



# Arrays

Forest Agostinelli

University of South Carolina

# Outline

- Creating Arrays
- Algorithms for finding values
- Algorithms for sorting

# Creating Arrays

- Arrays are a collection of variables of the same type
- Foundational Data Structure
- Contiguous Block of Memory
  - The size of the Array must be specified initially
  - Arrays cannot be resized
- In Java, Arrays are considered a special kind of Object
  - Container Object
  - Identifiers contain only the reference to its contents
  - The reference *points* to contents
  - “==” Does not check the contents of the array

## Creating an Array Syntax

```
//Declaring an Array
<<type>>[] <<id>>;
//Initializing an Array]
<<id>> = new <<type>>[<<size>>];
//or
<<type>>[] <<id>> = new <<type>>[size];
```

## Example

```
//Creates an array of 5 integers
int[] array = new int[5];
```

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];
```

## Memory

| Identifier | Contents | Byte Address |
|------------|----------|--------------|
| ...        | ...      | ...          |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
| ...        | ...      | ...          |



# Creating Arrays

```
//Creates an array of 5 integers
```


```
→ int[] array = new int[5];
```

## Memory

| Identifier | Contents | Byte Address |
|------------|----------|--------------|
| ...        | ...      | ...          |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
| ...        | ...      | ...          |

# Creating Arrays

//Creates an array of 5 integers


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

## Memory

| Identifier | Contents | Byte Address |
|------------|----------|--------------|
| ...        | ...      | ...          |
| array      | NULL     | 28           |
| ...        | ...      | ...          |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
|            |          |              |
| ...        | ...      | ...          |

# Creating Arrays

//Creates an array of 5 integers


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

## Memory

| Identifier | Contents | Byte Address |
|------------|----------|--------------|
| ...        | ...      | ...          |
| array      | NULL     | 28           |
| ...        | ...      | ...          |
| array[0]   | 0        | 60           |
| array[1]   | 0        | 64           |
| array[2]   | 0        | 68           |
| array[3]   | 0        | 72           |
| array[4]   | 0        | 76           |
| ...        | ...      | ...          |

# Creating Arrays

//Creates an array of 5 integers


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

## Memory

| Identifier | Contents | Byte Address |
|------------|----------|--------------|
| ...        | ...      | ...          |
| array      | 60       | 28           |
| ...        | ...      | ...          |
| array[0]   | 0        | 60           |
| array[1]   | 0        | 64           |
| array[2]   | 0        | 68           |
| array[3]   | 0        | 72           |
| array[4]   | 0        | 76           |
| ...        | ...      | ...          |

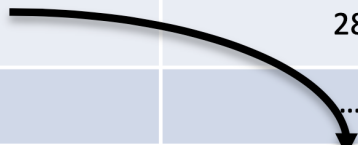
# Creating Arrays

//Creates an array of 5 integers

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

## Memory

| Identifier | Contents | Byte Address |
|------------|----------|--------------|
| ...        | ...      | ...          |
| array      | 60       | 28           |
| ...        | ...      | ...          |
| array[0]   | 0        | 60           |
| array[1]   | 0        | 64           |
| array[2]   | 0        | 68           |
| array[3]   | 0        | 72           |
| array[4]   | 0        | 76           |
| ...        | ...      | ...          |



# Creating Arrays

- Arrays have *Indices*
  - An “Index” corresponds to the individual values in the array
  - Indices start at 0
  - Indices End at Size-1 (or Length-1)
- “Indexing” is how we access and modify elements of an array.
  - Using an index that is less than 0 or greater than the Size-1 will cause a run-time error
- Random Access
  - Allows access and modification of data at any point in the array instantly
  - Address = Start\_Address+Type\_Size\*index
  - Best advantage of using an array

## Indexing Syntax

```
//Accessing Data
<<id>>[<<index>>];
//Modifying Data
<<id>>[<<index>>] = <<value>>;
```

## Example

