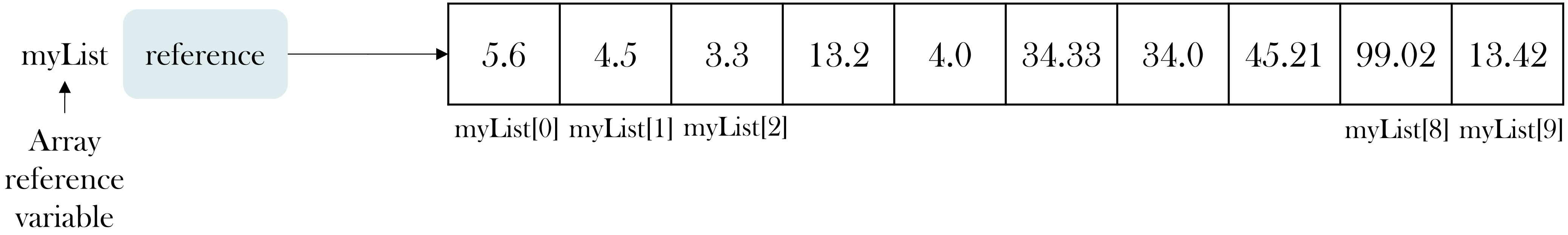
Arrays

Arrays

- Array stores a collection of values
- Data type should be the same for each array element

```
double[] myList; // declare array
myList = new double[10] // create array
myList[0] = 5.6; // put a value to the first position
System.out.println(“First element: ” + myList[0]) // get element
```



myList reference

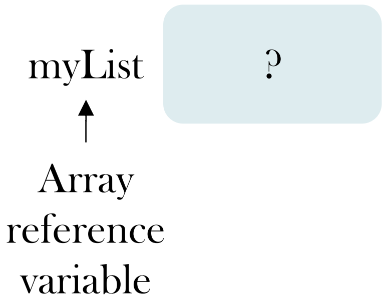
5.6	4.5	3.3	13.2	4.0	34.33	34.0	45.21	99.02	13.42
myList[0]	myList[1]	myList[2]						myList[8]	myList[9]

Array reference variable

Declaring Arrays

- Declaration of an array variable does not allocate any space in memory
- myList is not initialized, means array has not been created yet

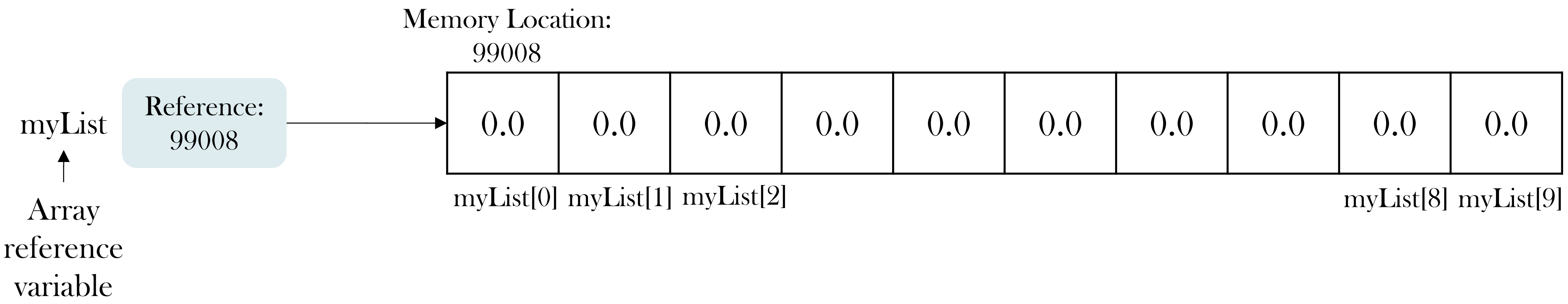
```
double[] myList; // declare array
double myArray[]; // This style is allowed, but not preferred
```



Creating Arrays

- After array declaration, you can create an empty array using the new statement
- Array reference variable (myList) now stores the memory location of the array (reference)
- Each empty array element gets the value of 0.0 for double type (0 for integers)

```
double[] myList; // declare array
myList = new double[10] // create array
```



Memory Location:
99008

myList

Reference:
99008

Array
reference
variable

0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0

myList[0] myList[1] myList[2] myList[8] myList[9]

Creating Arrays

- Declare and create an array in a single step

```
double[] myList = new double[10] // declare and create array
```

- Declare, create and initialize with values in a single step
  - This form is called array initializer

```
double[] myList = {5.6, 4.5, 3.3} // declare, create and initialize
```

Accessing Array Elements

- Accessing array elements
  - The array elements are accessed through the index, e.g., myArray[3]
  - The array indices start from 0 and go up to array length-1
- Once the array is created, its size is fixed: It can not be changed later!
- You can find array size using length: myList.length
- When an array is created, its elements are assigned the default value of
  - 0 for the numeric primitive types,
  - \u0000 for the char types,
  - false for the boolean types

Common Array Operations

- Printing array elements

```
for (int i = 0; i < myList.length; i++) // Using a loop
  System.out.print(myList[i] + " ");

System.out.println(Arrays.toString(myArray)); // Alternative
```

- Initializing array with random values

```
for (int i = 0; i < myList.length; i++)
  myList[i] = Math.random();
```

- Filling array with user inputs

```
Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " double values: ");
for (int i = 0; i < myList.length; i++)
  myList[i] = input.nextDouble();
```

Common Array Operations

- Finding the sum of an array

```
double total = 0;
for (int i = 0; i < myList.length; i++)
  total += myList[i];
```

- Find the maximum value in an array

```
double max = myList[0];
for (int i = 1; i < myList.length; i++)
  if (myList[i] > max)
    max = myList[i];
```

Common Error for Arrays

- Accessing an array out of bounds is a common programming error
  - If you try to access the array elements outside the array index range [0,length-1], you get java.lang.ArrayIndexOutOfBoundsException error
  - To avoid it, make sure you do not use an index beyond length-1 or, less than 0

Array Example

Analyze Numbers

Analyze Numbers

- Read specified number of double numbers from user, compute their average, and find out how many numbers are above the average

```
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter the number of items: ");
int n = input.nextInt();
double[] numbers = new double[n]; // Create an array
double sum = 0;

System.out.print("Enter the numbers: ");
for (int i = 0; i < n; i++) {
  numbers[i] = input.nextDouble();
  sum += numbers[i];
}
double average = sum / n;

int count = 0; // The numbers of elements above average
for (int i = 0; i < n; i++)
  if (numbers[i] > average) // Count if number[i] > average
    count++;

System.out.println("Average is " + average);
System.out.println("Number of elements above the average is ” + count);
```

AnalyzeNumbers

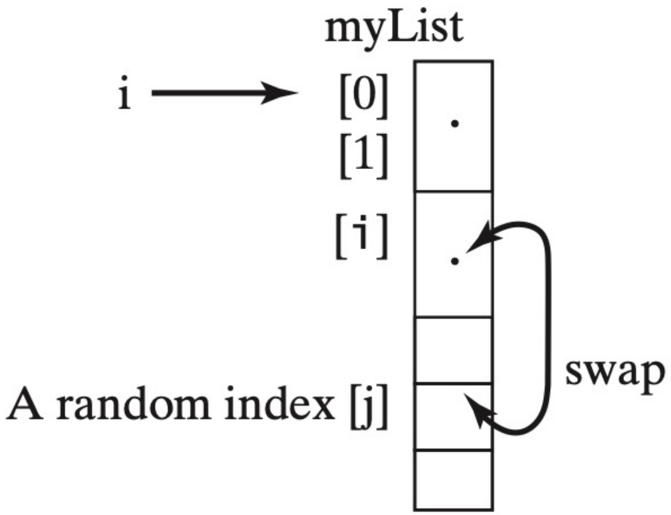
Array Example

Random Shuffle

Random Shuffling

- Algorithm: For each array element, starting from the beginning, pick a random array index and swap their values

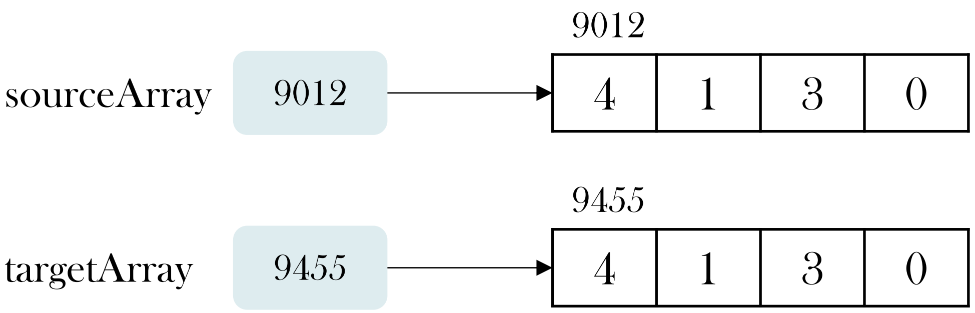
```
int[] myList = {1,2,3,4,5,6,7,8,9,10};

// For each array element, starting from index 0
for (int i = 0; i < myList.length-1; i++) {

  // Pick a random array index
  int j = (int)(Math.random()* myList.length);

  // Swap myList[i] with myList[j]
  int temp = myList[i];
  myList[i] = myList[j];
  myList[j] = temp;
}
```



Copying Arrays

- To copy the contents of one array into another, you must copy the array’s individual elements into the other array

```
int[] sourceArray = {4,1,3,0};
int[] targetArray = new int[sourceArray.length]; // create the second array

for (int i = 0; i < sourceArray.length; i++)
       targetArray[i] = sourceArray[i]; // copy each array element

System.out.println(Arrays.toString(sourceArray)); // print array contents
System.out.println(Arrays.toString(targetArray)); // print array contents
```



Copying Arrays

- Alternative way to copy arrays: Use clone method

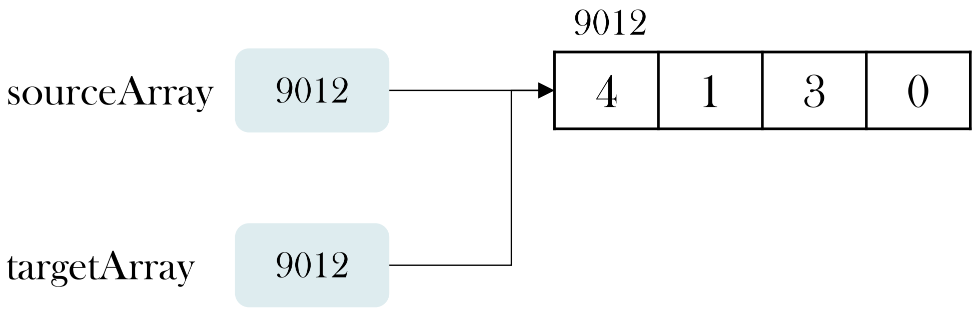
```
int[] sourceArray = {4,1,3,0};
int[] targetArray = sourceArray.clone(); // clone the first array
```

- Another approach to copy arrays: Use System.arraycopy method
  - System.arraycopy(firstArray, int srcPos, secondArray, int destPos, int elementCount)
  - Explanation: Copies elementCount elements from firstArray, starting with position srcPos to secondArray. Copied elements are stored into the secondArray, starting from destPos index

```
int[] sourceArray = {4,1,3,0};
int[] targetArray = new int[sourceArray.length]; // create the second array
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```

Copying Arrays

- `targetArray = sourceArray` does not copy array elements
- Assigning one array variable to another array variable copies array reference to another and makes both variables point to the same memory location
  - sourceArray is an array reference variable, stores the memory location of the array
  - 9012 is the memory location of the sourceArray
  - When you execute `targetArray = sourceArray`, only 9012 is copied to the targetArray variable



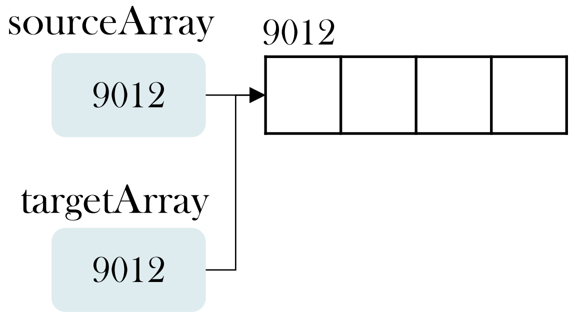
Copying Arrays

- `targetArray = sourceArray` does not copy array elements

```java
int[] array1 = {8,9,3,1};
System.out.println("array 1 before copy: ");
System.out.println(Arrays.toString(array1));

int[] array2 = array1; // try to copy array1 to array2
array2[0] = 2; // change a value in array2

