

Algorithms

Author: Boaz Kantor

The Interdisciplinary Center, Herzliya, Introduction to Computer Science, Winter 2009-10 Semester

Algorithms to Discuss

- ▶ Assume a collection of elements
- ▶ Search
 - ▶ Find an element in the collection
 - ▶ Search criteria:
 - ▶ Absolute (name, description, value, etc)
 - ▶ Relative (minimum, maximum, median, average)
- ▶ Sort
 - ▶ Reposition elements to be ordered
 - ▶ Sort criteria:
 - ▶ Anything that can be equalized with $<$ or $>$

Search

▶ Questions to ask:

1. Is the collection sorted?
2. Can there be more than one matching element?

▶ If so, does it matter which one is returned?

```
String passwoid = "ken sent me";  
System.out.println(passwoid.lastIndexOf('e')); // matters!  
findMinimum(3, 6, 2, 9, 2, 8); // doesn't matter as long as it's 2
```

3. Do we have to iterate through the entire collection?

▶ No, if we can stop when we found it

```
passwoid.indexOf('e');
```

▶ Yes, if there can be a better match (e.g.: find minimum)

Find An Element

▶ Assume collection is:

```
int[] collection;
```

1. Is the collection sorted?

- ▶ If so then we have advanced algorithms
- ▶ If not – see algorithm in next slide

2. Can there be more than one matching element?

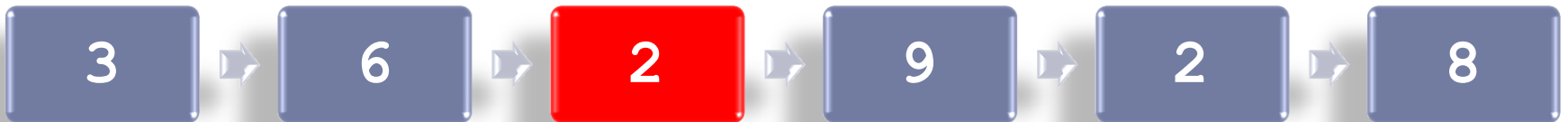
- ▶ Yes. Decide on algorithm behavior in such case:
 - ▶ **Return first**
 - ▶ Return last
 - ▶ Return “next” (like Scanner.next)

3. Do we have to iterate through the entire collection?

- ▶ No! Once found, the algorithm stops.

Find An Element – the Algorithm

```
public static boolean findElement(int[] collection, int value) {  
    for (int currElement = 0; currElement < collection.length; ++currElement) {  
        if (collection[currElement] == value) {  
            return true;  
        }  
    }  
    return false;  
}
```



Find Minimum (Maximum)

▶ Assume collection is:

```
int[] collection;
```

1. Is the collection sorted?

▶ If so then `return collection[0];`

▶ If not – see algorithm

2. Can there be more than one matching element?

▶ Yes, but they all have the same value so it doesn't matter

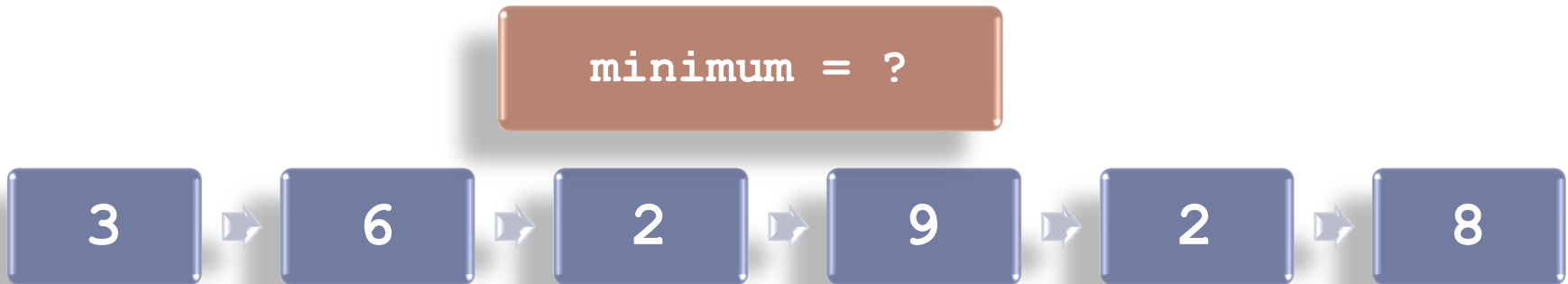
```
{3, 6, 2, 9, 2, 8}
```

3. Do we have to iterate through the entire collection?

▶ Yes! **there** **might** be a **smaller** number **down** **the** road

Find Minimum – the Algorithm

```
public static int findMinimum(int[] collection) {  
    // don't forget to validate parameters!  
    int minimum = collection[0];  
    for (int currElement = 1; currElement < collection.length; currElement++) {  
        if (collection[currElement] < minimum) {  
            minimum = collection[currElement];  
        }  
    }  
    return minimum;  
}
```