```
//Assigns the first and 5th elements
array[0] = 1;
array[4] = 5;
//Adds the first and 5th elements together
int firstPlusLast = array[0] + array[4];
```

# Creating Arrays

- The size of an array can be access through the property “.length”;
- For-Loops are the arrays “best friend”
  - Counting variable can be used for indexing
  - Using the property “.length” can be used in the Boolean expression

## Length Syntax

```
//Length of an Array  
<<id>>.length;
```

## Example

```
//Assigns the first and last elements  
array[0] = 1;  
array[array.length-1] = 5;  
//Adds the first and last elements together  
int firstPlusLast = array[0] + array[array.length-1];
```

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
→ for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 0        | 64           |
| array[2]     | 0        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| ...          | ...      | ...          |



# Creating Arrays

```
//Creates an array of 5 integers
int[] array = new int[5];
for(int i=0;i<array.length;i++)
{
    array[i] = i;
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 0        | 64           |
| array[2]     | 0        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 0        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0; i<array.length; i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 0        | 64           |
| array[2]     | 0        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 0        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

→ `array[i] = i;`

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 0        | 64           |
| array[2]     | 0        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 0        | 128          |

$$\text{Address} = 60 + 4 * i$$

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 0        | 64           |
| array[2]     | 0        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 1        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0; i<array.length; i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 0        | 64           |
| array[2]     | 0        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 1        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 0        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 1        | 128          |

$$\text{Address} = 60 + 4 * i$$

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
→ for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 0        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 2        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0; i<array.length; i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 0        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 2        | 128          |



# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 2        | 128          |

$$\text{Address} = 60 + 4 * i$$

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 3        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0; i<array.length; i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 0        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 3        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 3        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 3        | 128          |

$$\text{Address} = 60 + 4 * i$$

# Creating Arrays

```
//Creates an array of 5 integers
int[] array = new int[5];
for(int i=0;i<array.length;i++)
{
    array[i] = i;
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 3        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 4        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0; i<array.length; i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 3        | 72           |
| array[4]     | 0        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 4        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 3        | 72           |
| array[4]     | 4        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 4        | 128          |


$$\text{Address} = 60 + 4 * i$$

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 3        | 72           |
| array[4]     | 4        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 5        | 128          |





# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
→ for(int i=0; i<array.length; i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 3        | 72           |
| array[4]     | 4        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 5        | 128          |

# Creating Arrays

```
//Creates an array of 5 integers  
int[] array = new int[5];  
for(int i=0;i<array.length;i++)  
{  
    array[i] = i;  
}
```

## Memory

| Identifier   | Contents | Byte Address |
|--------------|----------|--------------|
| ...          | ...      | ...          |
| array        | 60       | 28           |
| ...          | ...      | ...          |
| array[0]     | 0        | 60           |
| array[1]     | 1        | 64           |
| array[2]     | 2        | 68           |
| array[3]     | 3        | 72           |
| array[4]     | 4        | 76           |
| ...          | ...      | ...          |
| array.length | 5        | 84           |
| ...          | ...      | ...          |
| i            | 5        | 128          |

# Example

```
/*
 * Written by JJ Shepherd
 */
import java.util.Scanner;
public class ClosestValue {

    public static final int SIZE = 5;
    public static final double PRICE = 5.97;
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        double[] prices = new double[SIZE];
        System.out.println("Welcome to the Closest without going over game!\nGuess the price of
Shaq-Fu");
        for(int i=0;i<prices.length;i++)
        {
            System.out.println("Contestant "+i+" enter a non-negative price!");
            double newPrice = keyboard.nextDouble();
            if(newPrice >= 0.0)
            {
                prices[i] = newPrice;
            }
            else
            {
                System.out.println("That price is invalid! Assigning it to $0.00");
                prices[i] = 0.0;
            }
        }
        System.out.println("The actual price is "+PRICE);

        double closestPrice = -1.00;
        int winnerIndex = -1;

        for(int i=0;i<prices.length;i++)
        {
            if(prices[i] <= PRICE && prices[i] > closestPrice)
            {
                closestPrice = prices[i];
                winnerIndex = i;
            }
        }
        if(winnerIndex == -1)
        {
            System.out.println("No one wins.");
        }
        else
        {
            System.out.println("The winner is contestant "+winnerIndex+" with a guess of
"+closestPrice);
        }
    }
}
```

# Creating Arrays

- If the values are known, it is possible to both construct the array and initialize the values at the same time.
- Values are put inside of curly braces (“{}”)
- Each value is separated by a comma (“,”)

## Creating an Array Syntax