System.out.println("array 1 and array2 after copy: ");
System.out.println("array1: " + Arrays.toString(array1));
System.out.println("array2: " + Arrays.toString(array2));
```

Program output

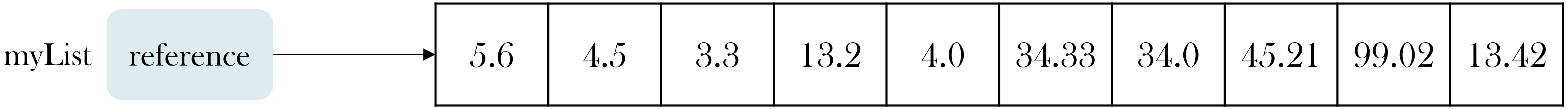
```
array 1 before copy:
[8, 9, 3, 1]
array 1 and array2 after copy:
array1: [2, 9, 3, 1]
array2: [2, 9, 3, 1]
```



Observe that array1 is changed mistakenly!

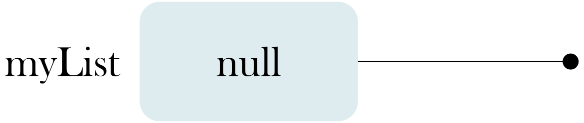
Deleting Array Contents

- You can simply assign `null` value to an array to delete its contents



5.6	4.5	3.3	13.2	4.0	34.33	34.0	45.21	99.02	13.42

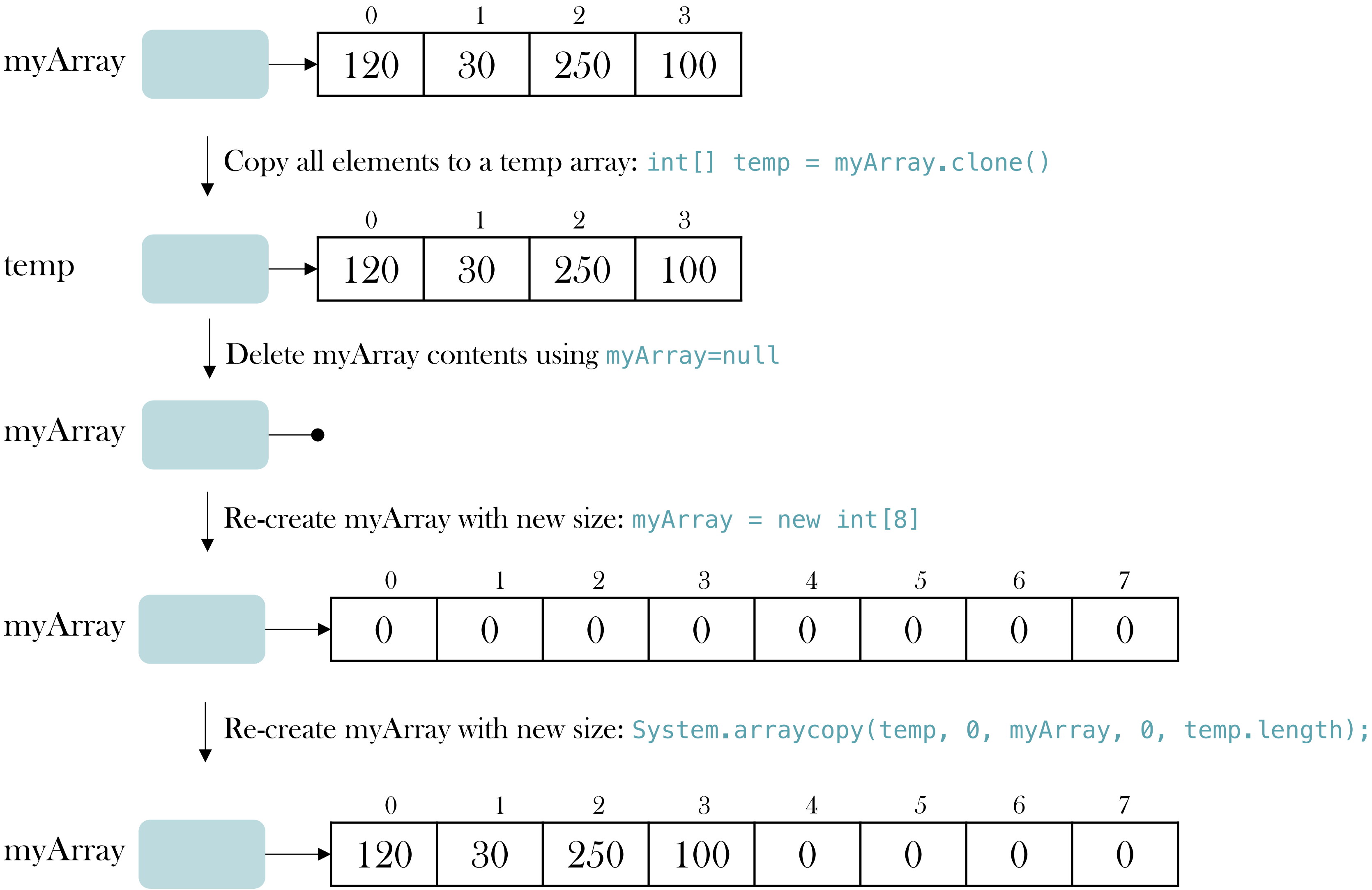
```
myList = null;
```



Increasing Array Size

- In Java, array size can not be change after it's created
- To increase the size of an array
    - Copy array contents to a temporary array
    - Re-create the original array with new size
    - Copy back all the elements from the temporary array to the newly created array

Increasing Array Size



myArray

0	1	2	3
120	30	250	100

Copy all elements to a temp array: `int[] temp = myArray.clone()`

temp

0	1	2	3
120	30	250	100

Delete myArray contents using `myArray=null`

myArray

Re-create myArray with new size: `myArray = new int[8]`

myArray

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

Re-create myArray with new size: `System.arraycopy(temp, 0, myArray, 0, temp.length);`

myArray

0	1	2	3	4	5	6	7
120	30	250	100	0	0	0	0

Shrinking Arrays

- If you want to decrease the size of an array, perform similar operations
  - Create a temporary array which is the clone of the original array
  - Decrease the size of the original array by setting it to null (optional), and re-create with the new size
  - Copy required elements back from temporary array to the newly created array
- Actually, you do not need to assign `null` to the original array: you can simply create the new array

```
int[] myArray = {4,0,12,0,3,0,2,0};
int[] temp = myArray.clone(); // copy all elements to temp array

myArray = new int[4]; // here we skip myArray=null
myArray[0] = temp[0]; // copy nonzero elements
myArray[1] = temp[2];
.
.
```

Traversing Arrays with foreach Loop

- foreach loop enables you to traverse the array sequentially without using an index variable
- Following code displays all the elements in the array myArray
  - You can read the code as “for each element currentElement in myArray, do the following.”
  - The variable, currentElement, must be declared as the same type as the elements in myArray

```
String[] myArray = {“John”, “Robert”, “Alice”, “Bob”};

for (String currentElement: myArray)
  System.out.println(“Element is : ” + currentElement)
```

- You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array
  - With foreach loop, you cannot modify array elements

Traversing Arrays with foreach Loop

- Following codes are equivalent

```
double[] myArray = {1.2, 5.9, 6.2};

for (double e: myArray)
  System.out.println(e)
```

```
double[] myArray = {1.2, 5.9, 6.2};

for (int i = 0; i < myArray.length; i++)
  System.out.println(myArray[i])
```

- If you do not modify array elements and just read array elements in a for loop, you can use foreach loop

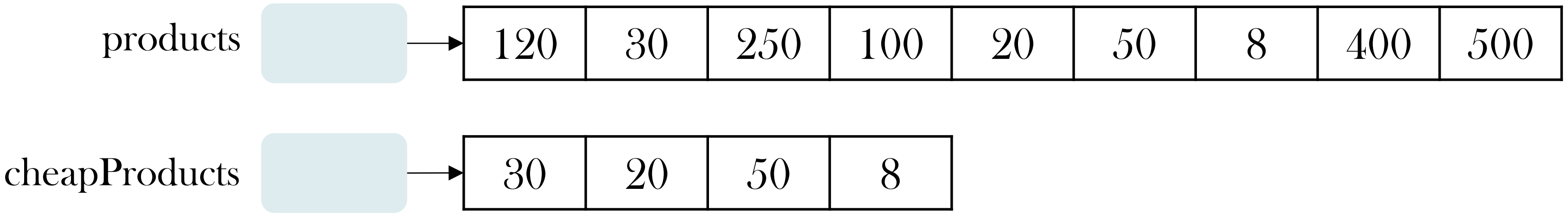
Array Example

Find Cheap Products

Array Example

- Amazon stores product prices in `products` array. Find products that are cheaper than 100TL and store them in `cheapProducts` array