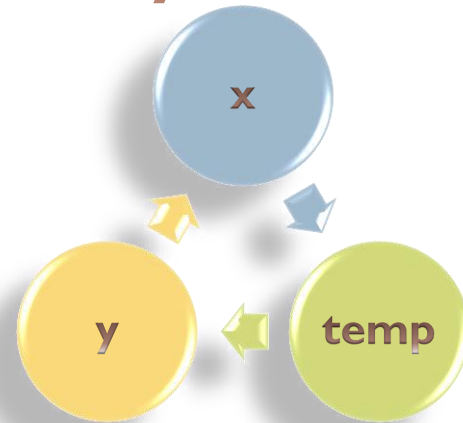


- ▶ Finding maximum is same same (but different – how?)

Switching

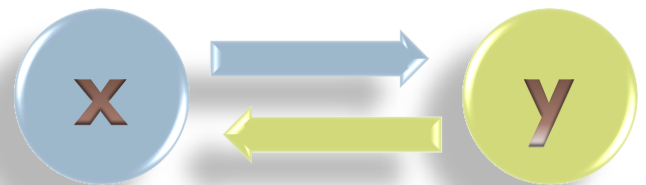
- ▶ Switch among two elements
- ▶ If **x** is **5** and **y** is **12** then **x** will be **12** and **y** will be **5**
- ▶ A fundamental algorithm

```
public static void switchElements(int x, int y) {  
    int temp = x;  
    x = y;  
    y = temp;  
}
```



- ▶ If you wish to save space (works only with numbers):

```
public static void switchElements(int x, int y) {  
    x += y;  
    y = x-y;  
    x -= y;  
}
```



Sort

- ▶ Decide on an order of a collection
- ▶ Output: a sorted collection
- ▶ Examples:
 - ▶ Numbers:
 - ▶ $x > y$ iff $x > y$
 - ▶ By age:
 - ▶ $x > y$ iff $x.getAge() > y.getAge()$
 - ▶ Lexicographic:
 - ▶ $x > y$ iff $(\text{unicode})x[i] > (\text{unicode})y[i]$, for smallest i s.t. $x[i] \neq y[i]$
 - ▶ zebra > mouse > mosquito > elephant > dinosaur

Sorting Algorithms

▶ Selection sort:

- ▶ Loop $i=0$ to size of array - 2:
 - ▶ Assume sub-collection of element $[i]$ to element $[\text{array size} - 1]$
 - ▶ Find minimum element in sub-collection
 - ▶ Switch with first element of sub-collection

▶ Insertion sort:

- ▶ Loop $i=1$ to size of array - 1:
 - ▶ Assume sub-collection of element $[0]$ to element $[i]$
 - ▶ Find the appropriate location j for element $[i]$
 - ▶ Push element $[i]$ to location

▶ Bubble sort:

- ▶ Loop $i=1$ to size of array - 1:
 - ▶ Loop $j = 0$ to size of array - i
 - ▶ Compare first two elements and switch if needed
 - ▶ Compare with next element and switch if needed

▶ Merge sort:

- ▶ Split the array in two halves and sort each separately
- ▶ Compare two left-most elements of both parts and copy them sorted

Complexity

- ▶ Considering input of size N , how many instructions will it take to run an algorithm in:
 - ▶ Best case scenario
 - ▶ Average case scenario
 - ▶ **Worst case scenario**
- ▶ Count only the instructions which are dependent on N
- ▶ Ignore relatively low-complex parts:
 - ▶ Order of $N + (5 \times N^2)/2$ is **N^2**

Fin

Algorithms

Author: Boaz Kantor

The Interdisciplinary Center, Herzliya, Introduction to Computer Science, Winter 2009-10 Semester