```
//Declaring an Array and Initializing its Values  
<<type>>[] <<id>> = {<<Value0>>,<<Value1>>,...};
```

## Example

```
//Creates an array of 5 integers  
int[] array = {0,1,2,3,4};
```

# Outline

- Creating Arrays
- Algorithms for finding values
- Algorithms for sorting

# Algorithm for Finding a Target Value

- Search and Sorting Arrays are fundamental to their function
- Searching involves *looking* through an array for some “target” value
  - Target values can be specific values
  - They can also be values with special properties like the minimum or maximum

## Searching for a Target Value

```
int[] a = {0,1,2,3,4};
boolean found = false;
int target = keyboard.nextInt();
for(int i=0;i<a.length;i++)
{
    if(a[i] == target)
    {
        found = true;
    }
}
```

# Algorithm for Finding the Max Value

- Search and Sorting Arrays are fundamental to their function
- Searching involves *looking* through an array for some “target” value
  - Target values can be specific values
  - They can also be values with special properties like the minimum or maximum

## Searching for the MAX Value

```
int[] a = {0,1,2,3,4};
int max = a[0]; //Should not be arbitrary
for(int i=1;i<a.length;i++)
{
    if(a[i] > max)
    {
        max = a[i];
    }
}
```

# Outline

- Creating Arrays
- Algorithms for finding values
- Algorithms for sorting



# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

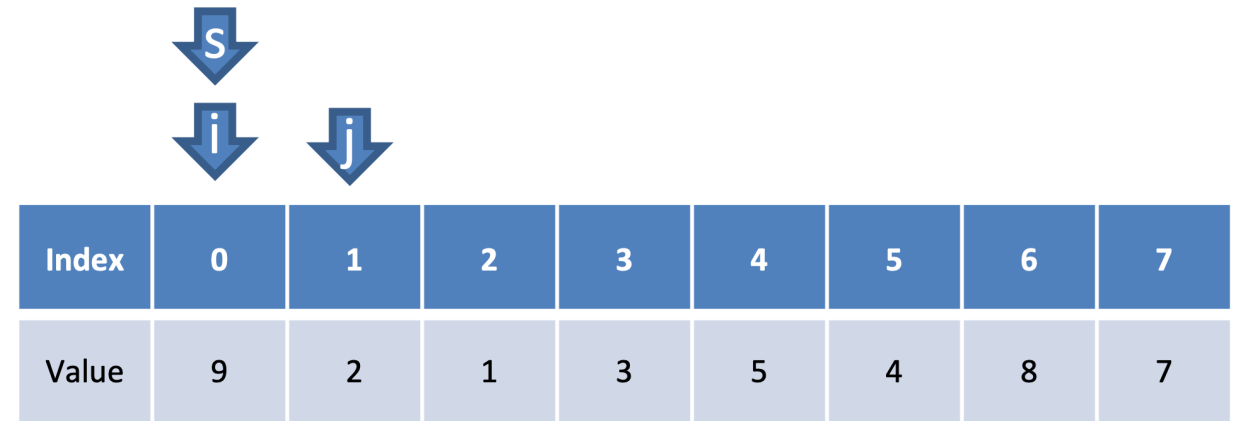