```
public class App {
  public static void main(String[] args) {
    int[] products = {120, 30, 250, 100, 20, 50, 8, 400, 500};
    System.out.println("Products: " + Arrays.toString(products));
  }
}
```



products → 120 | 30 | 250 | 100 | 20 | 50 | 8 | 400 | 500

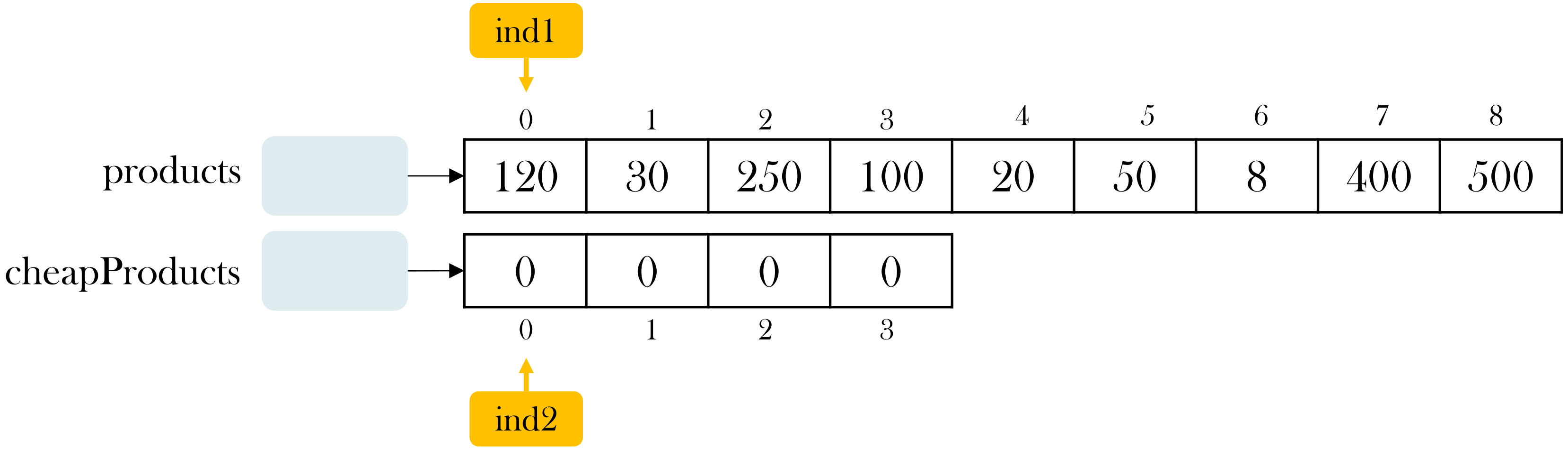
cheapProducts → 30 | 20 | 50 | 8

Array Example

- Solution algorithm
    - Compute the number of cheap products and store this value in a variable `cheapCounter`
    - Create an empty `cheapProducts` array of size `cheapCounter`
    - Traverse along the `products` array from start to end, and store the price of a cheap product into the `cheapProducts` array if its price is less than 100TL.
    - You need to use an array index `ind2` to keep track of the location to store the prices in the `cheapProducts` array

Array Example

- Steps of the algorithm after you found the number of cheap products and created the `cheapProducts` array
  - Let `ind1` points to the current product in products array. Initially 0
  - Let `ind2` points to the current product in cheapProducts array. Initially 0



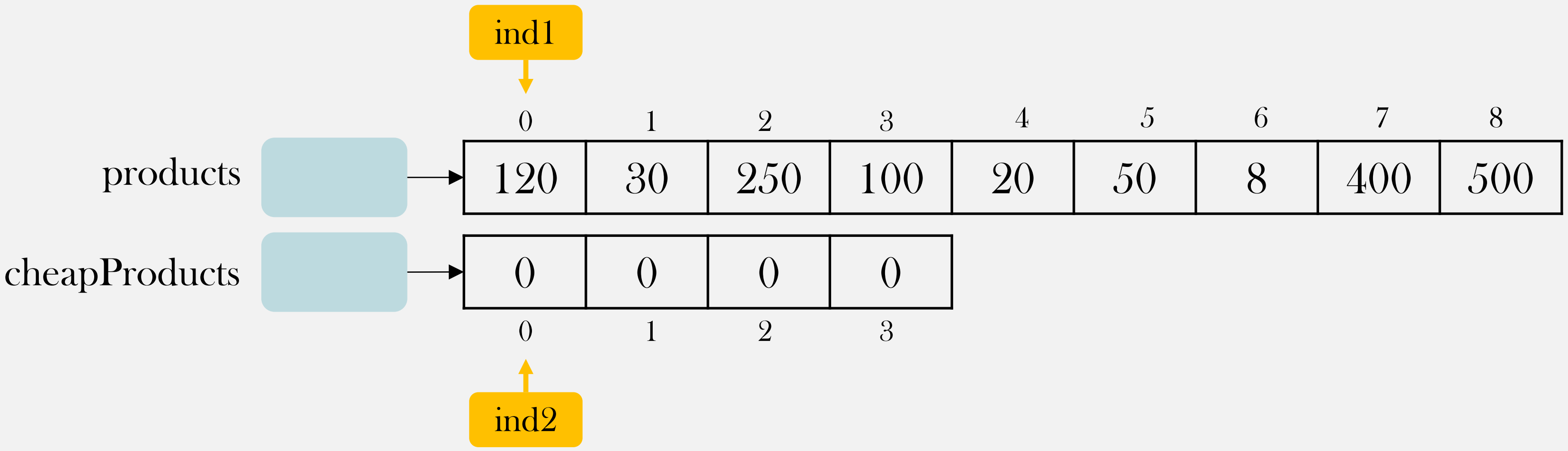
ind1

	0	1	2	3	4	5	6	7	8
products	120	30	250	100	20	50	8	400	500
cheapProducts	0	0	0	0					
	0	1	2	3					

ind2

Array Example

Step 1



ind1

products

0	1	2	3	4	5	6	7	8
120	30	250	100	20	50	8	400	500

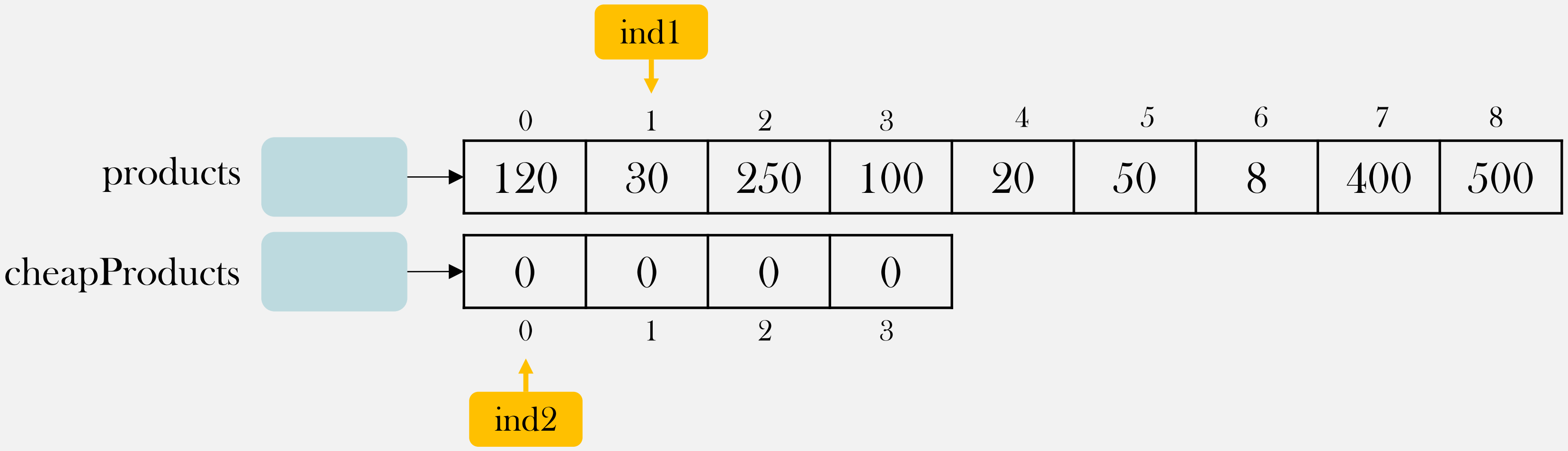
cheapProducts

0	1	2	3
0	0	0	0

ind2

120 is greater than 100. Move ind1 to the right. Do not update ind2.

Step 2



ind1

products

0	1	2	3	4	5	6	7	8
120	30	250	100	20	50	8	400	500

