x00E7 - 0x0106 ← 1

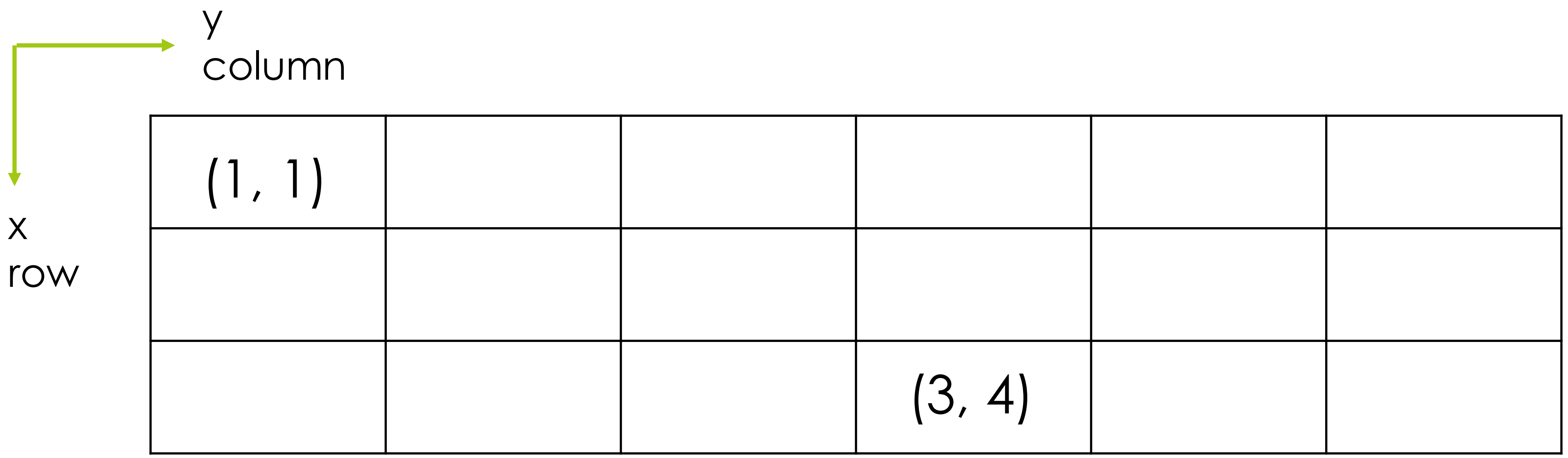
What's special about Arrays?

- Constant-time Access
- Given an index *i*, reading time and writing time are constant



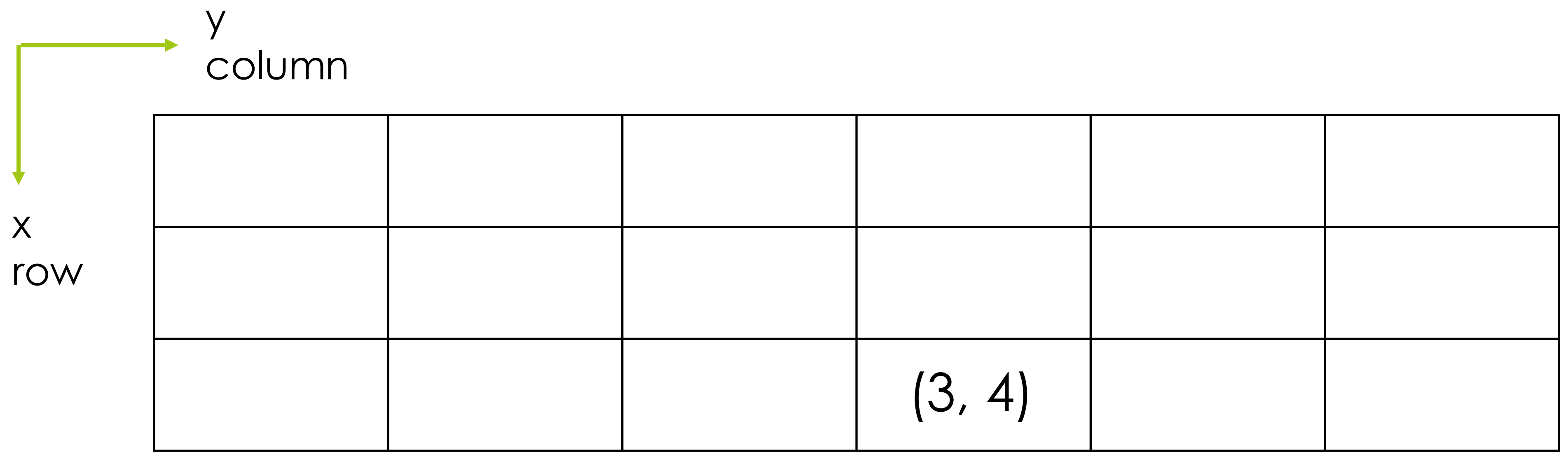
accessing_addr =
array_addr + *element_size* x (*i* - *first_index*)

Multi-Dimensional Array



my2DArray(row_index, col_index) = my1DArray(index)

Multi-Dimensional Array using 1D array



How to implement 2-dimensional array using one-dimensional array?

If you have a 2D array indexes at (3, 4), what should be the index in 1D array.