## Selection Sort Example

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm. It shows a table with two rows: 'Index' and 'Value'. The 'Index' row contains indices from 0 to 7. The 'Value' row contains the values 9, 2, 1, 3, 5, 4, 8, and 7. Above the table, three blue arrows point downwards to the first three columns. The top arrow is labeled 's' and points to index 0. The middle arrow is labeled 'i' and points to index 1. The bottom arrow is labeled 'j' and points to index 2. This indicates that the algorithm is currently comparing the value at index 0 (9) with the value at index 1 (2) and index 2 (1).

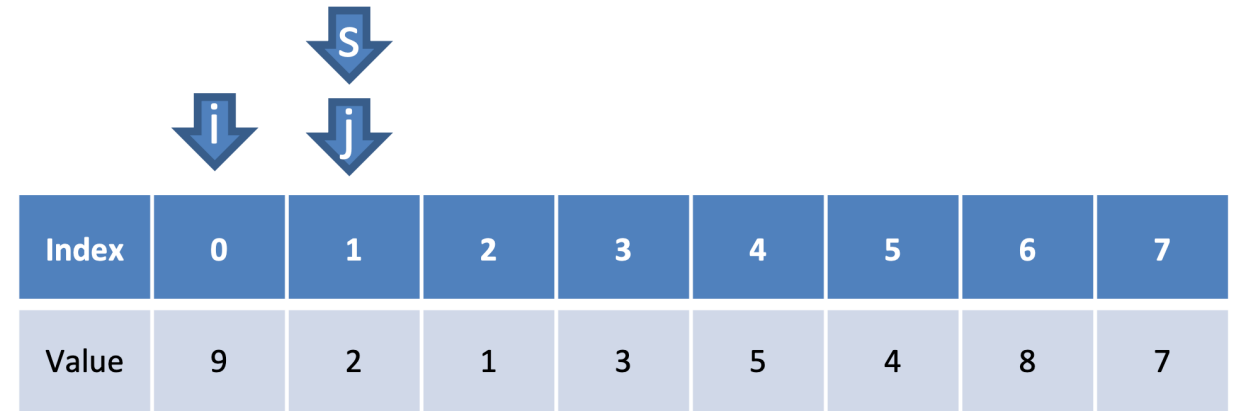
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm. It shows an array of values: [9, 2, 1, 3, 5, 4, 8, 7]. The current index being compared is 0 (value 9). A smaller value is found at index 2 (value 1). The smallest value in the current subarray is at index 1 (value 2). Arrows labeled 'i', 'j', and 's' point to indices 0, 2, and 1 respectively. A legend below the table defines 's' as the index with the smallest value.

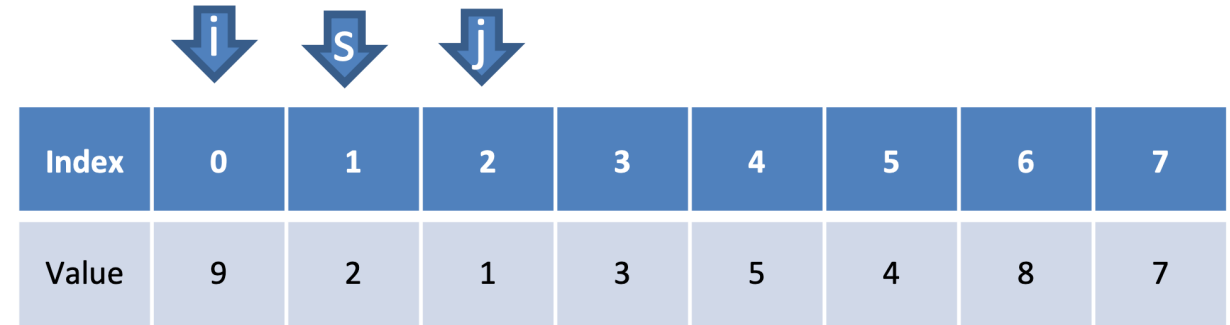
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm. It shows an array of values: [9, 2, 1, 3, 5, 4, 8, 7]. Above the array, three blue arrows point down to indices 0, 1, and 2, labeled 'i', 's', and 'j' respectively. This indicates that the algorithm is currently comparing the value