cheapProducts

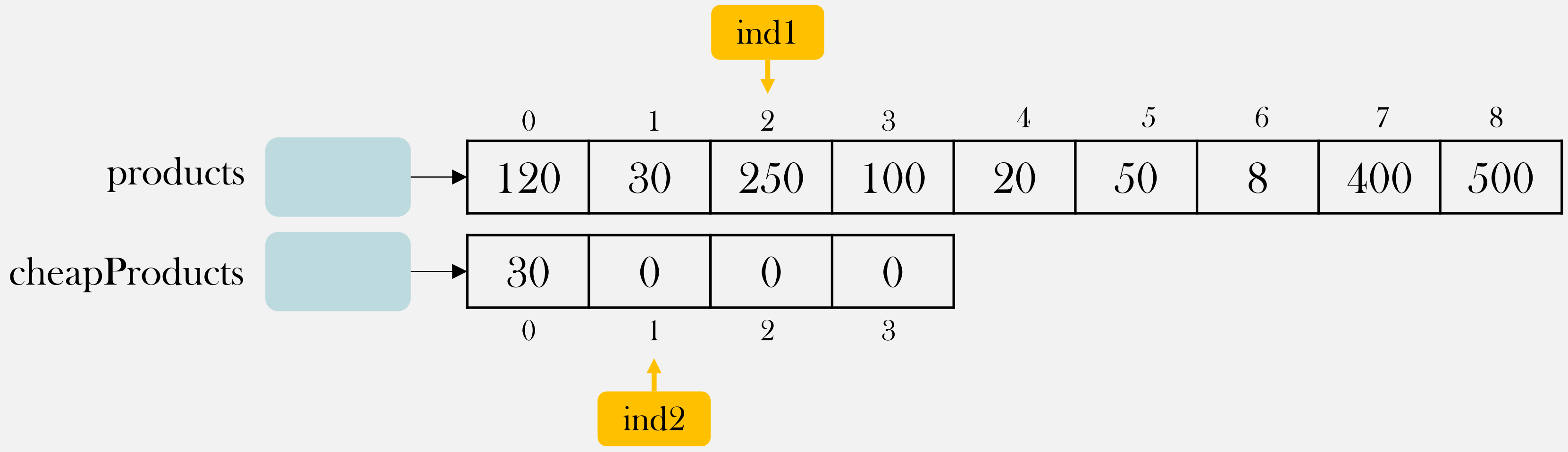
0	1	2	3
0	0	0	0

ind2

30 is less than 100. Place 30 to ind2 location. Update both ind1 and ind2.

Array Example

Step 3



products

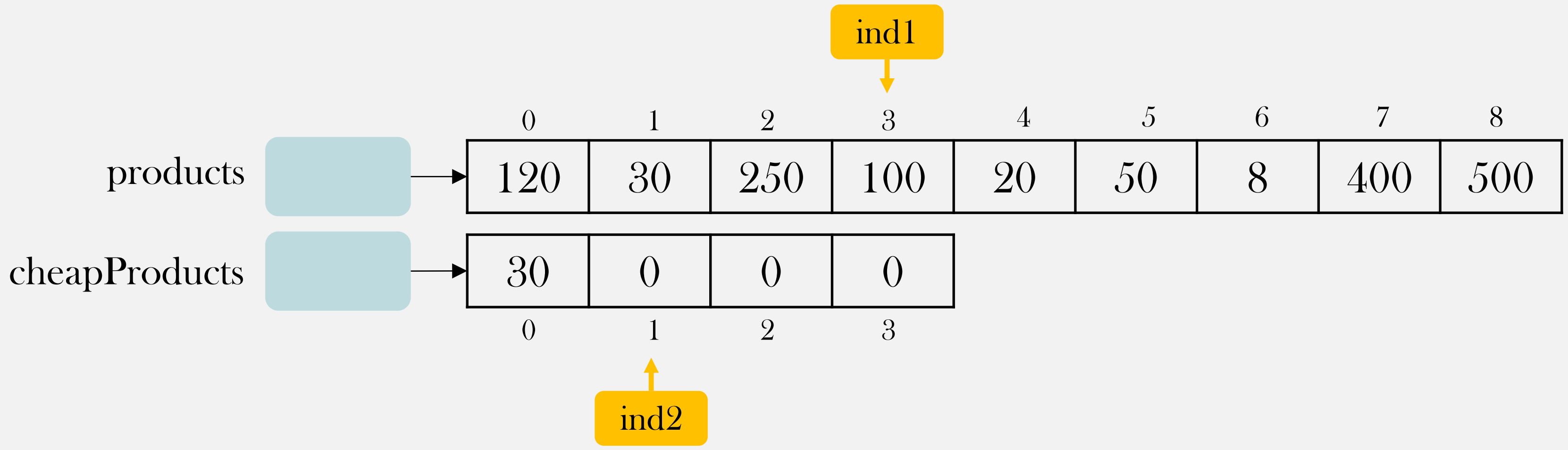
0	1	2	3	4	5	6	7	8
120	30	250	100	20	50	8	400	500

cheapProducts

0	1	2	3
30	0	0	0

250 is greater than 100. Move ind1 to the right. Do not update ind2.

Step 4



products

0	1	2	3	4	5	6	7	8
120	30	250	100	20	50	8	400	500

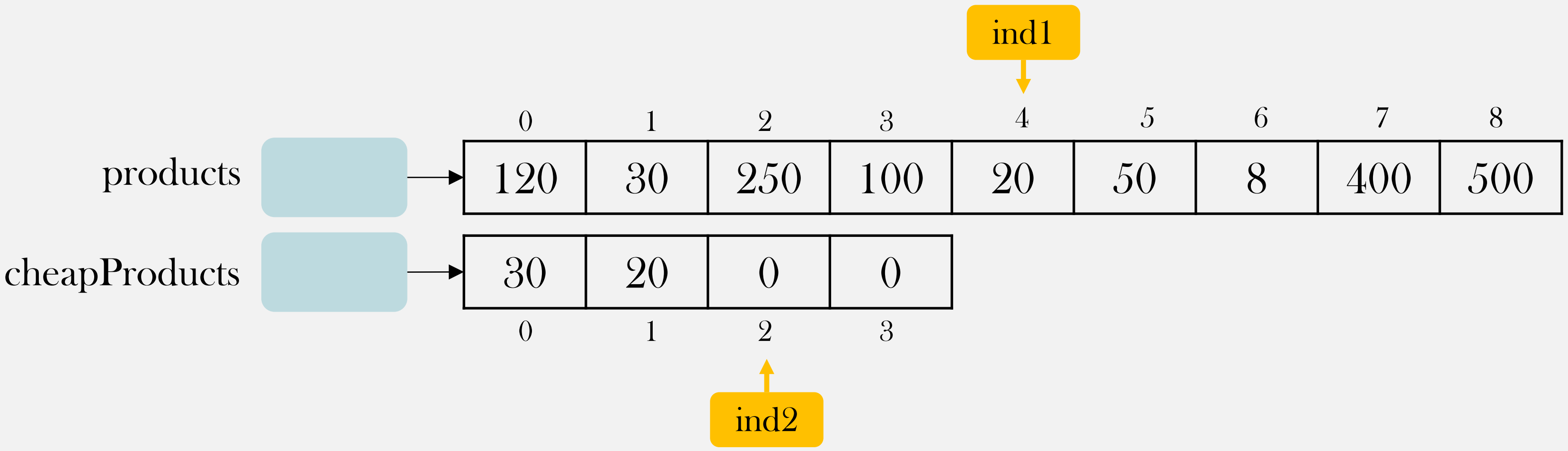
cheapProducts

0	1	2	3
30	0	0	0

100 is not less than 100. Move ind1 to the right. Do not update ind2.

Array Example

Step 5



products

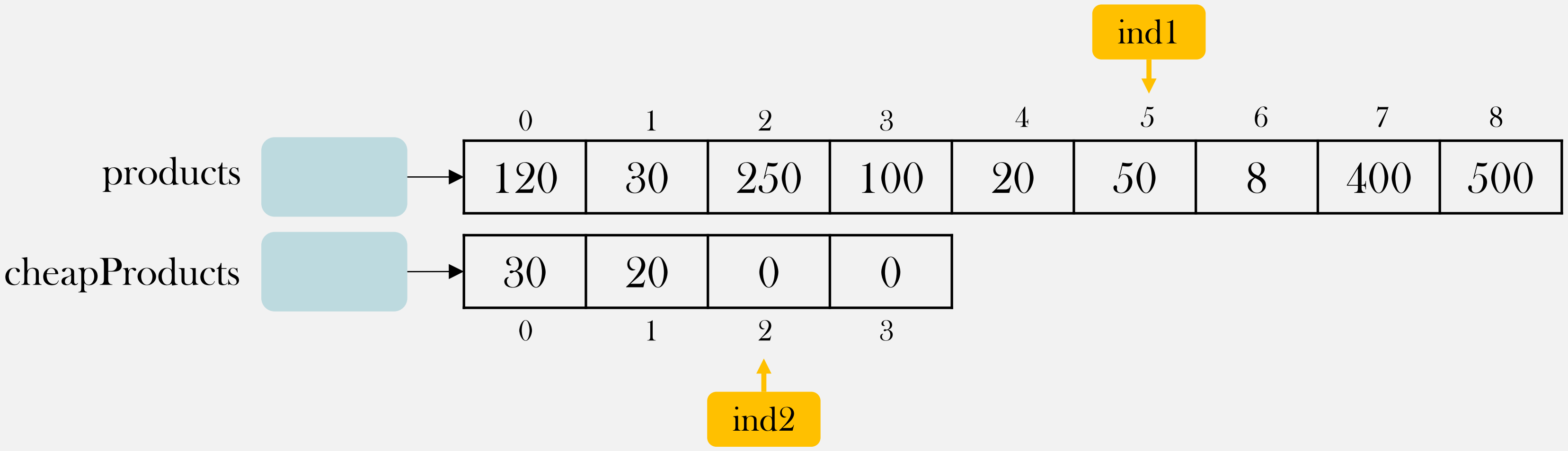
0	1	2	3	4	5	6	7	8
120	30	250	100	20	50	8	400	500

cheapProducts

0	1	2	3
30	20	0	0

20 is less than 100. Place 20 into ind2 location. Update both ind1 and ind2.

Step 6



products

0	1	2	3	4	5	6	7	8
120	30	250	100	20	50	8	400	500

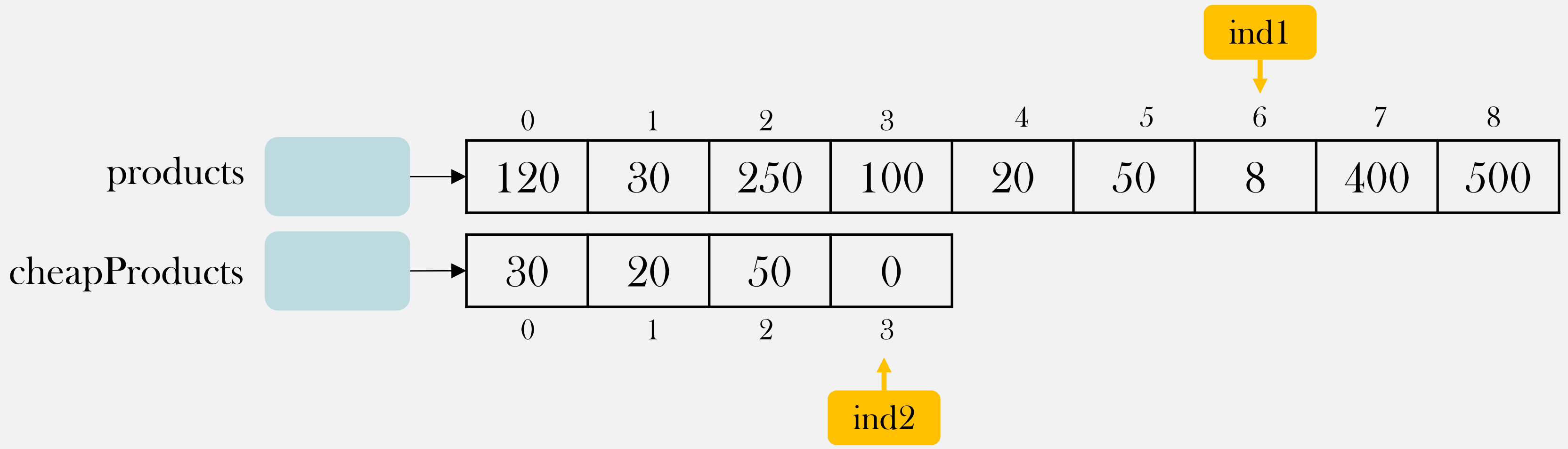
cheapProducts

0	1	2	3
30	20	0	0

50 is less than 100. Place 50 into ind2 location. Update both ind1 and ind2.

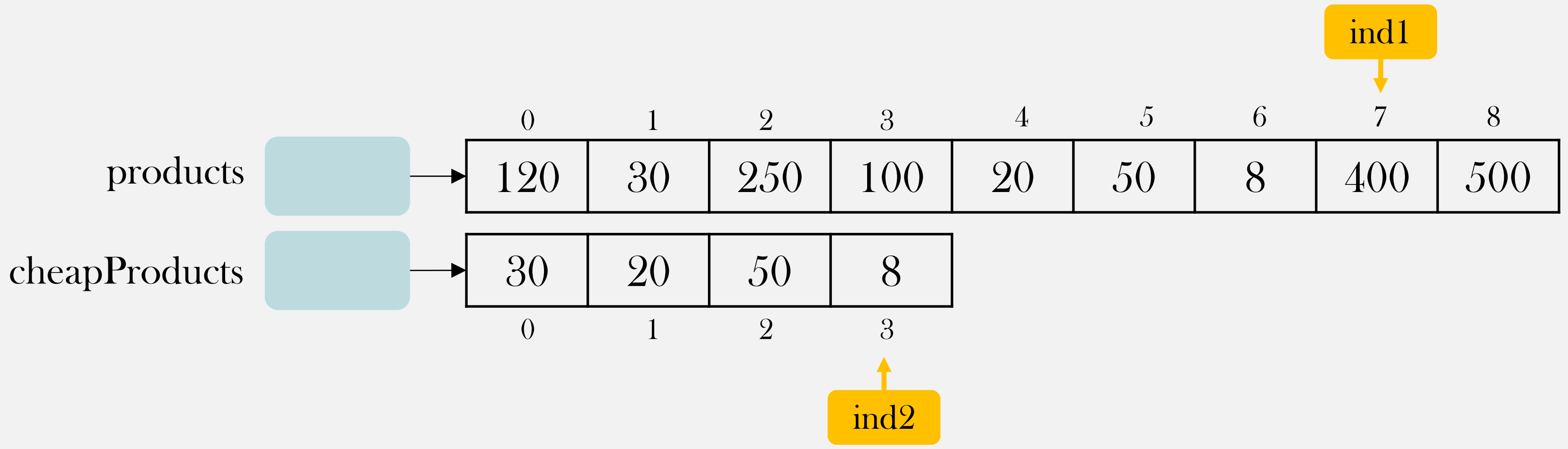
Array Example

Step 7



8 is less than 100. Place 8 into ind2 location. Update both ind1 and ind2.

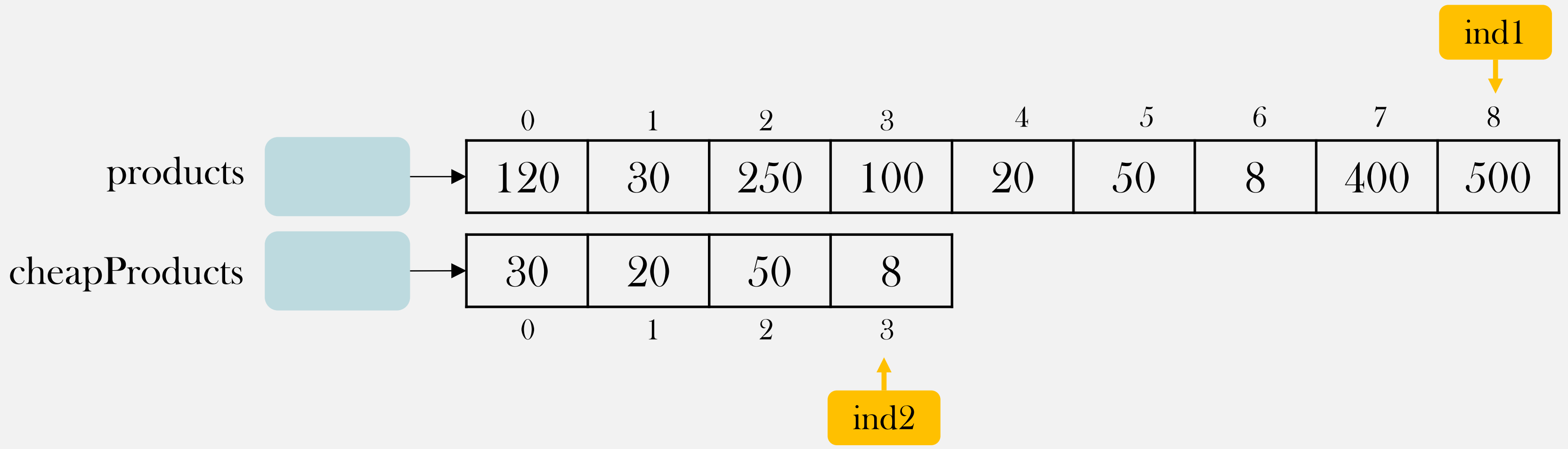
Step 8



400 is not less than 100. Update ind1

Array Example

Step 9



products

0	1	2	3	4	5	6	7	8
120	30	250	100	20	50	8	400	500

cheapProducts

0	1	2	3
30	20	50	8

500 is less than 100

Array Example: Source Code

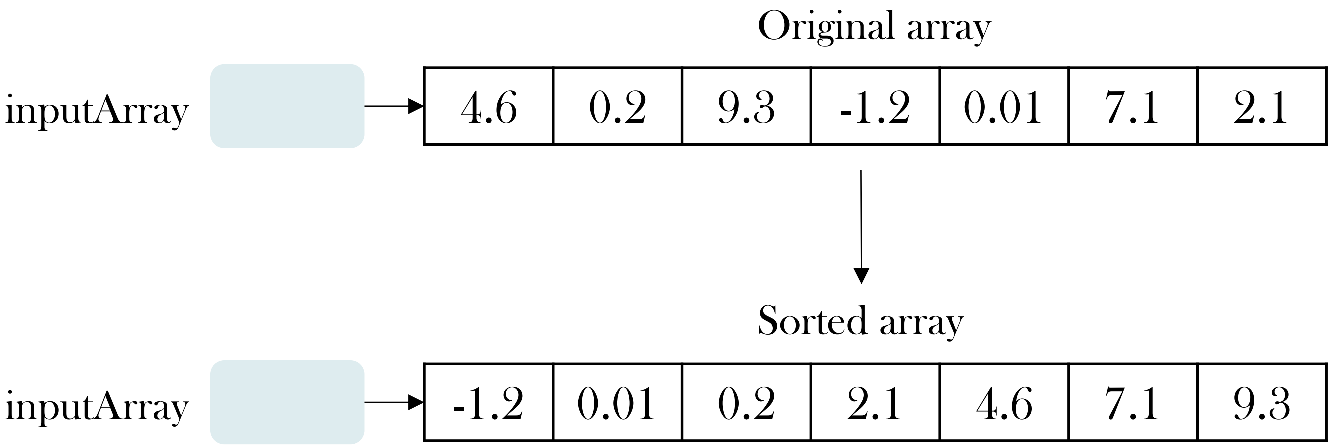
```java
import java.util.Arrays; // required for Arrays.toString method