Row Major Indexing

y, column

x row

	1	2	3	4	5	6
1	0	1	2	3	4	5
2	6	7	8	9	10	11
3	12	13	14	15	16	17

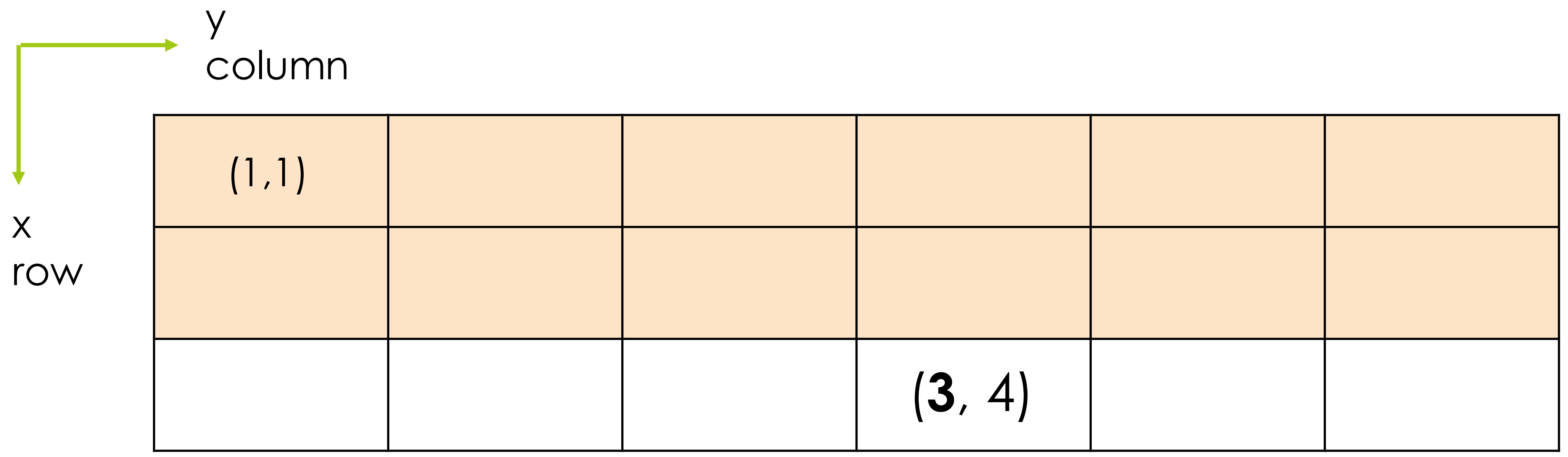
Fill row first, then column

$(1, 1) \leftrightarrow 0$

$(2, 2) \leftrightarrow 7$

$(3, 4) \leftrightarrow 15$

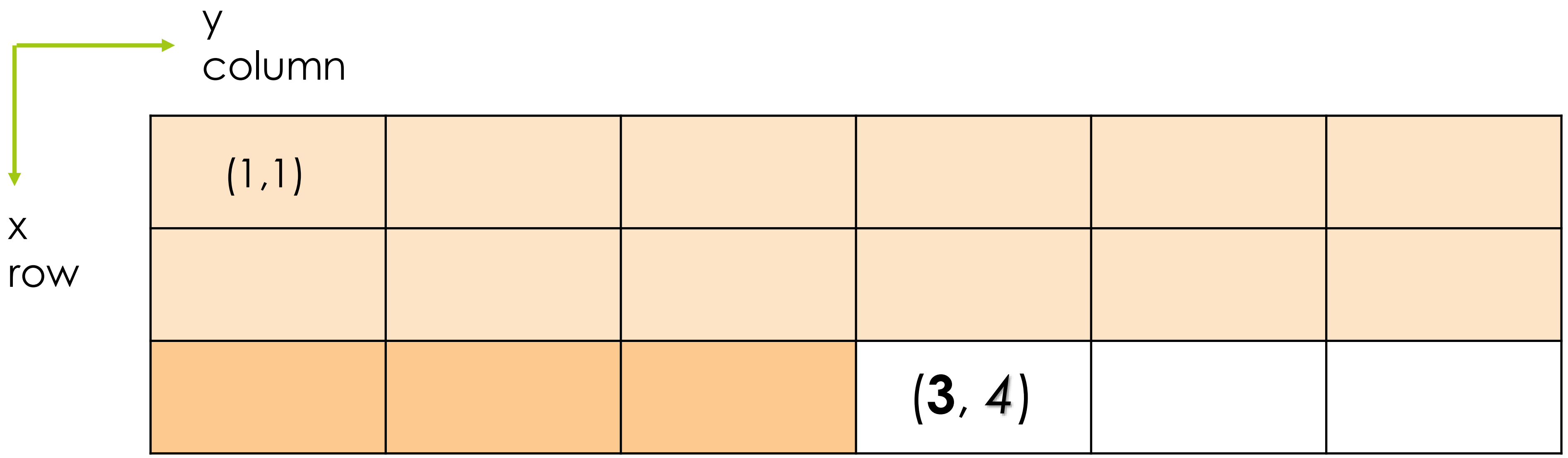
Multi-Dimensional Array using 1D array



If you have a *One-indexed 2D array* indexes at (3, 4), what should be the index in ***Zero-indexed*** *1D array*?

$(\mathbf{3} - 1) \times 6$

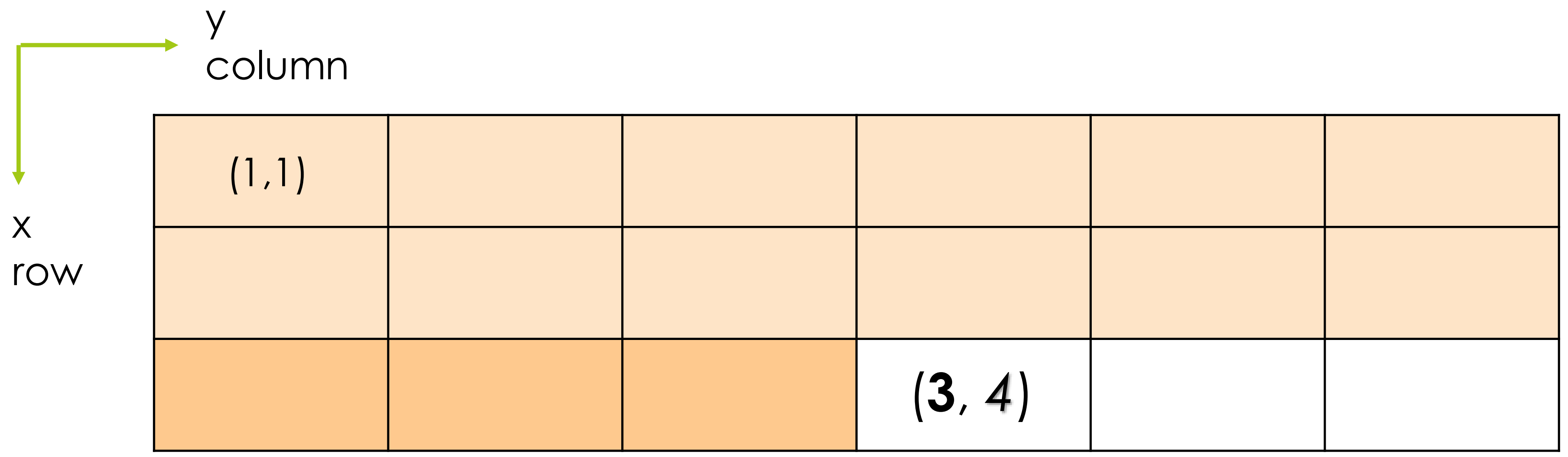
Multi-Dimensional Array using 1D array



If you have a *One-indexed 2D array* indexes at (3, 4), what should be the index in the ***Zero-indexed*** *1D array*?