at index 0 (9) with the value at index 1 (2) to find the smallest element. The value at index 2 (1) is also shown, but the arrow 'j' is positioned above it, suggesting it might be the current candidate for the smallest value.

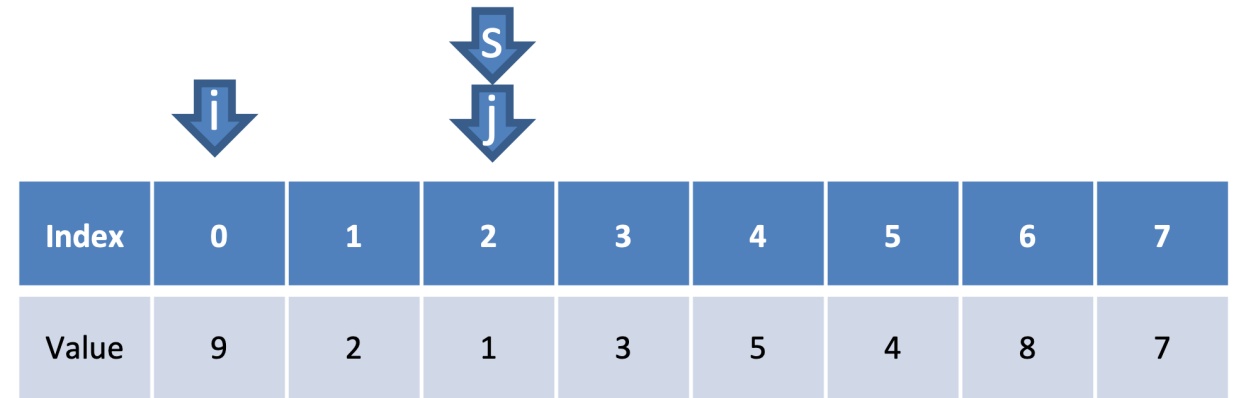
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm on an array. A table shows the current state of the array with indices 0 through 7 and values 9, 2, 1, 3, 5, 4, 8, 7. Above the table, a blue arrow labeled 'i' points to index 0. Another blue arrow labeled 's' points to index 2, and a third blue arrow labeled 'j' points to index 2, indicating that 's' and 'j' are both pointing to the current smallest element found.

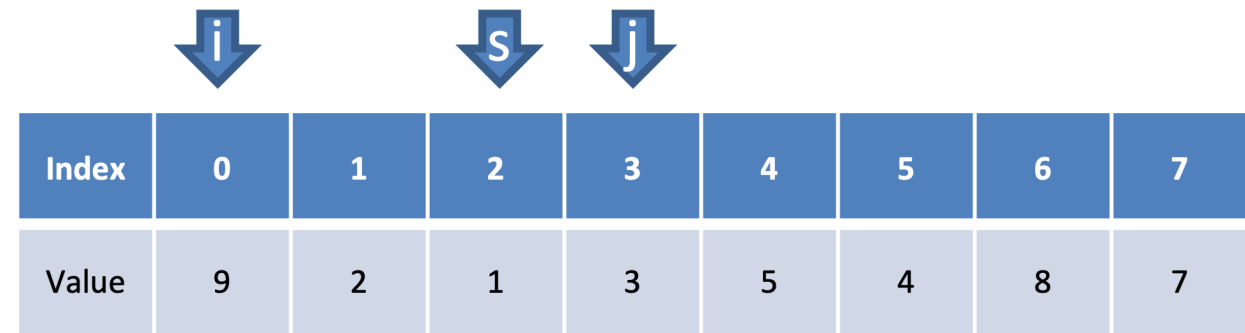
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm. It shows an array of values: 9, 2, 1, 3, 5, 4, 8, 7. The current index being processed is 0 (value 9). The algorithm is searching for the smallest value in the remaining array (indices 1 to 7). The smallest value found is 1 at index 2. The variable 's' is marked at index 2, indicating the swap location. The variable 'j' is marked at index 3, indicating the current element being compared. The variable 'i' is marked at index 0, indicating the current element being compared.

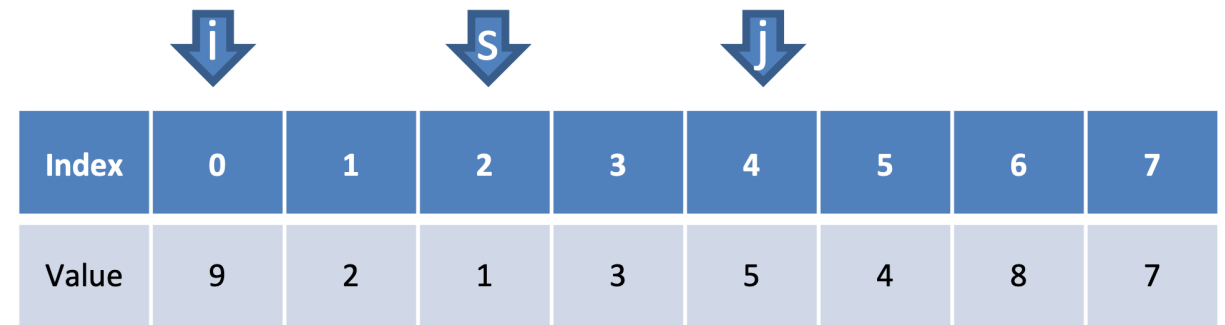