public class App {
  public static void main(String[] args) {
    int[] products = {120, 30, 250, 100, 20, 50, 8, 400, 500};

    // find the number of cheap products
    int priceThreshold = 100;
    int counter = 0; // stores the number of cheap products
    for (int i = 0; i < products.length; i++)
      if (products[i] < priceThreshold) // if current product is cheap, increment the counter
        counter++;

    int[] cheapProducts = new int[counter]; // create cheapProducts array
    int ind2 = 0; // initialize ind2 index that will be used for cheapProducts array
    for (int ind1 = 0; ind1 < products.length; ind1++)
      if (products[ind1] < priceThreshold)
        cheapProducts[ind2++] = products[ind1]; // if ind1 points to a cheap product,
                                                // place its value into ind2 location and update ind2
    System.out.println("Cheap products: " + Arrays.toString(cheapProducts));
  }
}
```

Array Example

Sort an Array in Increasing Order

Array Example: Sorting

- Sort an array in increasing order
- Do not use or create a new array: modify the input array



Original array

inputArray

4.6	0.2	9.3	-1.2	0.01	7.1	2.1

Sorted array

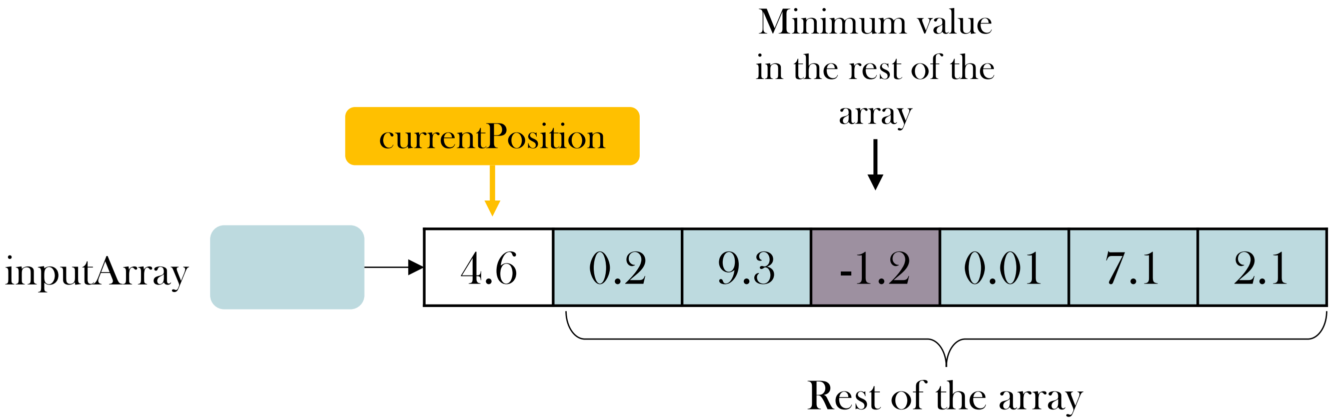
inputArray

-1.2	0.01	0.2	2.1	4.6	7.1	9.3

Array Example: Algorithm

1. Start from the beginning of the array. Let the `currentPosition=0` be the starting index position
2. Find the minimum value, named `minVal`, in the remaining part of the array
3. If `minVal` is less than the current element at index position currentPosition, swap their contents
4. Increment `currentPosition`
5. Perform steps 2 to 4 until `currentPosition` reaches `array length-2`

Array Example: Algorithm Illustration



Minimum value in the rest of the array

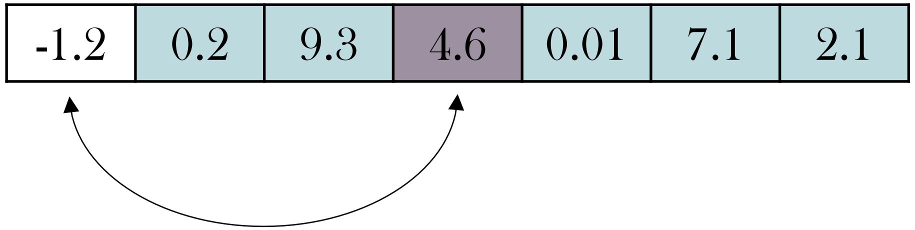
currentPosition

inputArray

4.6	0.2	9.3	-1.2	0.01	7.1	2.1

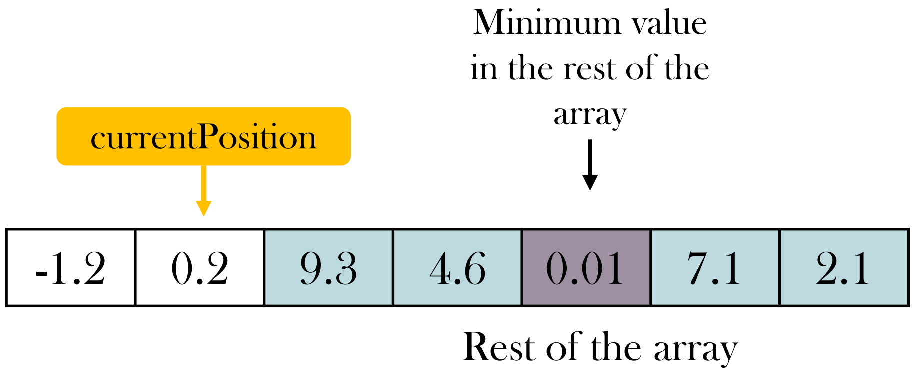
Rest of the array

Since -1.2 is smaller than 4.6, swap them



-1.2	0.2	9.3	4.6	0.01	7.1	2.1

Array Example: Algorithm Illustration



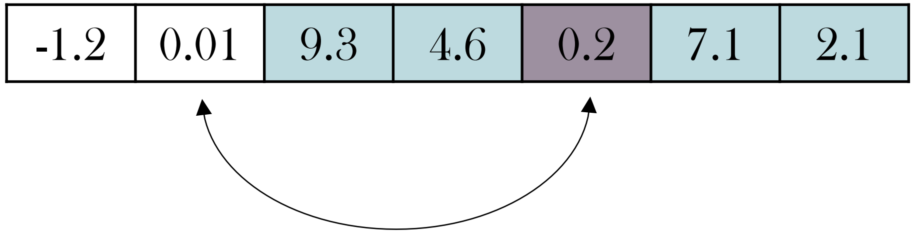
Minimum value in the rest of the array

currentPosition

-1.2	0.2	9.3	4.6	0.01	7.1	2.1

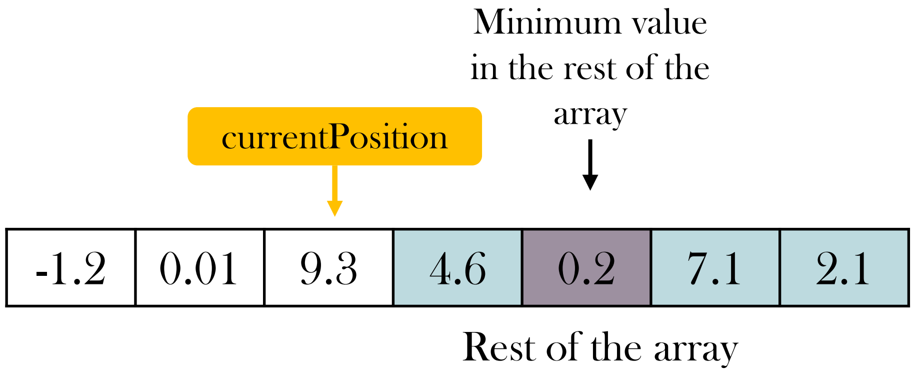
Rest of the array

Since 0.01 is smaller than 0.2, swap them



-1.2	0.01	9.3	4.6	0.2	7.1	2.1

Array Example: Algorithm Illustration



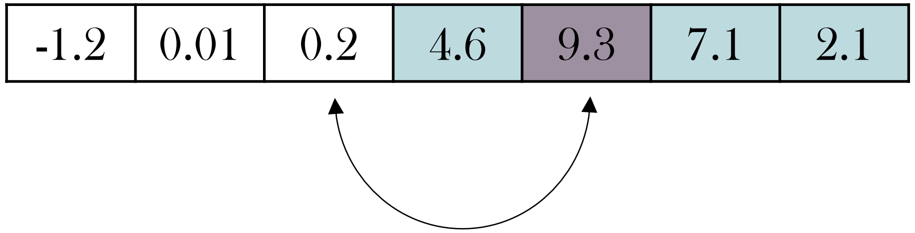
Minimum value in the rest of the array

currentPosition

-1.2	0.01	9.3	4.6	0.2	7.1	2.1

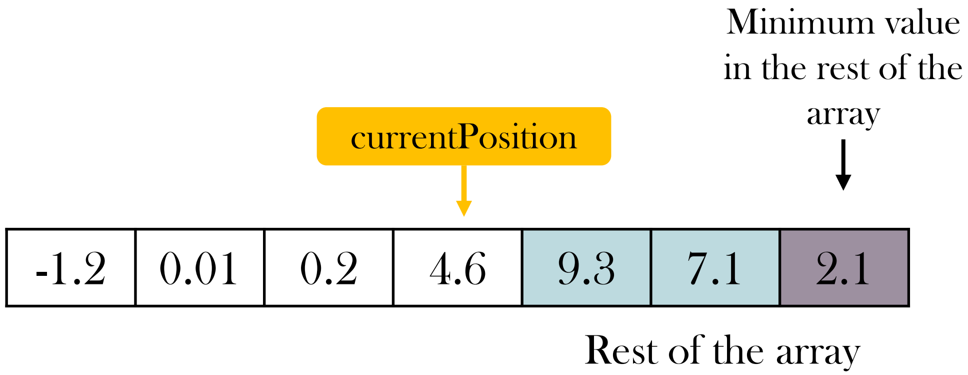
Rest of the array

Since 0.2 is smaller than 9.3, swap them



-1.2	0.01	0.2	4.6	9.3	7.1	2.1

Array Example: Algorithm Illustration



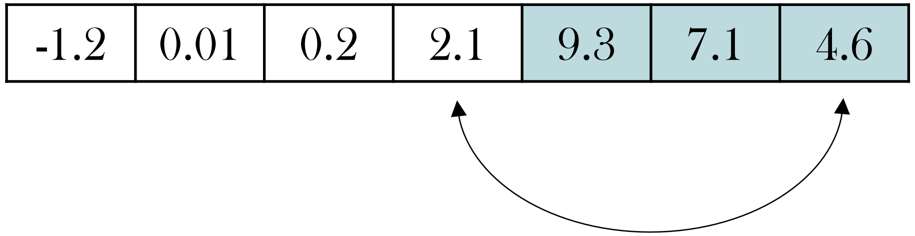
Minimum value in the rest of the array

currentPosition

-1.2	0.01	0.2	4.6	9.3	7.1	2.1

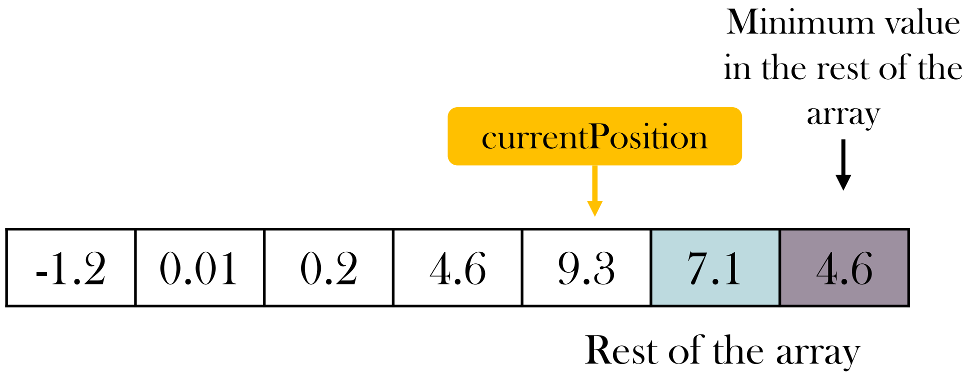
Rest of the array

Since 2.1 is smaller than 4.6, swap them



-1.2	0.01	0.2	2.1	9.3	7.1	4.6

Array Example: Algorithm Illustration



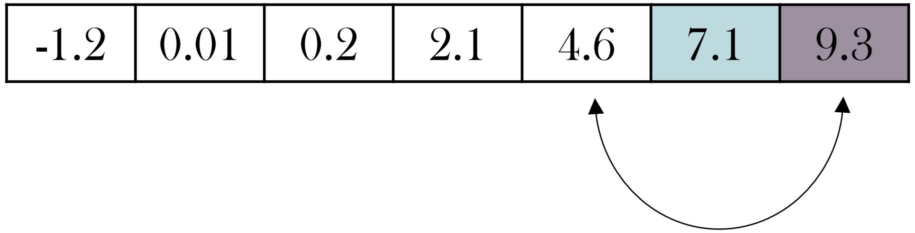
Minimum value in the rest of the array

currentPosition

-1.2	0.01	0.2	4.6	9.3	7.1	4.6

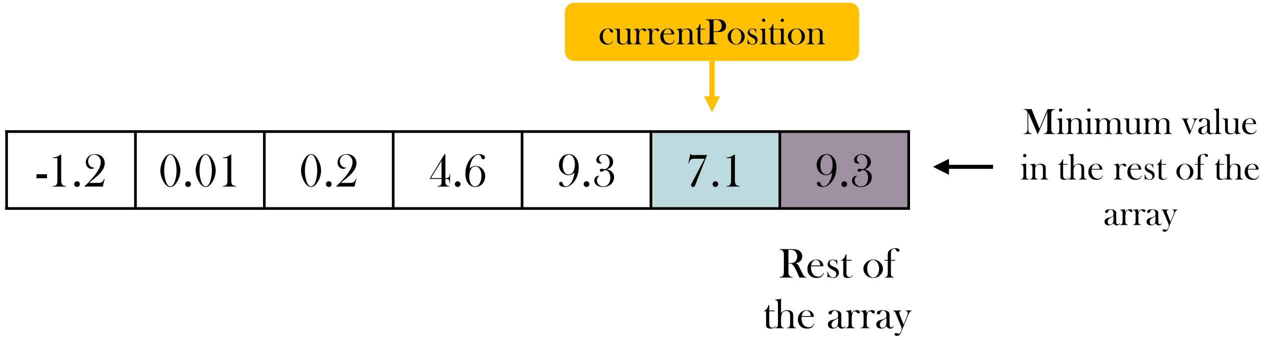
Rest of the array

Since 4.6 is smaller than 9.3, swap them



-1.2	0.01	0.2	2.1	4.6	7.1	9.3

Array Example: Algorithm Illustration



currentPosition

-1.2	0.01	0.2	4.6	9.3	7.1	9.3

Minimum value in the rest of the array

Rest of the array

Since 9.3 is greater than 7.1, do nothing

-1.2	0.01	0.2	2.1	4.6	7.1	9.3
0	1	2	3	4	5	6

currentPosition reaches `array length-2`, which is 5, and the algorithm stops

Array Example: Source Code

```
public class AppSort {
 public static void main(String[] args) {
  double[] inputArray  = {4.6, 0.2, 9.3, -1.2, 0.01, 7.1, 2.1};

  // currentPosition starts from the beginning and iterates until array length-2
  for (int currentPosition = 0; currentPosition <= inputArray.length-2; currentPosition++) {
   double currentElement = inputArray[currentPosition]; // store the current element in a variable

   // find the smallest element in the rest of the array
   double minVal = inputArray[currentPosition+1]; // minVal will be the minimum value in the rest of the array
   int minIndex = currentPosition + 1; // minIndex stores the index location of minVal
   for (int i = currentPosition+2; i < inputArray.length; i++)
    if (inputArray[i] < minVal) { // if the ith element is smaller than minVal, update minVal and minIndex
     minVal = inputArray[i];
     minIndex = i;
    }
   // swap currentElement with minVal if minVal is smaller than currentElement
   if (minVal < currentElement) {
    inputArray[currentPosition] = minVal;
    inputArray[minIndex] = currentElement;
   }
  }
 }
}
```

Array Example

Best Selling Product

Array Example: Best Selling Product

- Amazon sells 10 products and stores all transaction information for sold items in arrays
    - Products: Nike, Adidas, Vans, Converse, Puma, NewBalance, Asics, Slazenger, Columbia, NorthFace
    - Transaction information: Product name and price

products	itemSold	priceSold
”Nike”	”Vans”	300
“Adidas”	“Vans”	400
“Vans”	“Adidas”	350
“Converse”	“Asics”	450
“Puma”	“Asics”	250
“NewBalance”	“Asics”	250
“Asics”	“Puma”	350
“Slazenger”	“NorthFace”	475
“Columbia”	“Vans”	325
“NorthFace”	“Vans”	225
	“Adidas”	600
	“Asics”	700
	“Vans”	150

Means: a Vans shoe is sold for 300TL

Array Example: Best Selling Product

- Find and print the product that has the highest total sell value
  - In the example below, total sell value of product Asics is the highest with 1650TL
  - Vans: 1400TL, Adidas: 950TL, Puma: 350TL, NorthFace: 475TL. Other products are zero.

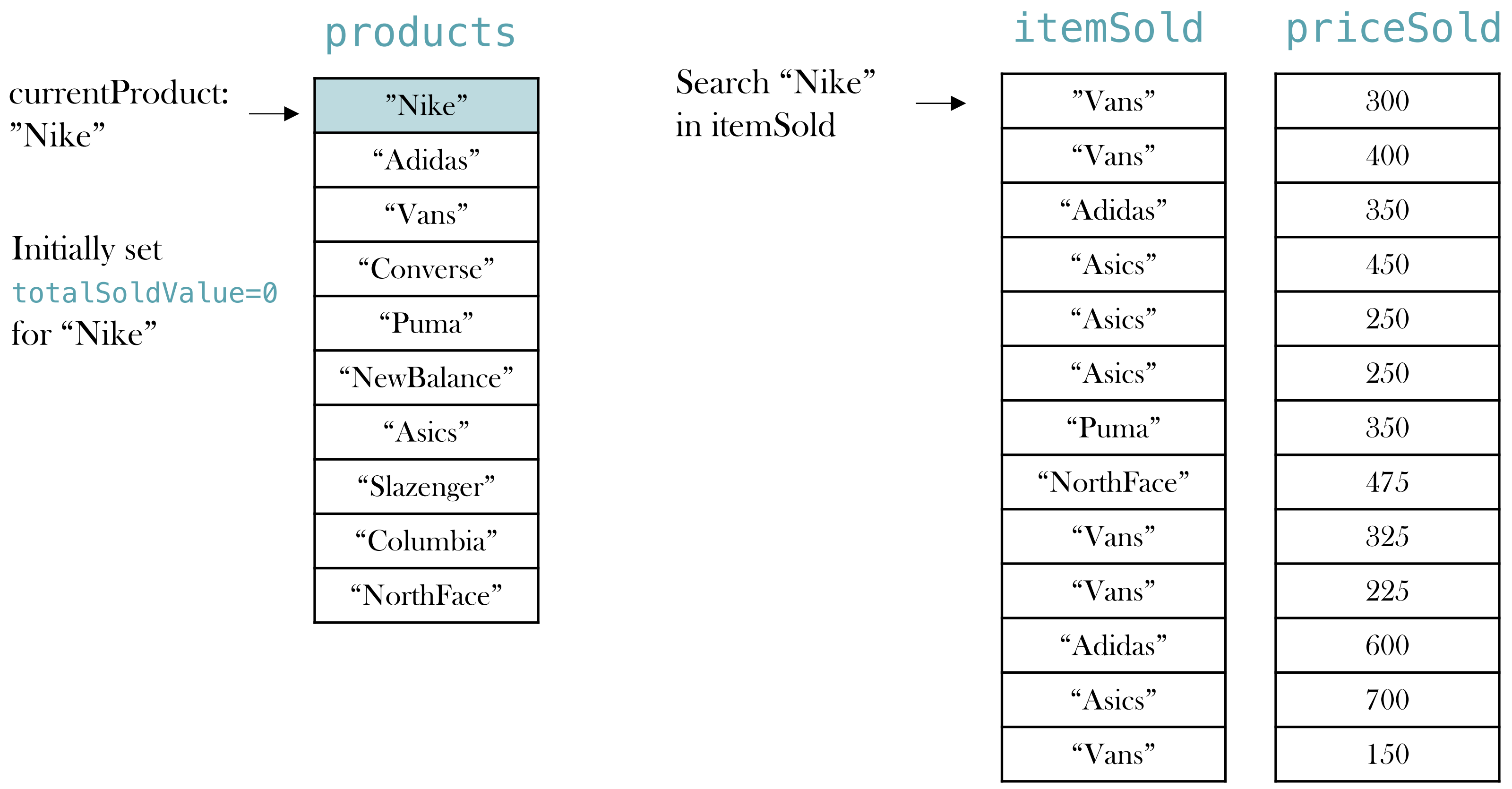
products
”Nike”
“Adidas”
“Vans”
“Converse”
“Puma”
“NewBalance”
“Asics”
“Slazenger”
“Columbia”
“NorthFace”

itemSold	priceSold
”Vans”	300
“Vans”	400
“Adidas”	350
“Asics”	450
“Asics”	250
“Asics”	250
“Puma”	350
“NorthFace”	475
“Vans”	325
“Vans”	225
“Adidas”	600
“Asics”	700
“Vans”	150

Solution Algorithm

- For each product in the `products` array, search the `itemSold` array
- If the current product is found in the `itemSold` array, increase its total value by using the corresponding price in the `priceSold` array
- After a search for each product is finished, check whether the total value is maximum
- If, for the current product, the total value sum is highest, store product name and total value in special variables: `maxProductName` and `maxTotalValue`
- Print `maxProductName` and `maxTotalValue`

Algorithm Illustration



currentProduct:
”Nike” →

Initially set
`totalSoldValue=0`
for “Nike”

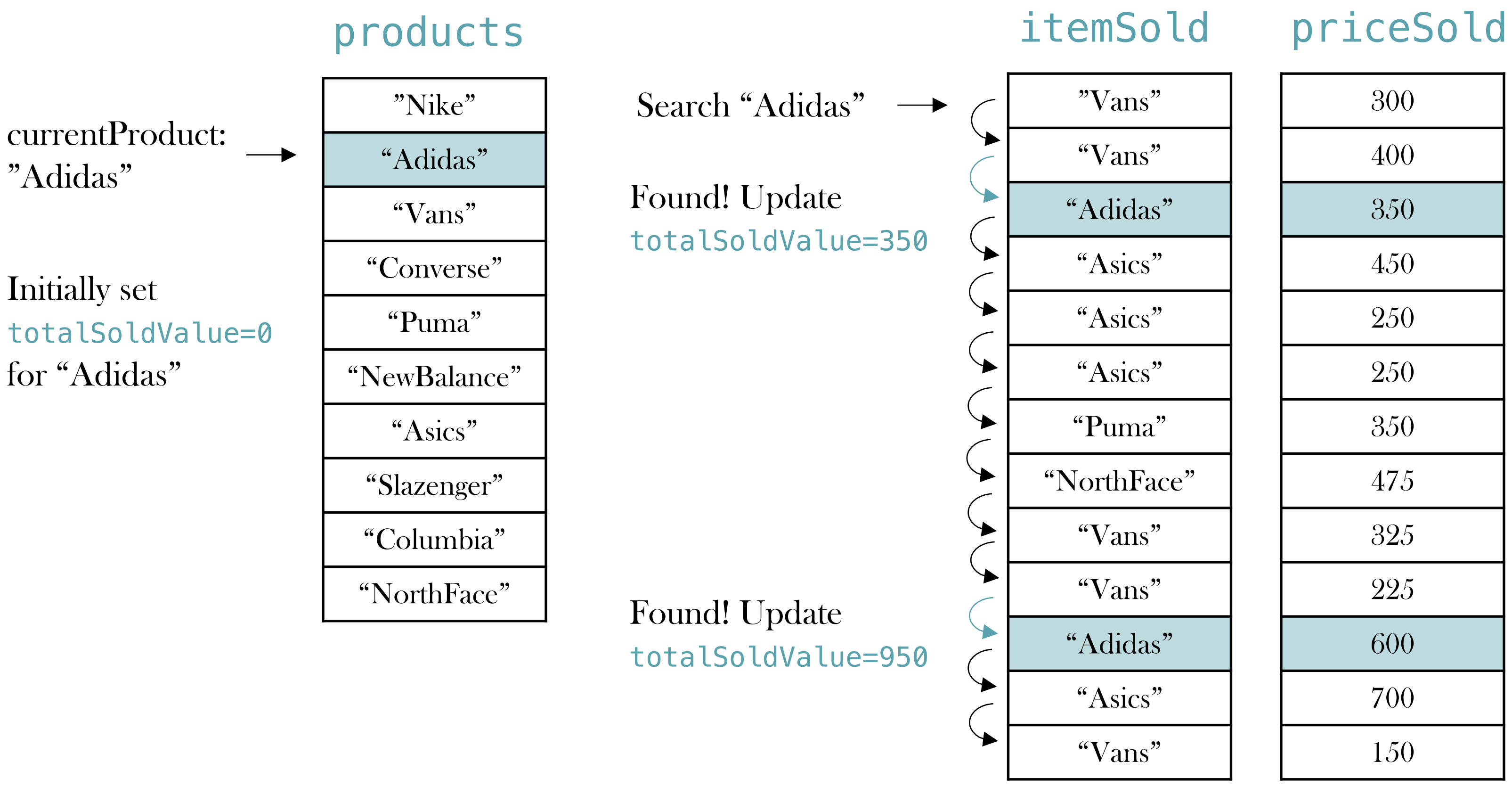
products
”Nike”
“Adidas”
“Vans”
“Converse”
“Puma”
“NewBalance”
“Asics”
“Slazenger”
“Columbia”
“NorthFace”

Search “Nike”
in itemSold →

itemSold	priceSold
”Vans”	300
“Vans”	400
“Adidas”	350
“Asics”	450
“Asics”	250
“Asics”	250
“Puma”	350
“NorthFace”	475
“Vans”	325
“Vans”	225
“Adidas”	600
“Asics”	700
“Vans”	150

After searching “Nike”, `totalSoldValue` will still be zero since no Nike sold in Amazon

Algorithm Illustration



currentProduct:
"Adidas" →

Initially set
`totalSoldValue=0`
for "Adidas"

products
"Nike"
"Adidas"
"Vans"
"Converse"
"Puma"
"NewBalance"
"Asics"
"Slazenger"
"Columbia"
"NorthFace"

Search "Adidas" →

Found! Update
`totalSoldValue=350`

Found! Update
`totalSoldValue=950`

itemSold	priceSold
"Vans"	300
"Vans"	400
"Adidas"	350
"Asics"	450
"Asics"	250
"Asics"	250
"Puma"	350
"NorthFace"	475
"Vans"	325
"Vans"	225
"Adidas"	600
"Asics"	700
"Vans"	150

After searching "Adidas", `totalSoldValue` will still be 950TL

Source Code