1D Index = $(\mathbf{3} - 1) \times 6 + (4 - 1)$

Multi-Dimensional Array using 1D array



If you have a *One-indexed 2D array* indexes at (3, 4), what should be the address in the ***Zero-indexed*** *1D array*?

accessing_addr =
array_addr + element_size x ((**3** – 1) x 6 + (*4* - 1))

Column Major Indexing

y, column

x row

	1	2	3	4	5	6
1	0	3	6	9	12	15
2	1	4	7	10	13	16
3	2	5	8	11	14	17

Fill column first, then row

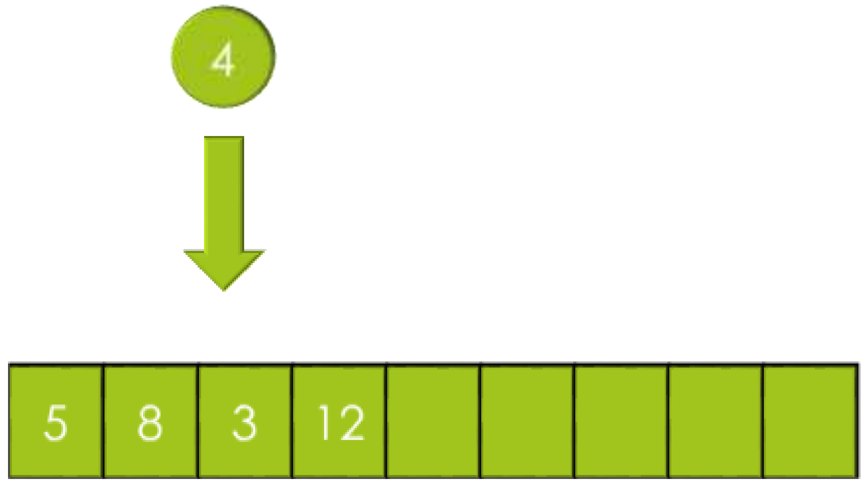
$(1, 1) \leftrightarrow 0$

$(2, 2) \leftrightarrow 4$

$(3, 4) \leftrightarrow 11$

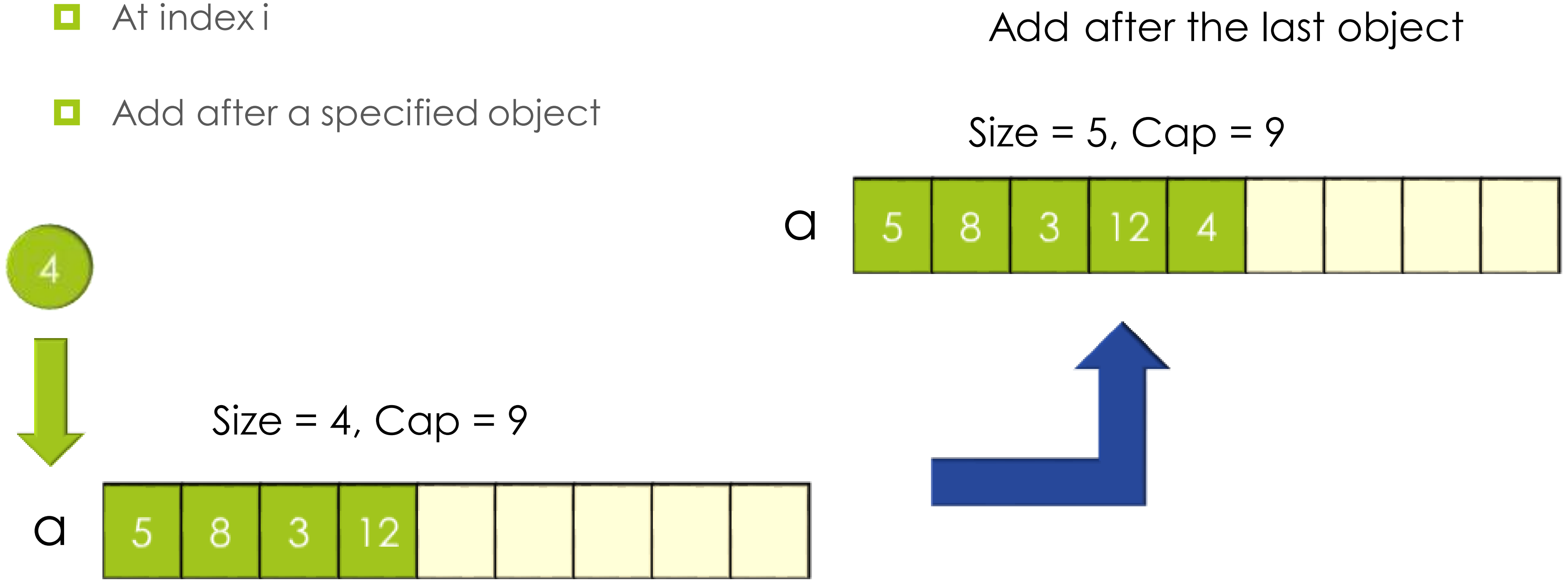
Array as a data structure

- Add object (method)
  - After the last object
  - At the beginning
  - At index i
  - Add after a specified object
- Remove object (method)
  - The last object
  - The first object
  - Object at index i
  - Remove a specified object
- Size or Length: number of objects contained (property)
- Capacity or Max: maximum number of objects allowed (property)