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm. It shows a horizontal array of 9 cells. The top row is labeled 'Index' and contains values 0 through 7. The bottom row is labeled 'Value' and contains values 9, 2, 1, 3, 5, 4, 8, and 7. Above the array, three blue arrows point downwards to specific indices: 'i' points to index 0, 's' points to index 2, and 'j' points to index 4. The value at index 2 is 1, which is the smallest value in the array.

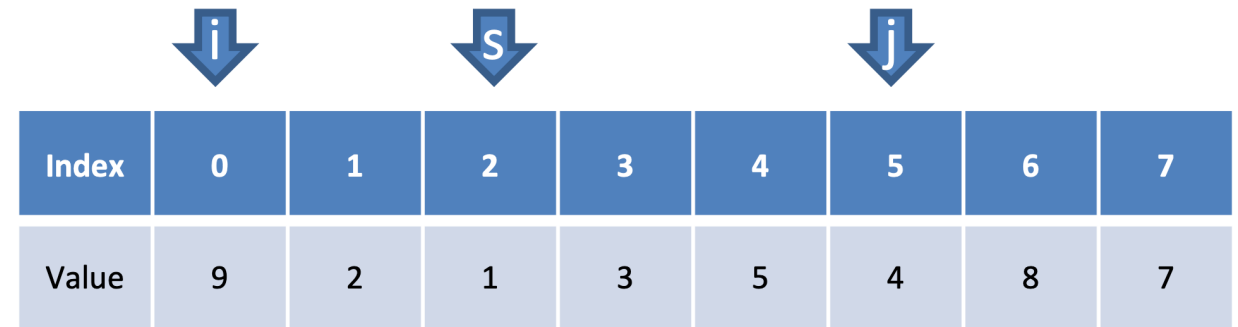
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm. It shows an array of 9 elements with indices 0 to 7. The values are 9, 2, 1, 3, 5, 4, 8, and 7. Three blue arrows point to indices 0, 2, and 5, labeled 'i', 's', and 'j' respectively. The value at index 2 (1) is the smallest in the array.

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

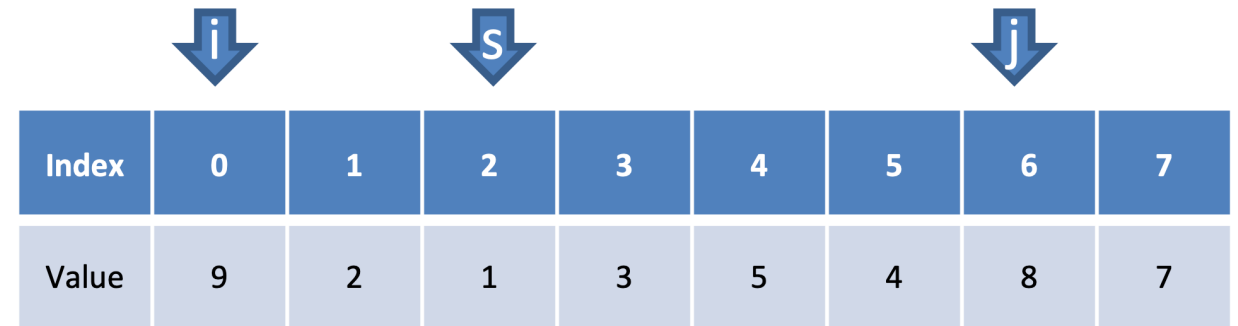
s = Index with  
smallest value



# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm. It shows an array of values: 9, 2, 1, 3, 5, 4, 8, 7. The current index being processed is 0 (marked with 'i'). The smallest value found in the remaining array is 1 at index 2 (marked with 's'). The next index to be processed is 6 (marked with 'j').

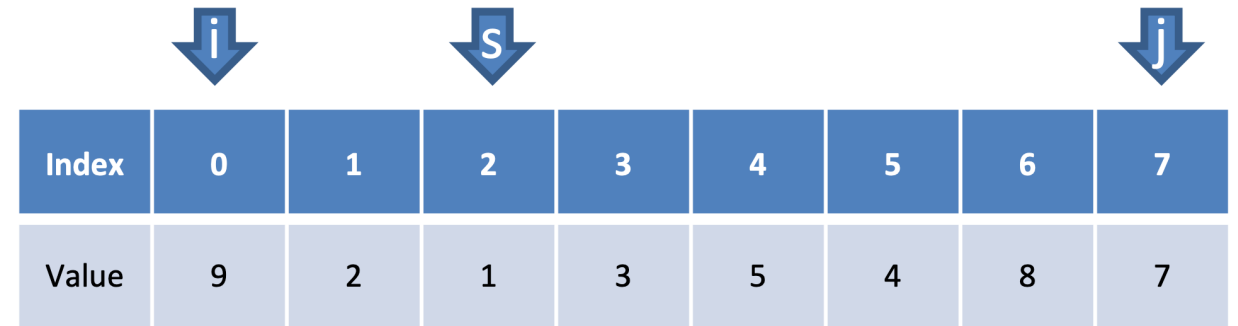
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm on an array of values. The array is represented as a table with two rows: 'Index' and 'Value'. The indices are 0 through 7, and the values are 9, 2, 1, 3, 5, 4, 8, and 7. Three blue arrows point to specific indices: 'i' points to index 0, 's' points to index 2, and 'j' points to index 7. The value at index 2 (1) is the smallest in the array.

