Fundamentals of Data Structure

Mahesh Shirole
VJTI, Mumbai-19

Slides are prepared from
1. Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014
2.Data Structures and Algorithms in Java, by Robert Lafore, Second Edition, Sams Publishing

Arrays

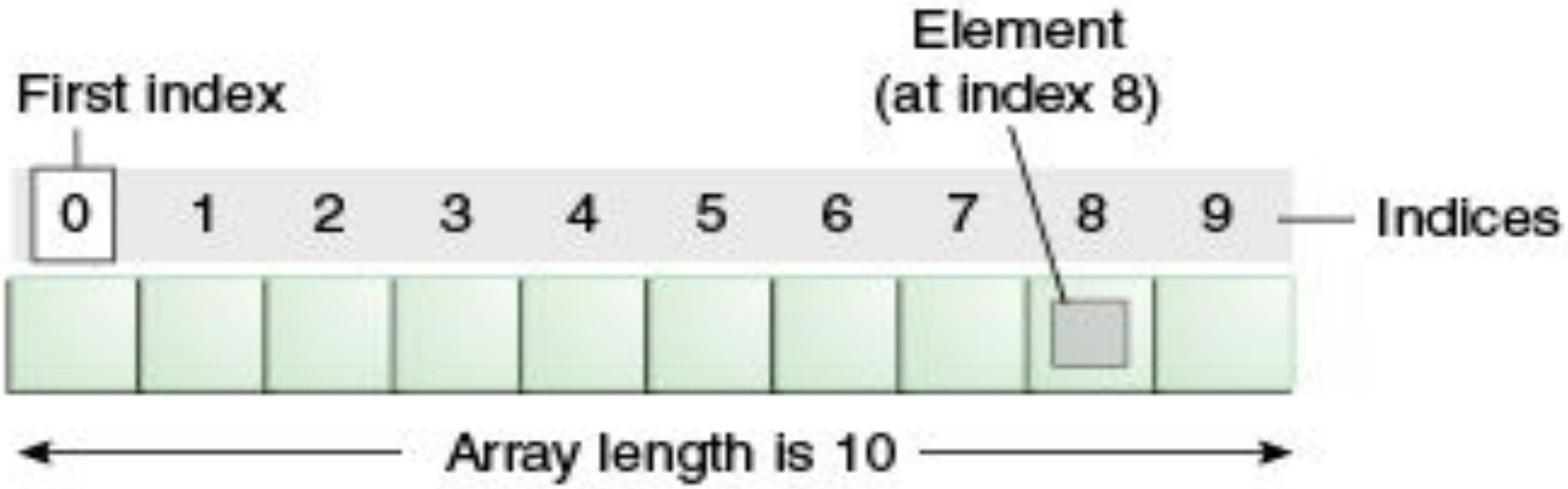
- The array is the most commonly used **data storage structure**; it's built into most programming languages
- A common programming task is to keep track of an ordered sequence of related values or objects
- For example, we may want a video game to keep track of the top ten scores for that game
- Rather than using ten different variables for this task, we would prefer to use a single name for the group and use index numbers to refer to the high scores in that group

Arrays

- An array is a *sequenced collection of variables all of the same type*
- Each variable, or cell, in an array has an **index**, which uniquely refers to the value stored in that cell
- The cells of an array, ***A***, are numbered 0,1,2, and so on
- Each value stored in an array is often called an element of that array

What is capacity of Array?

- The length of an array determines the maximum number of things that can be stored in the array
- The length of an array is its capacity



Operation on Arrays

- Search an element
  - Find data element is available in Array. If available then return its index otherwise null.
- Insert an element
  - Add an element in Array
    - at end of array (last index)
    - at specified index
- Delete an element
  - Find a specified data element and remove from that location
- Length of array
  - Returns number of data elements available in Array

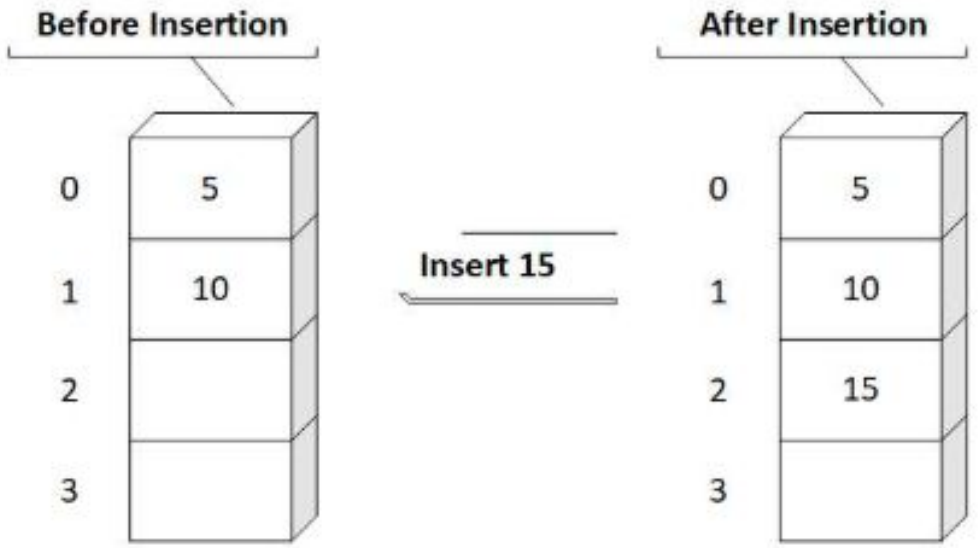
Search an element

- Compare elements from index ***0*** to length of array with specified element
- If element is found at specific index then return the index