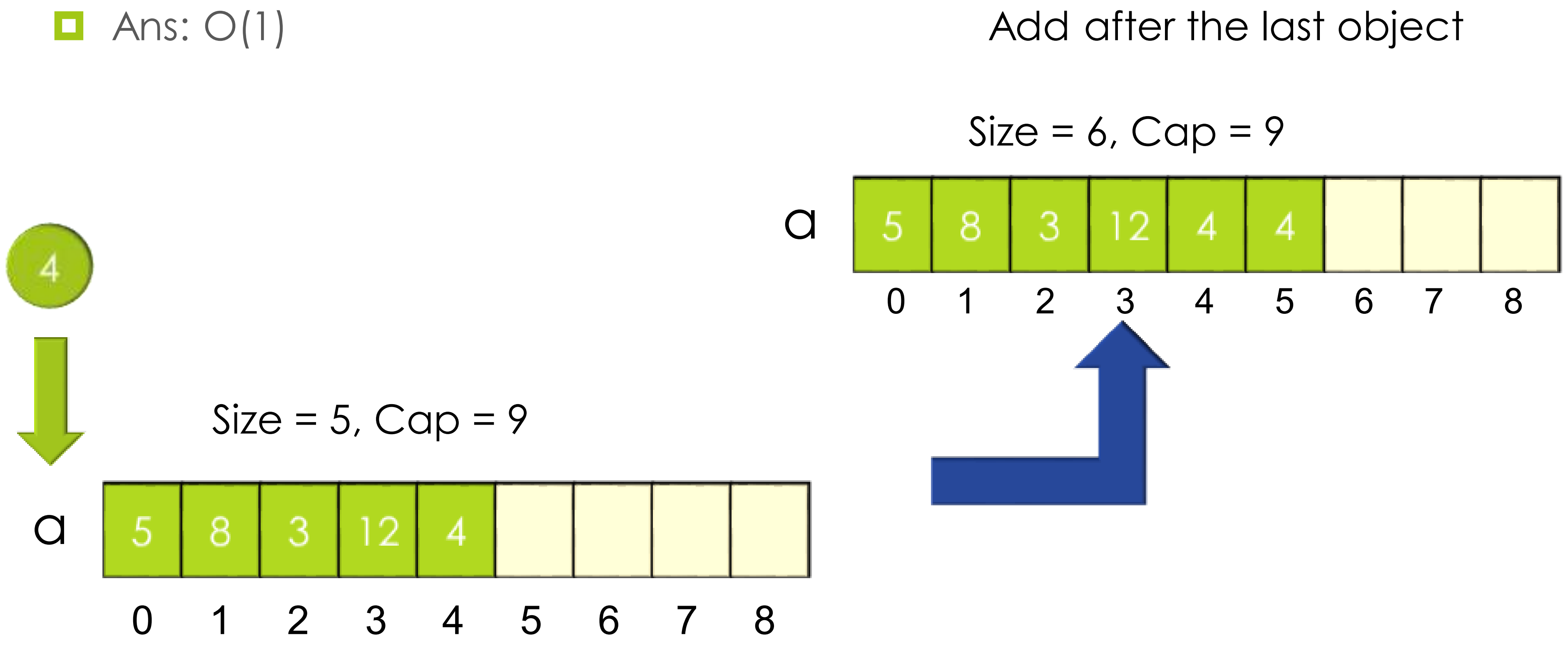
Add objects to an array

- After the last object
- At the beginning
- At index i
- Add after a specified object



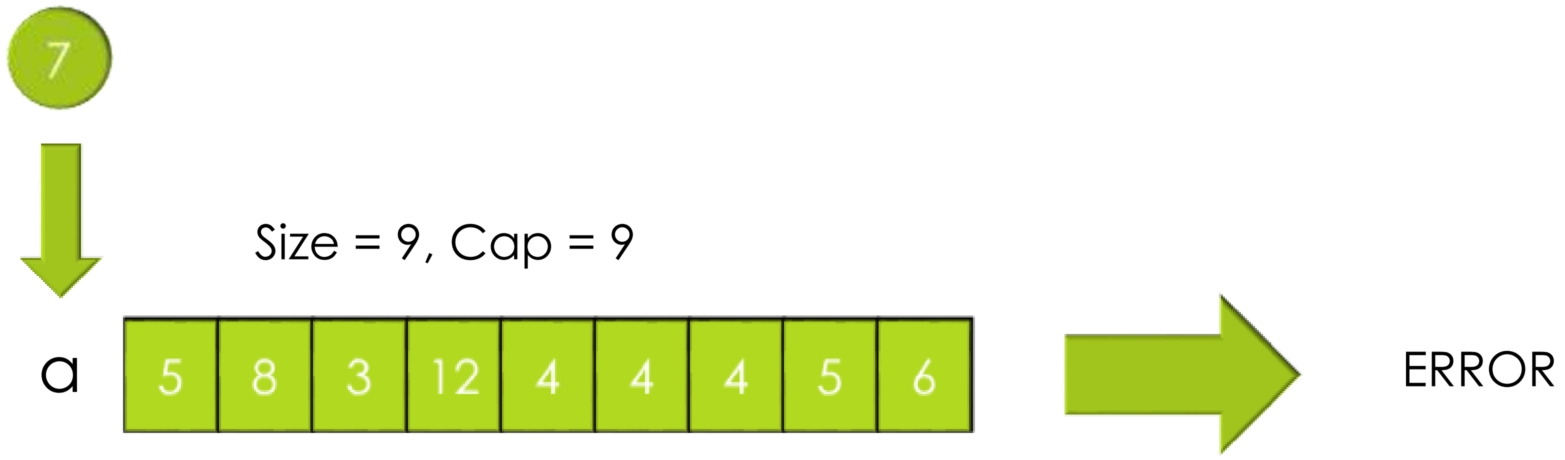
Add objects to an array

- After the last object
- What is Big O of the AddLast operation?
- Ans: O(1)