```
public class AppBestSellers {
  public static void main(String[] args) {
    String[] products = {"Nike", "Adidas", "Vans", "Converse", "Puma", "NewBalance", "Asics", "Slazenger", "Columbia", "NorthFace"};
    String[] itemSold = {"Vans","Vans","Adidas","Asics","Asics","Asics","Puma","NorthFace","Vans","Vans","Adidas","Asics","Vans"};
    int[] priceSold = {300,400,350,450,250,250,350,475,325,225,600,700,150};

    String maxProductName = null; // initially set best seller product name to null
    int maxTotalValue = -1; // initially set best seller product total value to -1

    for (int i = 0; i < products.length; i++) { // for each product, perform search
      String currentProduct = products[i]; // set the current product
      int totalSoldValue = 0; // initialize total sold value to zero

      for (int j = 0; j < itemSold.length; j++)
        if (currentProduct.contentEquals(itemSold[j])) // if product is found in the itemSold array, increare totalSoldValue
          totalSoldValue = totalSoldValue + priceSold[j];
      System.out.printf("%-10s : %5d TL\n", currentProduct, totalSoldValue);

      if (totalSoldValue > maxTotalValue) { // after the search is done per product, check whether total sold value is maximum
        maxProductName = currentProduct; // if so, update maxProductName and maxTotalValue
        maxTotalValue = totalSoldValue;
      }
    }
    System.out.printf("\nBest selling product is %s (%d TL)\n", maxProductName, maxTotalValue); // print best selling product and total value
  }
}
```

Program Output

Program output

```
Nike       :      0 TL
Adidas     :    950 TL
Vans       :   1400 TL
Converse   :      0 TL
Puma       :    350 TL
NewBalance :      0 TL
Asics      :   1650 TL
Slazenger  :      0 TL
Columbia   :      0 TL
NorthFace  :    475 TL

Best selling product is Asics (1650 TL)
```

Multi-dimensional Arrays

Multi-dimensional Arrays

- Arrays in Java can have multiple dimensions
- Two-dimensional array is very common

```
int[][] personInfo = new int[6][4] // 6 rows and 4 columns
personInfo[0][0] = 23;     // set John’s age
personInfo[0][1] = 191;    // set John’s height
personInfo[0][2] = 89;     // set John’s weight
personInfo[0][3] = 34010;  // set John’s postcode

personInfo[5][3] = 32910;  // set Jesica’s postcode
```

Columns

			Age	Height	Weight	Postcode
			0	1	2	3
Rows	John	0	23	191	89	34010
	Alice	1	20	180	75	34200
	Robert	2	25	174	69	31050
	Sarah	3	19	167	69	31120
	Bob	4	22	178	80	32600
	Jesica	5	23	186	84	32910

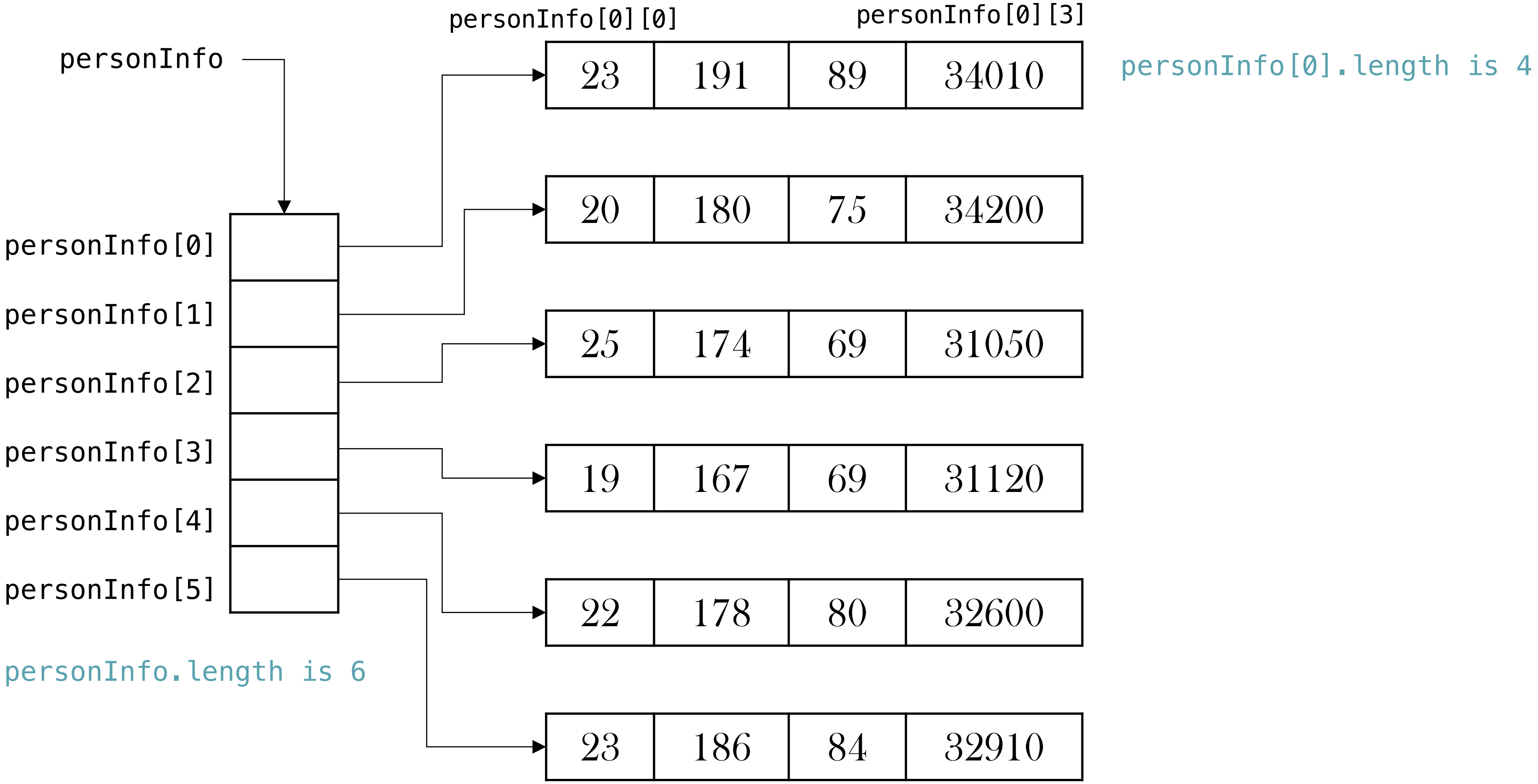
- 2D array initializer

```
int[][] personInfo = {
 {23,191,89,34010},
 {20,180,75,34200},
 {25,174,69,31050},
 {19,167,69,31120},
 {22,178,80,32600},
 {23,186,84,32910},
};
```

```
// Getting row size
int rowSize = personInfo.length;