- What if duplicate data items are available?
  - Continue search for complete array length and return every index where specified data available
- Analysis
  - If ***N*** is the number of items, the average number of steps needed to find an item is ***N/2***. In the worst-case scenario, the specified item is in the last occupied cell, and ***N*** steps will be required to find it

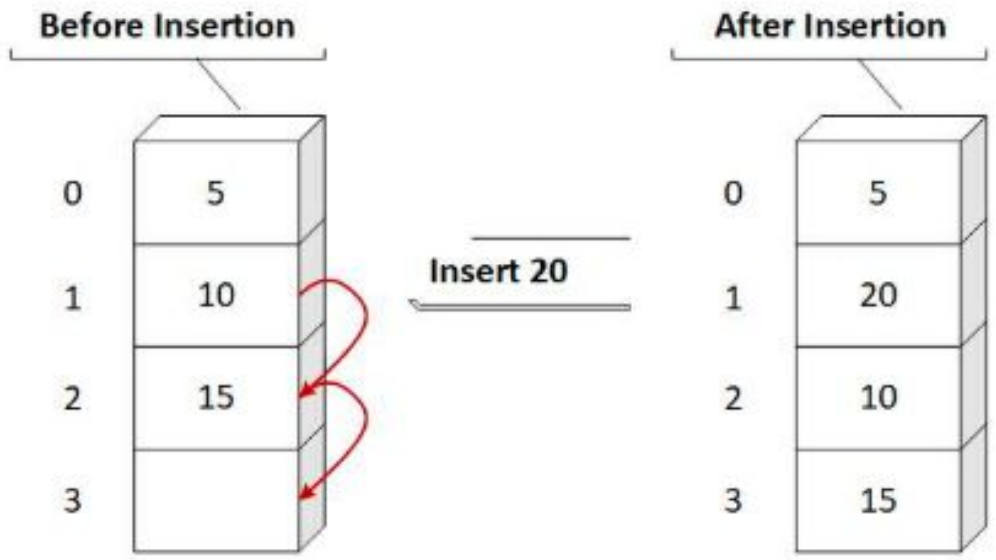
Insert an element

- Add an element data in the first empty cell in the array.
- Analysis
  - The insertion process is very fast, requiring only a single step
  - This is true because a new item is always inserted in the first vacant cell in the array, and the algorithm knows this location because it knows how many items are already in the array. The new item is simply inserted in the next available space



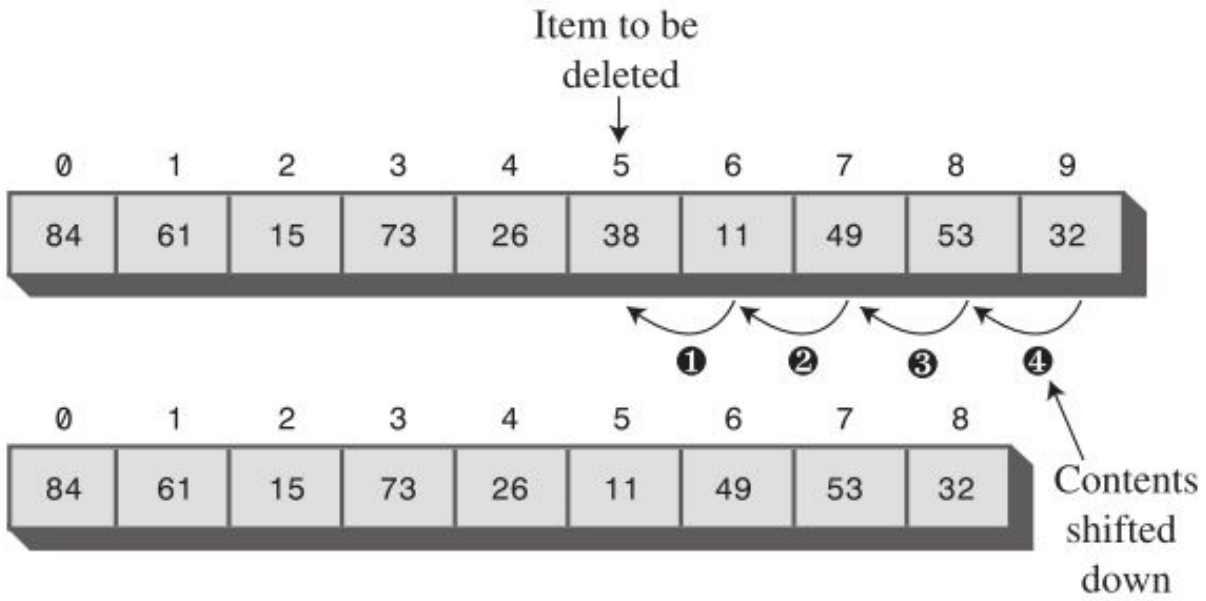
Insert an element at specific location

- Insertion at a given position is a time consuming and expensive process, as it results in shifting the array elements, to make room for a new element
- Analysis
  - an average of ***N/2*** elements shifting