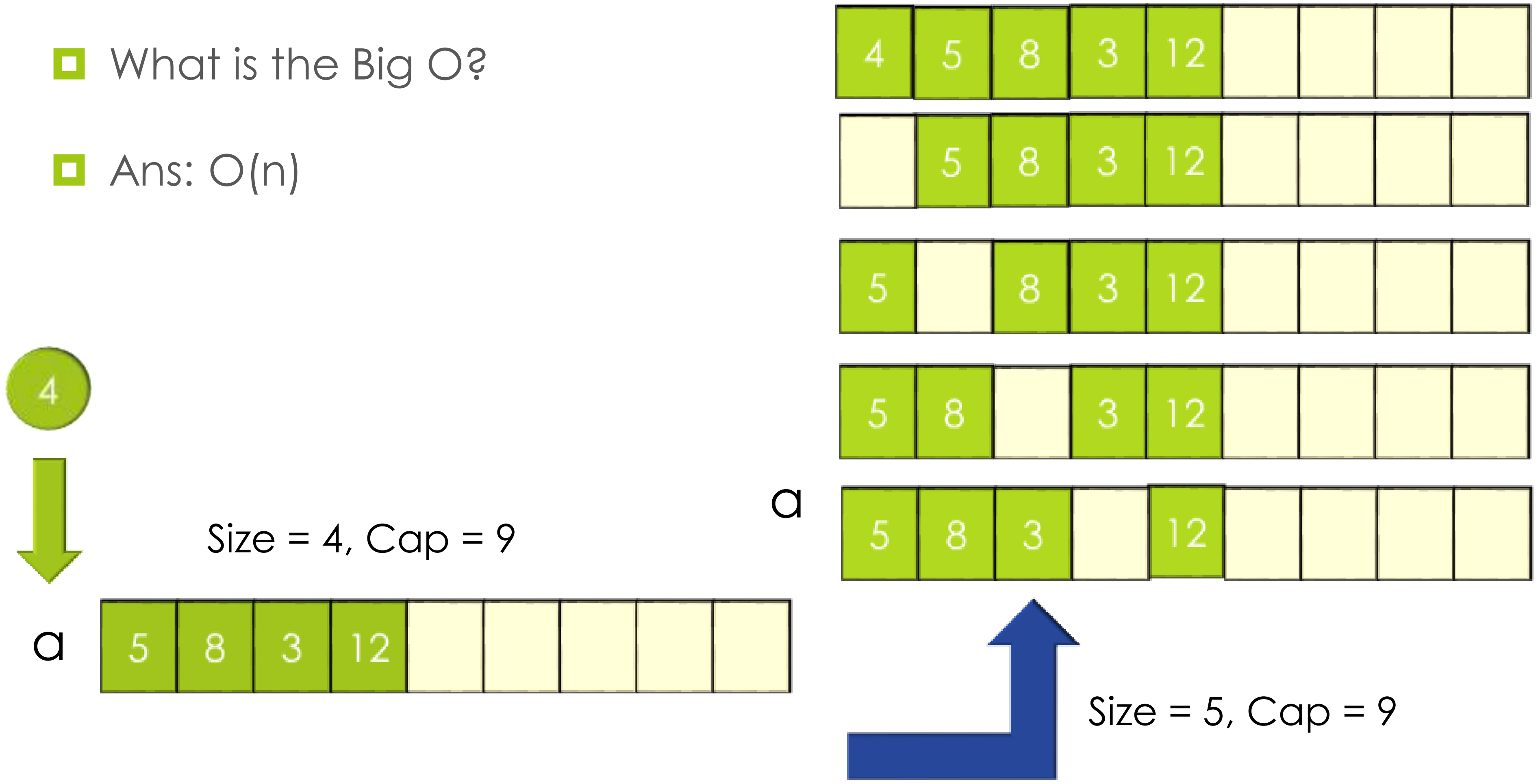
Add objects to an array

- After the last object
- At the beginning
- At index i
- Add after a specified object



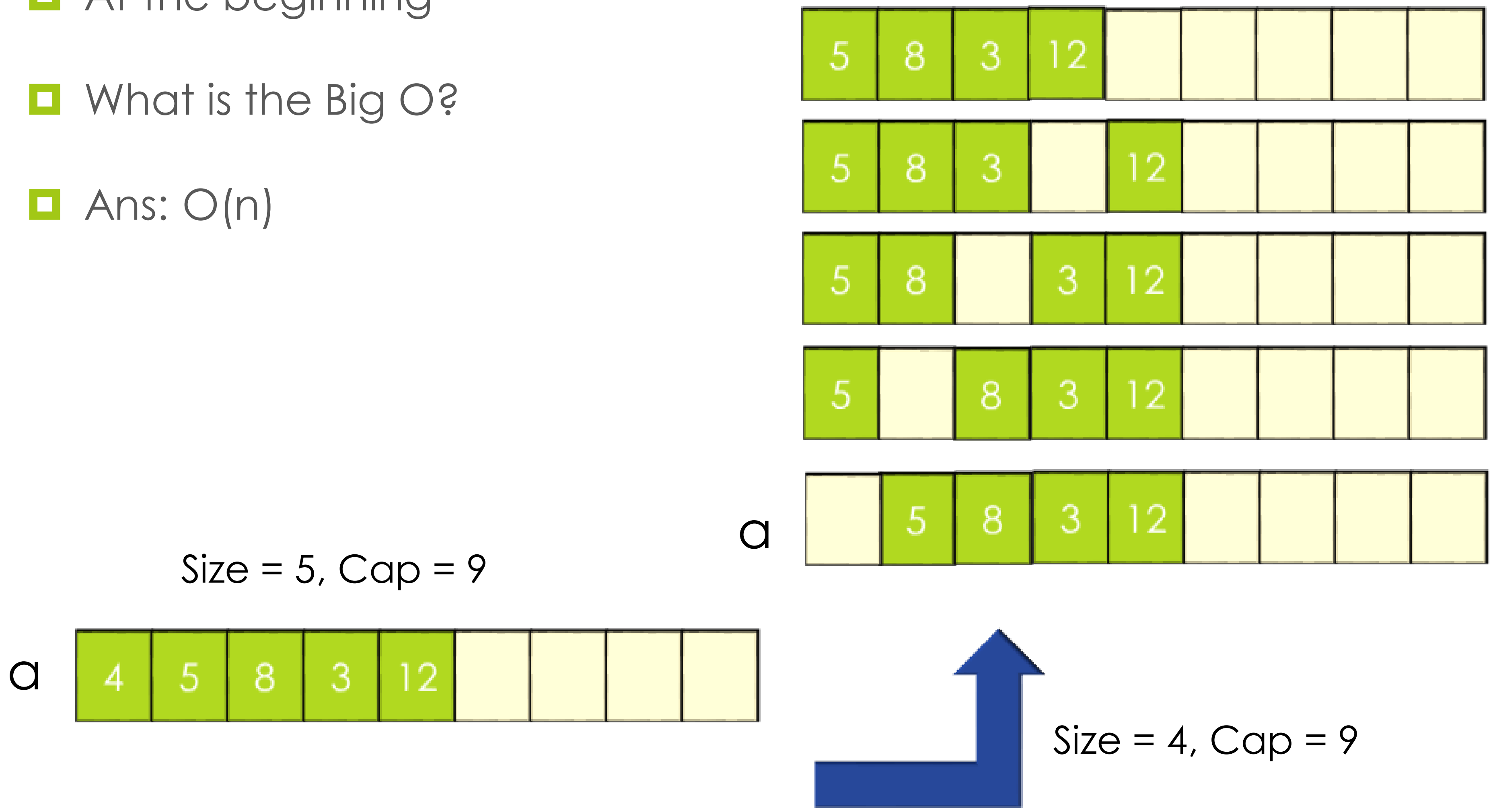
Add an object to the beginning of an array

- At the beginning
- What is the Big O?
- Ans: O(n)