// Getting column size
int colSize = personInfo[0].length;
```

Structure of 2D Arrays

- Each row in a two-dimensional array is itself an array



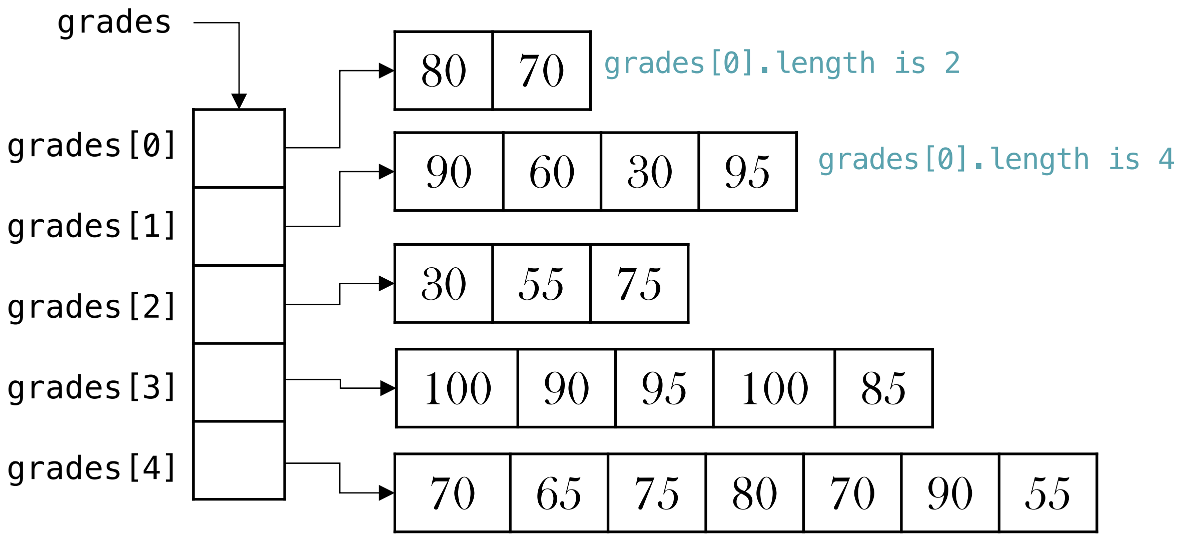
23	191	89	34010
20	180	75	34200
25	174	69	31050
19	167	69	31120
22	178	80	32600
23	186	84	32910

`personInfo[0].length is 4`

`personInfo.length is 6`

Ragged Arrays

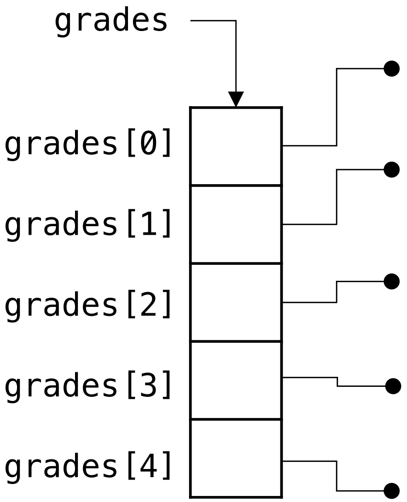
- Each row in a two-dimensional array is itself an array
- Rows can have different lengths: Such an array is known as a ragged array



Ragged Arrays

- Each row in a two-dimensional array is itself an array
- Rows can have different lengths: Such an array is known as a ragged array

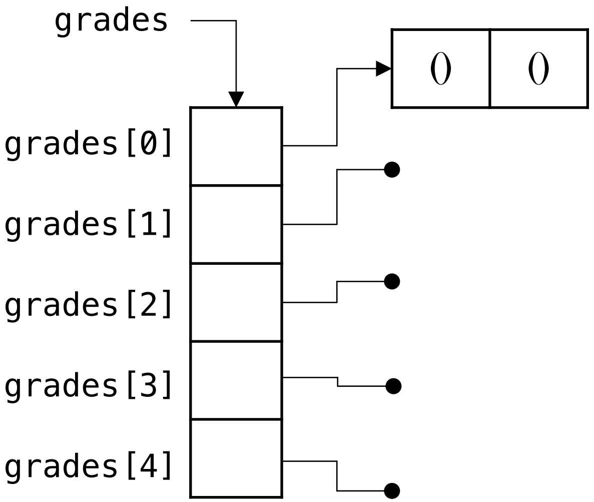
```
// create ragged array. Do not specify column size
int[][] grades = new int[5][]; // 5 rows, unknown columns
```



Ragged Arrays

- Each row in a two-dimensional array is itself an array
- Rows can have different lengths: Such an array is known as a ragged array

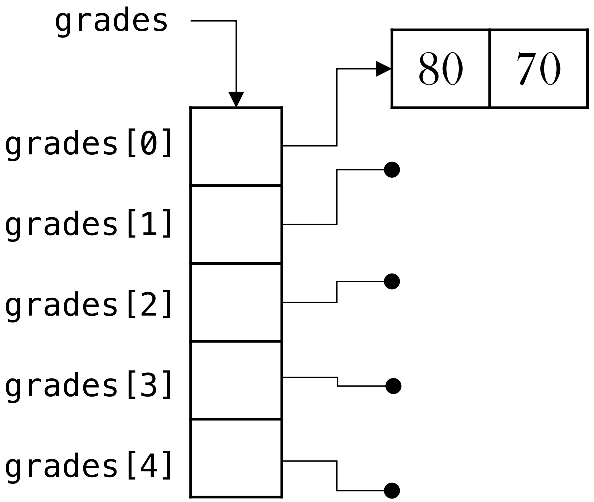
```
// create ragged array. Do not specify column size
int[][] grades = new int[5][]; // 5 rows, unknown columns

// create first row
// first create the row array using new operator
grades[0] = new int[2]; // first student has two grades
```



Ragged Arrays

- Each row in a two-dimensional array is itself an array
- Rows can have different lengths: Such an array is known as a ragged array

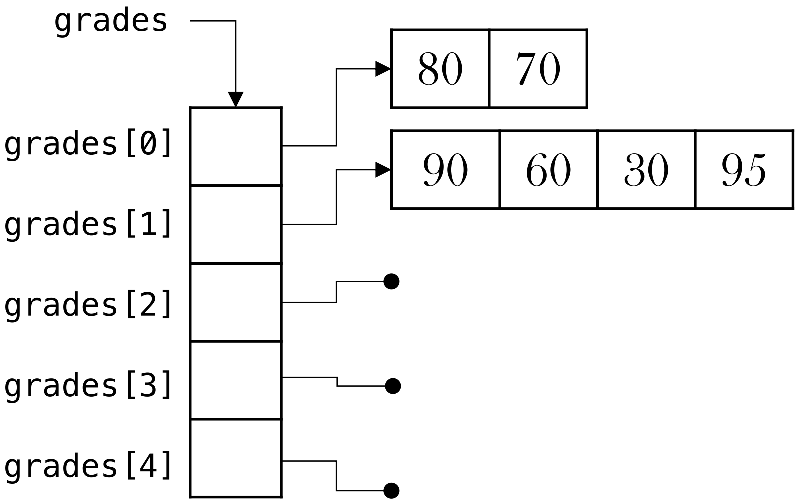
```
// create ragged array. Do not specify column size
int[][] grades = new int[5][]; // 5 rows, unknown columns

// create first row
// first create the row array using new operator
grades[0] = new int[2]; // first student has two grades
grades[0][0] = 80;      // enter grade 80
grades[0][1] = 70;      // enter grade 70
```



Ragged Arrays

- Each row in a two-dimensional array is itself an array
- Rows can have different lengths: Such an array is known as a ragged array

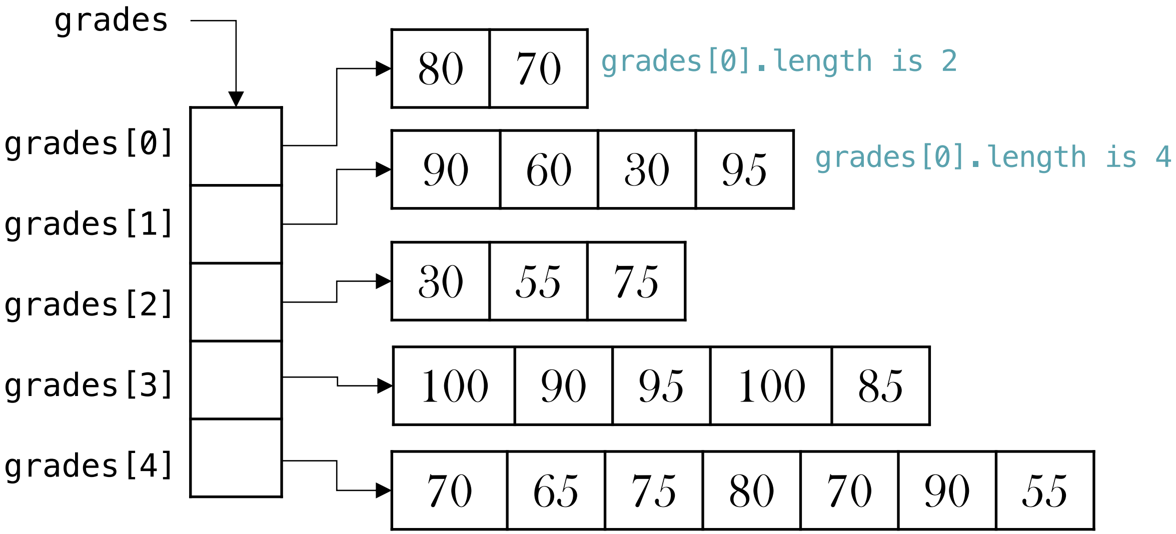
```
// create ragged array. Do not specify column size
int[][] grades = new int[5][]; // 5 rows, unknown columns

// create first row
// first create the row array using new operator
grades[0] = new int[2]; // first student has two grades
grades[0][0] = 80;      // enter grade 80
grades[0][1] = 70;      // enter grade 70

// create second row
// create the row using new operator
grades[1] = new int[4]; // second student has four grades
grades[1][0] = 90;
grades[1][1] = 60;
grades[1][2] = 30;
grades[1][3] = 95;
```



Ragged Arrays

- Each row in a two-dimensional array is itself an array
- Rows can have different lengths: Such an array is known as a ragged array

```
// create ragged array. Do not specify column size
int[][] grades = new int[5][]; // 5 rows, unknown columns

// create first row
// first create the row array using new operator
grades[0] = new int[2]; // first student has two grades
grades[0][0] = 80;      // enter grade 80
grades[0][1] = 70;      // enter grade 70

// create second row
// create the row using new operator
grades[1] = new int[4]; // second student has four grades
grades[1][0] = 90;
grades[1][1] = 60;
grades[1][2] = 30;
grades[1][3] = 95;
```



Ragged Arrays

- When creating ragged arrays, only specify row size. Leave column size empty

```
// create ragged array. Do not specify column size
int[][] grades = new int[5][]; // 5 rows, unknown columns
```

- For each row, first create the row array using the new operator

```
// Create first row: First create the row array using the new operator
grades[0] = new int[2]; // first student has two grades
```

- Array initializer for ragged arrays

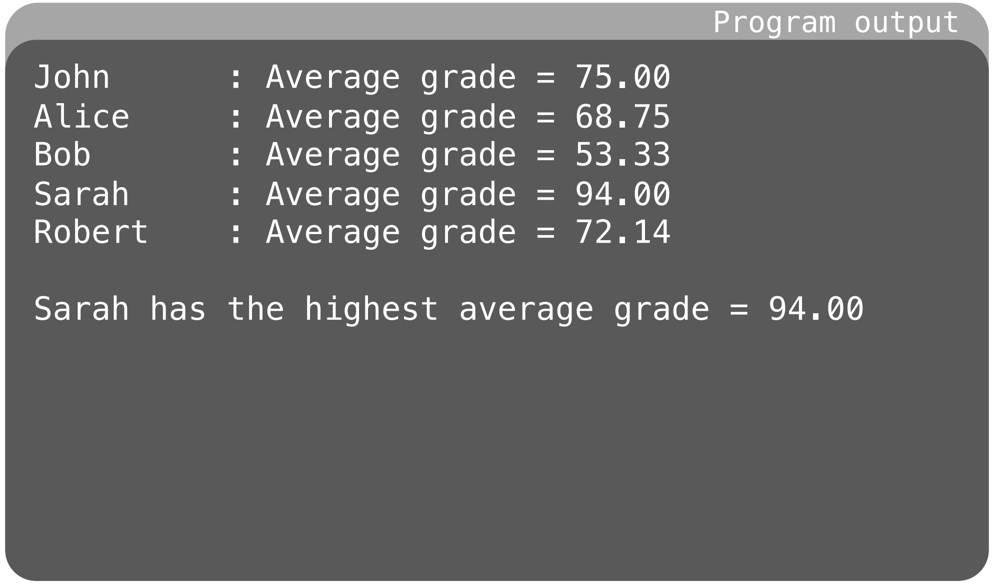
```
int[][] grades = {
 {80,70},
 {90,60,30,95},
 {30,55,75},
 {100,90,95,100,85},
 {70,65,75,80,70,90,55},
};
```

Multi-dimensional Array Example

Student with Highest Average Grade

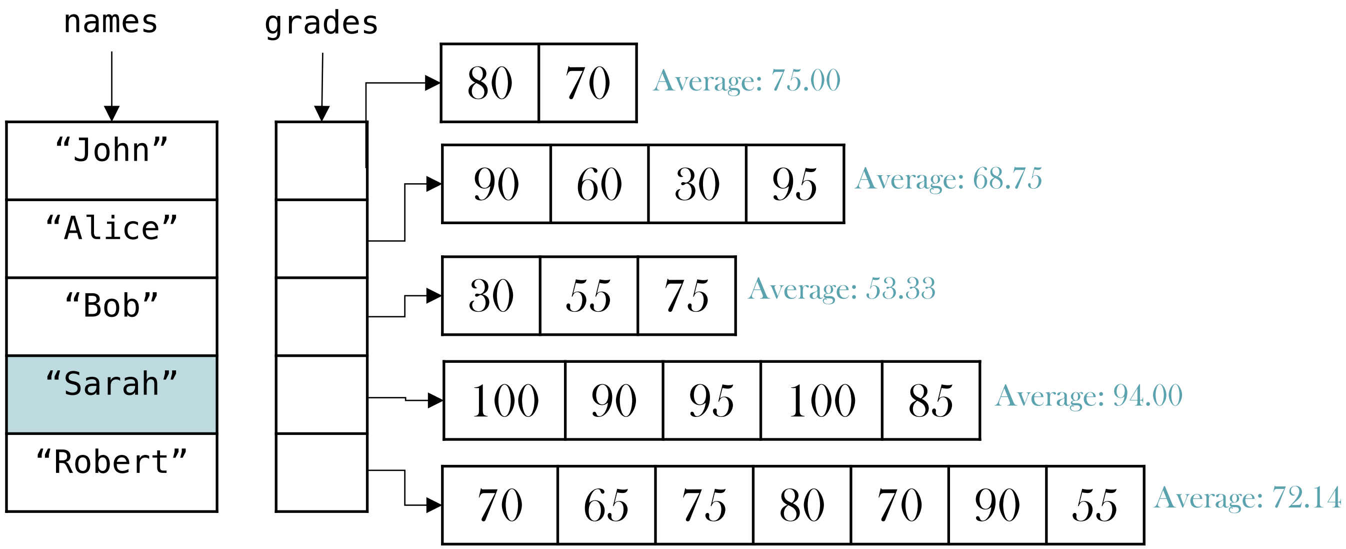
2D Array Example

- Find and print name and average grade of a student with highest aveage grade



Program output

```
John       : Average grade = 75.00
Alice      : Average grade = 68.75
Bob        : Average grade = 53.33
Sarah      : Average grade = 94.00
Robert     : Average grade = 72.14