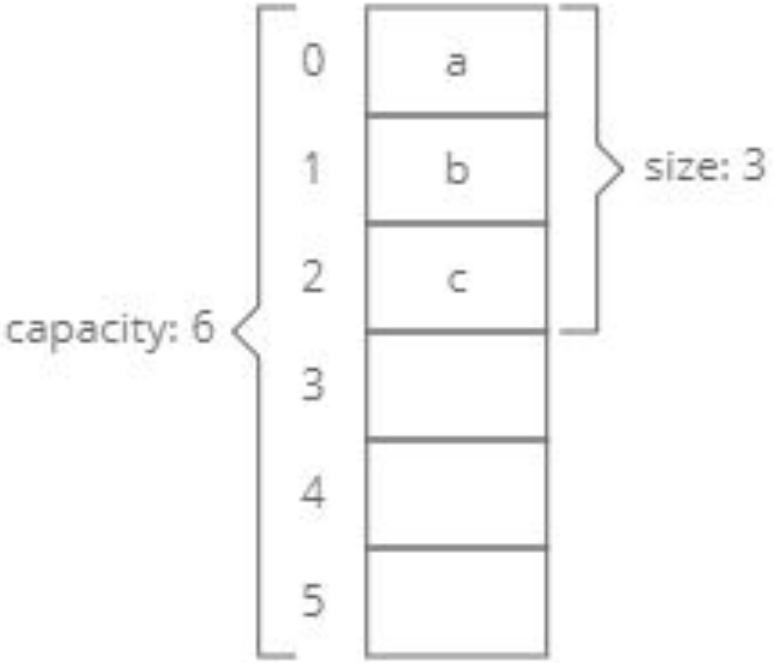
Delete an element

- To delete an item, you must first find it
- Once you deletes the item, the cell becomes empty
- Shift the contents of each subsequent cell down one space to fill in the empty cell
- Analysis: A deletion requires (assuming no duplicates are allowed) searching through an average of ***N/2*** elements and then moving the remaining elements (an average of ***N/2*** moves) to fill up the resulting hole. This is ***N*** steps in all



Length of array

- Array capacity is maximum number of the elements an array can hold
- Array size is number of elements available in an array



Java Arrays

- There are two kinds of data in Java: primitive types (such as int and double) and objects
- In Java Arrays are treated as objects. Accordingly, you must use the new operator to create an array.

```
int[] intArray; // defines a reference to an array
intArray = new int[100]; // creates the array, and sets intArray to refer to it
```

- Java Arrays have a length field, which you can use to find the size (the number of elements) of an array:

```
int arrayLength = intArray.length; // find array
size
```

Accessing Array Elements

- Array elements are accessed using an index number in square brackets. This is similar to how other languages work.
- The first element index is numbered 0, so that the indices in an array of 10 elements run from 0 to 9

```
temp = intArray[3]; // get contents of fourth element of array
```

- Insert an Element

```
intArray[7] = 66; // insert 66 into the eighth cell
```

- If you use an index that's less than 0 or greater than the size of the array less 1, you'll get the *Array Index Out of Bounds* runtime error

Initialization

- Unless you specify otherwise, an array of integers is automatically initialized to ***0*** when it's created.
- Until the array elements are given explicit values, they contain the special null object. If you attempt to access an array element that contains ***null***, you'll get the runtime error ***Null Pointer Assignment***
- You can initialize an array of a primitive type to something besides 0 using this syntax:

```
int[] intArray = { 0, 3, 6, 9, 12, 15, 18, 21, 24, 27 };
```

- This single statement takes the place of both the reference declaration and the use of new to create the array. The numbers within the curly brackets are called the *initialization list*
- The size of the array is determined by the number of values in this list

Home Assignment

- How to handle duplicate items in array for search items, delete items? What is analysis time of operations Search, Insertion, and Deletion for a) with Duplicates and b) Without Duplicates
- What is dynamic array? How to handle growing arrays in implementation?

Lab Assignment

- Design your own long interger array class with following interface. Write a test application for your class.

MyLongArray
- a[] : long - currentIndex: int
+ MyLongArray(int size): void + find(long searchKey) : int + insert(long value) : void + getElem(int index) : long + delete(long value) : boolean + display() : void + dupDelete(long value) : int + insert(int index, long value): void + deleteAt(int index): long