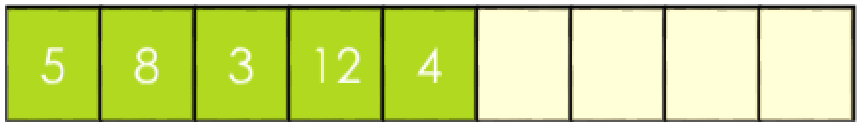
Remove an object to the beginning of an array

- At the beginning
- What is the Big O?
- Ans: O(n)



Times for Common Operations

	Add	Remove	Read/Write
Beginning	O(n)	O(n)	O(1)
End	O(1)	O(1)	O(1)
Middle	O(n)	O(n)	O(1)



Summary

- Array: Contiguous area of memory consisting of equal-size elements indexed by contiguous integers
- Constant-time access to any element
- Constant time to add/remove at the end
- Linear time to add/remove at an arbitrary location

What is the output?

```
int[] myArray = new int[10];

for (int i = 0; i < 10; i++){

      myArray[i] = i;

}

for (int i = 0; i < 10; i++) {

      Console.WriteLine(myArray[i]);

}
```

Your choice?

1. Compilation error
2. Runtime Exception
3. Nothing
4. Print out numbers?

Assume that the following code is in a proper main function

What is the output?

```
int[] myArray = new int[5];

Console.WriteLine(myArray[0]);

myArray[0] = 1;

Console.WriteLine(myArray[0]);

myArray = new int[10];

Console.WriteLine(myArray[0]);
```

Your choice?

1. Compilation error
2. Runtime Exception
3. Nothing
4. Print out numbers?

Assume that the following code is in a proper main function

What is the output?

```
int[] myArray = new int[10];

for (int i = 0; i <= 10; i++){

      myArray[i] = i;

}

for (int i = 0; i <= 10; i++){

      Console.WriteLine(myArray[i]);

}
```

Assume that the following code is in a proper main function

Your choice?

1. Compilation error
2. Runtime Exception
3. Nothing
4. Print out numbers?

Problem with static array???

- Static arrays are static!
- Static array size is fixed; if there is a new coming object (there always is), it will be an error "IndexOutOfRangeException".
- What is the solution?
  - Use other data structures!
  - If you still prefer "constant time to read/write" and "integer indexing", what should you do?
  - Dynamic arrays can be your solution

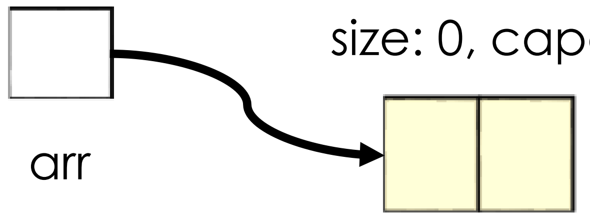
Dynamic Array

- Dynamic array is an array
  - Contiguous area of memory consisting of equal-size elements indexed by contiguous integers
  - Constant time to read/write
- What special is: Dynamic array can always accept new data
- The idea is: Dynamic array will extend it capacity when the size is full
- The implementation trick is: (1) If array is full, create a new array with a bigger size, (2) Change *the array reference* to a new bigger reallocated array.

Dynamic Array Resizing

Event 1: Allocate a dynamic array name “arr” type “char” with initial cap of 2

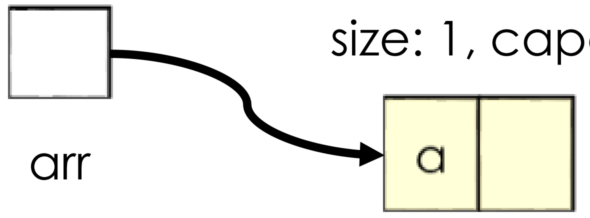
size: 0, capacity: 2



Event 2: Add a new character ‘a’ → Size < Cap → Just add it

PushBack(‘a’)

size: 1, capacity: 2

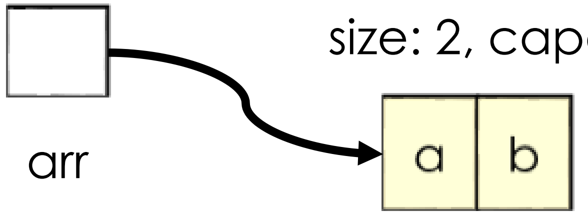


Dynamic Array Resizing

Event 3: Add a new character ‘b’ → Size < Cap → Just add it

PushBack(‘b’)

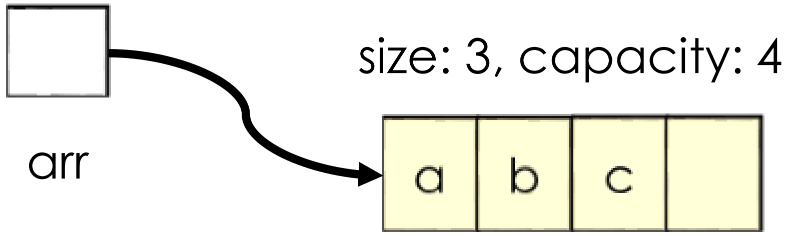
size: 2, capacity: 2



Event 3: Add a new character ‘c’ → Size == Cap → Resize the array and Add

PushBack(‘c’)

size: 3, capacity: 4

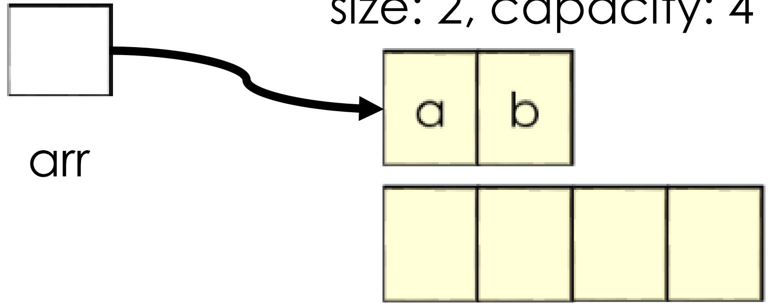


Dynamic Array Resizing

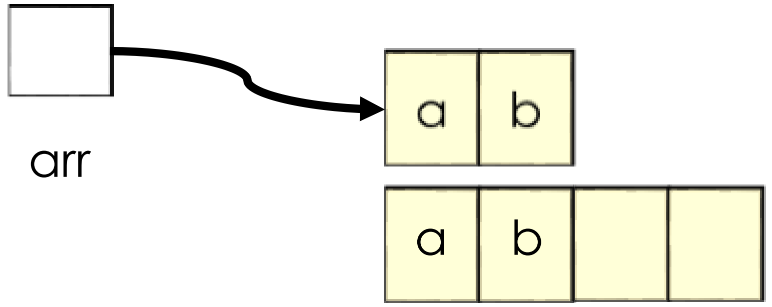
Implementation Step 1: allocate a new array with the same type but the capacity is double the old capacity