Sarah has the highest average grade = 94.00
```



80 70 Average: 75.00

90 60 30 95 Average: 68.75

30 55 75 Average: 53.33

100 90 95 100 85 Average: 94.00

70 65 75 80 70 90 55 Average: 72.14

Solution Algorithm

- Initialize maxAverage variable to a very small negative number
  - maxAverage stores the highest average score
- Initialize maxInd variable to zero
  - maxInd stores the array index of the student with the highest average grade
- For each student, calculate his/her average grade
- If the current student's average grade is higher than maxAverage, update maxAverage and maxInd
  - maxAverage should be updated with current student's average grade
  - maxInd should be the array index location of the current student

Source Code - Part 1

```
public class AppGrades {
 public static void main(String[] args) {

   // grades of students
   int[][] grades = {
      {80,70},
      {90,60,30,95},
      {30,55,75},
      {100,90,95,100,85},
      {70,65,75,80,70,90,55},
   };
   String[] names = {"John", "Alice", "Bob", "Sarah", "Robert"}; // names of students

   double average = 0.0;
   int sum;

   double maxAverage = -1 * Double.MAX_VALUE; // set maxAverage to a smallest negative number
   int maxInd = 0; // maxInd stores the array location of the student with the highest average grade

   // code continues
}
```

Source Code - Part 2

```java
public class AppGrades {
  public static void main(String[] args) {

    // code continues from here

    // for each student calculate average grade
    for (int i = 0; i < grades.length; i++) {

      sum = 0; // sum each students grades
      for (int j = 0; j < grades[i].length; j++)
        sum = sum + grades[i][j];

      // calculate average grade of a student
      average = (double)sum / grades[i].length;
      System.out.printf("%-10s: Average grade = %5.2f\n", names[i], average);

      // if current student’s average is greater than maxAverage, update maxAverage and
      // store the array location of the student with the highest average
      if (average > maxAverage) {
        maxAverage = average;
        maxInd = i;
      }
    }
    System.out.printf("\n%s has the highest average grade = %5.2f\n", names[maxInd], maxAverage);
  }
}
```

Multi-dimensional Array Example

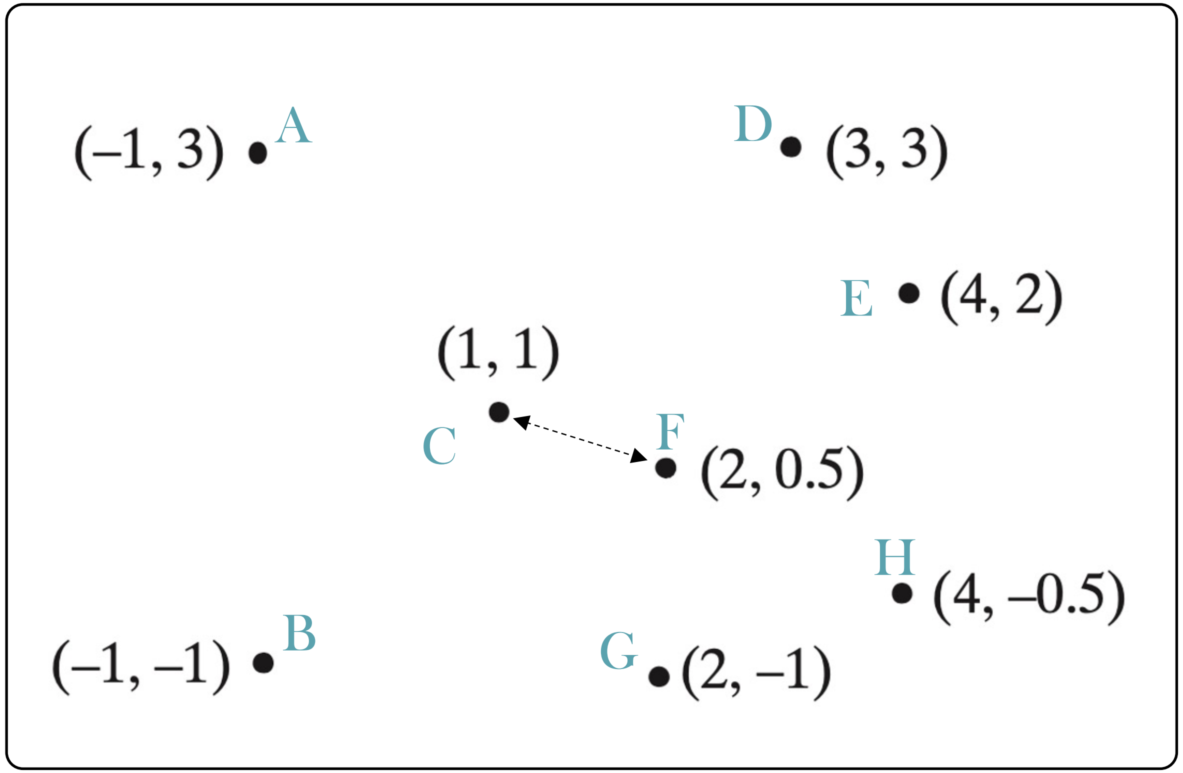
Closest Cities in a Map

2D Array Example: Closest Cities

- Given (x,y) coordinates of cities, find the city pair where their distance is smallest
- Print closest city names and the distance between them

Program output

```
Closest cities are: C and F
Distance = 1.12km
```

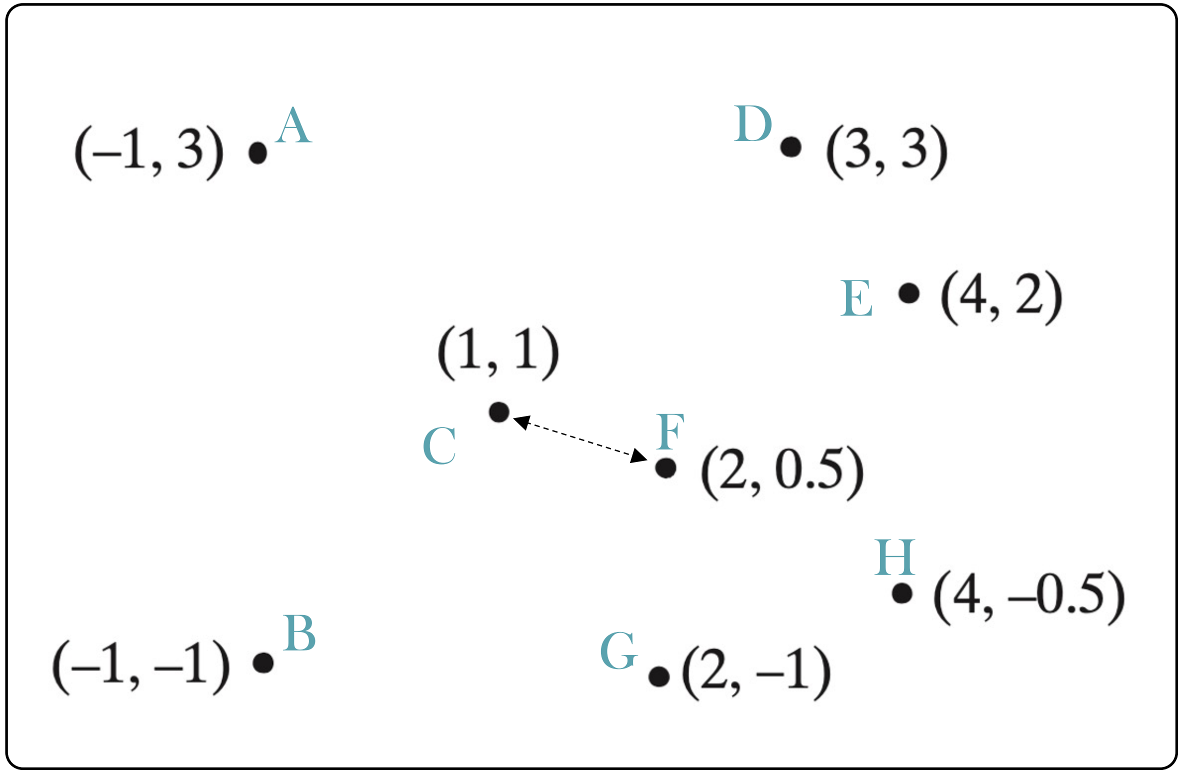


2D Array Example: Closest Cities

- Store (x,y) coordinates of cities in a 2D array

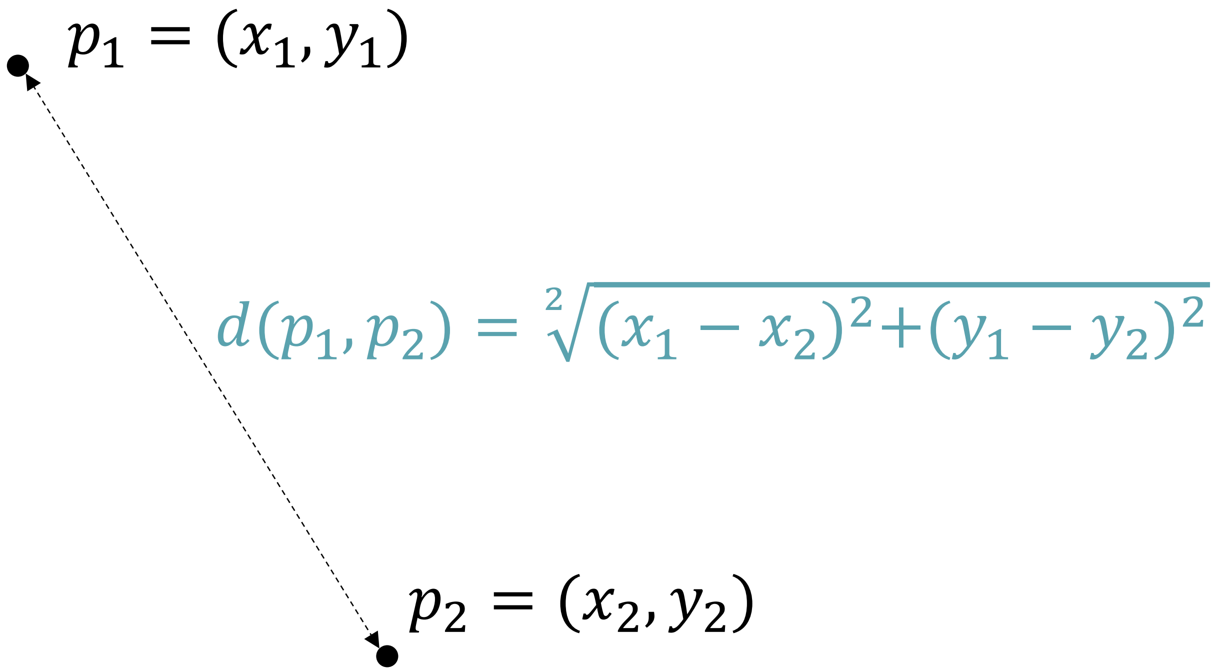
cityNames
”A”
“B”
“C”
“D”
“E”
“F”
“G”
“H”

coordinates	
x	y
-1	3
-1	-1
1	1
3	3
4	2
2	0.5
2	-1
4	-0.5



2D Array Example: Closest Cities

- Euclidean distance between two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is defined as:



$p_1 = (x_1, y_1)$

$$d(p_1, p_2) = \sqrt[2]{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$p_2 = (x_2, y_2)$

Solution Algorithm

- Define minDist initially with a very large value
- For each city pair c1 and c2, compute distance
    - If the computed distance is less than the minDist
        - Update minDist
        - Store these city indices in index variables

Source Code - Part 1

```
public class AppClosestCities {
 public static void main(String[] args) {
   // city names
   String[] cityNames = {"A","B","C","D","E","F","G","H"};

    // city coordinates
   double[][] coordinates = {
      {-1.0,3.0},
      {-1.0,-1.0},
      {1.0,1.0},
      {3.0,3.0},
      {4.0,2.0},
      {2.0,0.5},
      {2.0,-1.0},
      {4.0,-0.5}
   };

    // store minimum distance and indexes of the closest cities
   double minDistance = Double.MAX_VALUE;
   int cityInd1 = 0;
   int cityInd2 = 0;

   // code continues
}
```

Source Code – Part 2

```
public class AppClosestCities {
 public static void main(String[] args) {

  // code continues here
  for (int i = 0; i < coordinates.length-1; i++) {
   for (int j = i + 1; j < coordinates.length; j++) {
    // get (x,y) coordinates of city1 and city2
    double x1 = coordinates[i][0];
    double y1 = coordinates[i][1];
    double x2 = coordinates[j][0];
    double y2 = coordinates[j][1];

    // compute Euclidean distance between city1 and city2
    double distance = Math.pow(Math.pow(x1-x2, 2) + Math.pow(y1-y2, 2), 0.5);

    // update min distance and closest city indexes if necessary
    if (distance < minDistance) {
     minDistance = distance;
     cityInd1 = i;
     cityInd2 = j;
    }
   }
  }
  System.out.printf("Closest cities are: %s and %s. Distance = %5.2f\n", cityNames[cityInd1], cityNames[cityInd2], minDistance);
 }
}
```

java.util.Arrays Class

Useful Array Methods

Arrays Class

- The `java.util.Arrays` class contains various methods for sorting and searching arrays, comparing arrays, filling array elements, and returning a string representation of the array
- Import `java.util.Arrays` to use Arrays class methods

Sort	`Arrays.sort(numbers);`
Search	`int loc = Arrays.binarySearch(names, "John")`
Compare	`boolean result = Arrays.equals(array1, array2)`
Fill	`Arrays.fill(list, 88)`
Conversion to String	`String str = Arrays.toString(numbers)`

Arrays Class: Examples

```java
// Sort an array
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
Arrays.sort(numbers); // Sort the whole array in increasing order

// Printing an array
String str = Arrays.toString(numbers);
System.out.println(str);

// Searching an element in an array
String[] names = {"Alice", "Bob", "John", "Robert"};
int location = Arrays.binarySearch(names, "John");
System.out.println("Location : " + location);

// Check if arrays are equal
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
int[] list3 = {4, 2, 7, 10};
System.out.println(java.util.Arrays.equals(list1, list2)); // prints true
System.out.println(java.util.Arrays.equals(list2, list3)); // prints false

// Fill arrays with a value
int[] list = new int[5];
java.util.Arrays.fill(list, 88); // Fill 88 to the whole array
System.out.println(Arrays.toString(list));
```