PushBack('c')

size: 2, capacity: 4



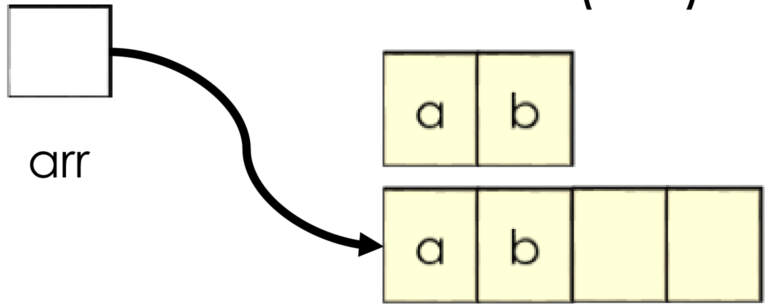
Implementation Step 2: Copy all data from the old array to the new array at the same position



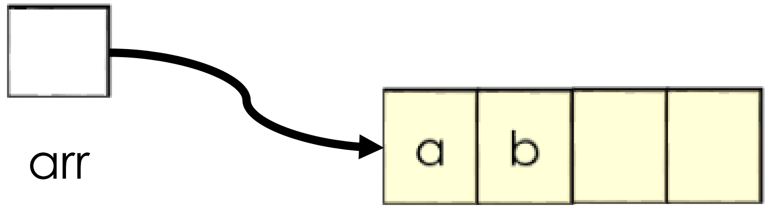
Dynamic Array Resizing

Implementation Step 3: Change the reference to the new array

PushBack('c')



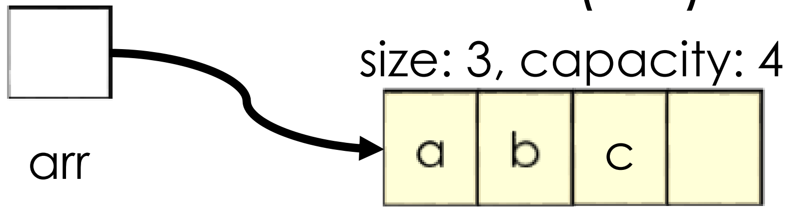
Implementation Step 4: Delete the old array (Automatically done in Java)



Dynamic Array Resizing

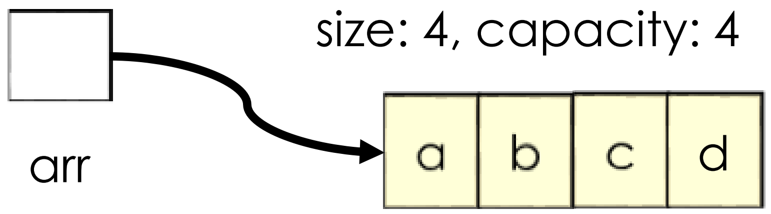
Implementation Step 5: Add the new data as usual

PushBack('c')



Event 4: Add a new character 'd' → Size < Cap → Just add it

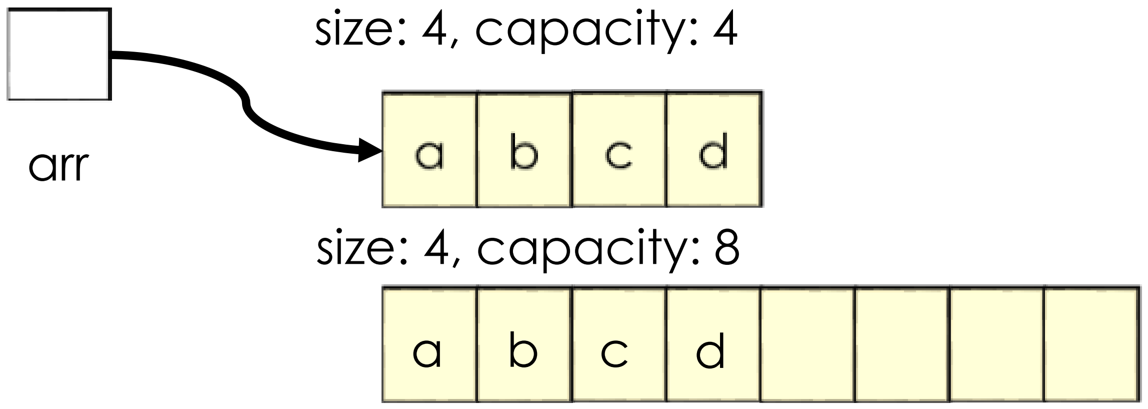
PushBack('d')



Dynamic Array Resizing

Event 5: Add a new character ‘e’ → Size == Cap → Resize the array and Add

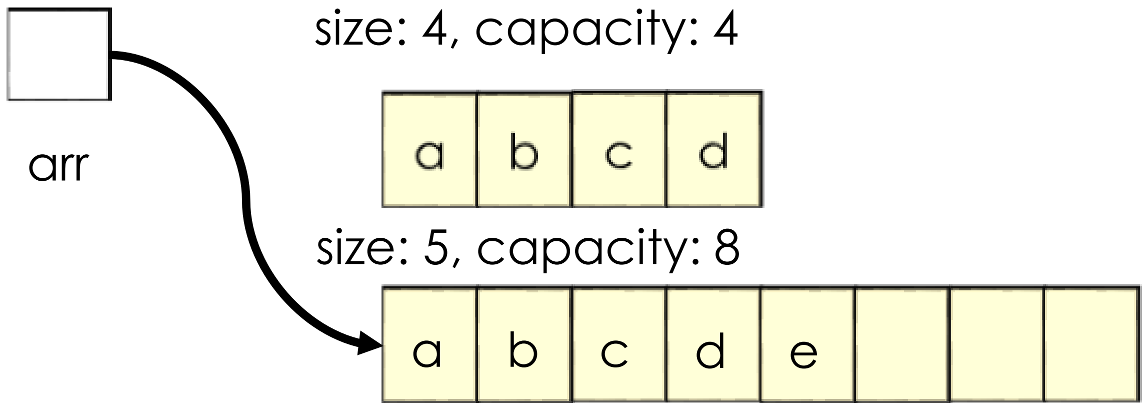
PushBack(‘e’)



Dynamic Array Resizing

Event 5: Add a new character ‘e’ → Size == Cap → Resize the array and Add

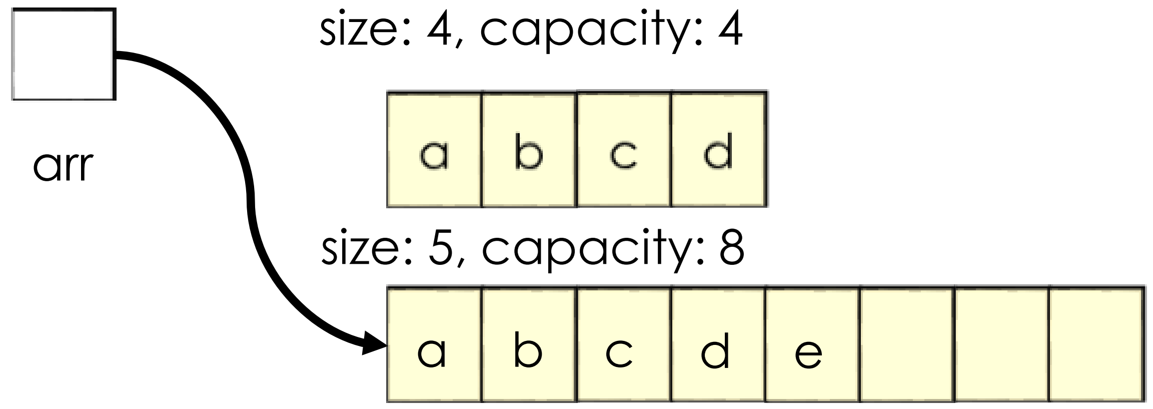
PushBack(‘e’)



Dynamic Array Resizing

Event 5: Add a new character ‘e’ → Size == Cap → Resize the array and Add

PushBack(‘e’)



BigO of the Dynamic Array

- PushBack of Dynamic Array requires copying of **n data** from the old array to the new array
- By worst case analysis, O(n)
- *By Amortization analysis (average), O(1)*

Dynamic Array as a ADT

- Dynamic Array should have the following operations (at minimum):
  - **Get**(***i***): Return element at location ***i***
  - **Set**(***i***, ***value***): Sets element i to ***value***
  - **PushBack**(***value***): Adds ***value*** to the end
  - **Remove**(***i***): Removes element at location ***i***

Dynamic Array as a ADT

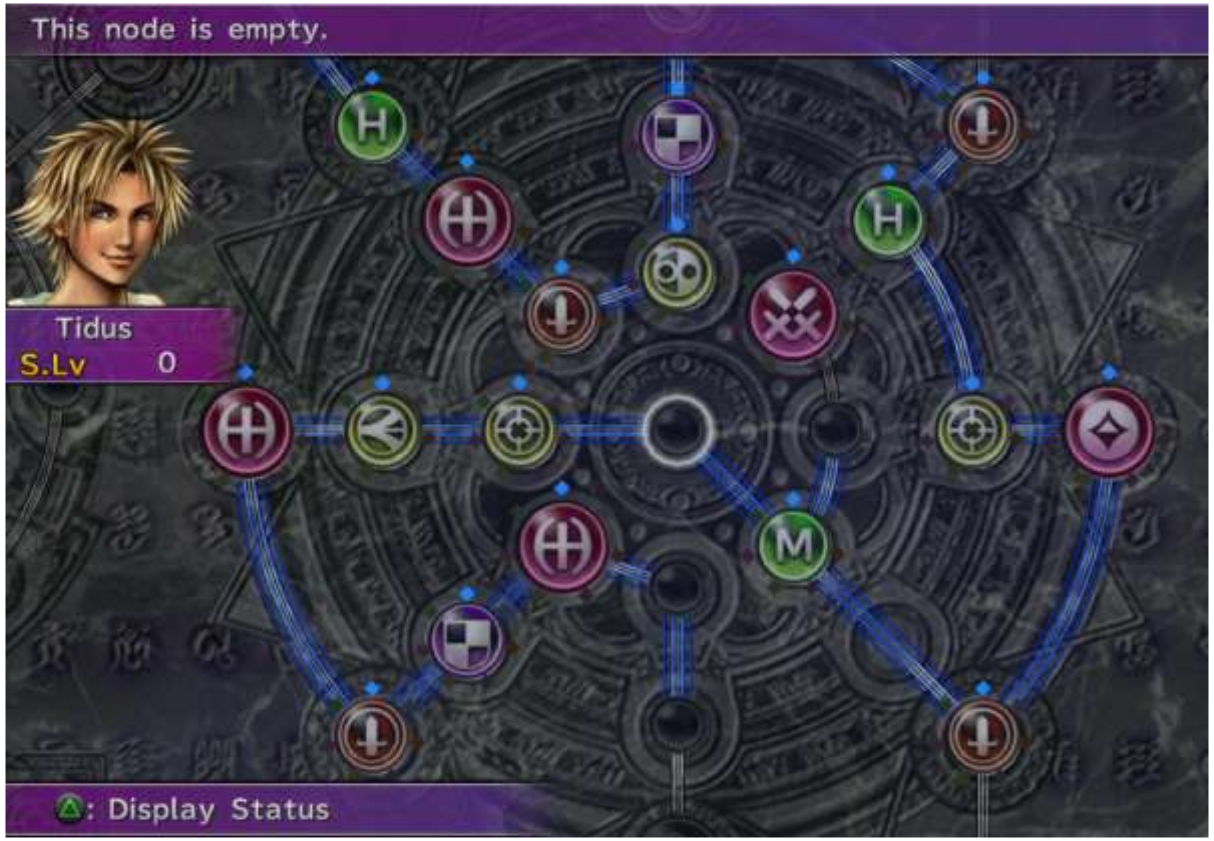
- Dynamic Array should have the following properties (at minimum):
  - ***arr***: dynamically-allocated array reference
  - ***capacity***: size of the dynamically-allocated array
  - ***size***: number of elements currently in the array
  - In the homework, these variables should be set to private, you should implement additional method to return the values.

Homework: Implement Dynamic Array

- Demo with PushBack, PopBack and PrintAll

- Your job is to do the rest

FFX Sphere Grid


This node is empty.

Tidus
S.Lv 0

- Demo: FFX sphere grid sim