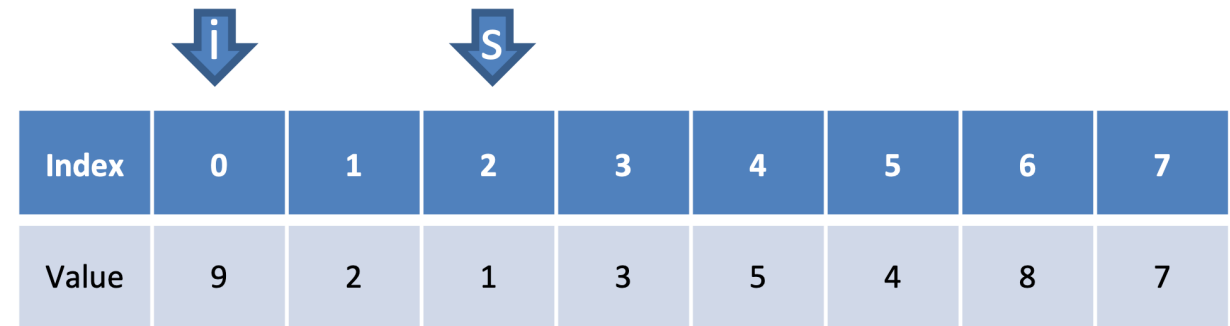
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm. It shows an array of values: 9, 2, 1, 3, 5, 4, 8, 7. The current index being processed is 0, marked with a blue arrow labeled 'i'. The smallest value in the array is 1, located at index 2, marked with a blue arrow labeled 's'.

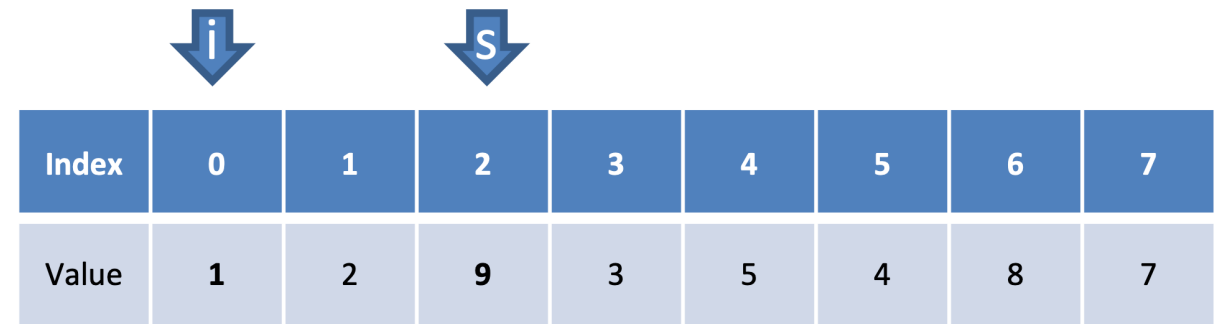
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram illustrates the Selection Sort algorithm. It shows a horizontal array of 9 cells. The top row is labeled 'Index' and contains values 0 through 7. The bottom row is labeled 'Value' and contains values 1, 2, 9, 3, 5, 4, 8, and 7. A blue arrow labeled 'i' points to index 0. Another blue arrow labeled 's' points to index 2. The value 9 at index 2 is the smallest value in the array.

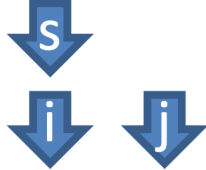
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 1 | 2 | 9 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

- We will assume ascending order (smallest value to largest)
- Selection Sort Algorithm
  1. Start at the first index
  2. Assume this value is in the correct location
  3. Check all other values after this index
  4. If a smaller value is found, then mark that index
  5. If the assumed index does not contain smallest value, then swap that with the marked index
  6. Repeat step 2 for all indices

## Selection Sort Example



The diagram shows three blue arrows pointing downwards. The top arrow is labeled 's' and points to index 0. The middle arrow is labeled 'i' and points to index 1. The bottom arrow is labeled 'j' and points to index 2.

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 1 | 2 | 9 | 3 | 5 | 4 | 8 | 7 |

s = Index with  
smallest value

# Selection Sort

```
/*
 * Written by JJ Shepherd
 */
import java.util.Scanner;
public class SelectionSort {

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Welcome to the Selection Sort Program!\nEnter the size of the
Array.");
        int size = keyboard.nextInt();
        if(size <= 0)
        {
            System.out.println("That is an invalid size.");
            System.exit(0);
        }
        int[] a = new int[size];
        for(int i=0;i<a.length;i++)
        {
            System.out.println("Enter value at index "+i);
            a[i] = keyboard.nextInt();
        }
        //Selection Sort
        for(int i=0;i<a.length;i++)
        {
            int smallestIndex = i;
            for(int j = i+1;j<a.length;j++)
            {
                if(a[j] < a[smallestIndex])
                {
                    smallestIndex = j;
                }
            }
            if(smallestIndex != i)
            {
                //Swap
                int temp = a[i];
                a[i] = a[smallestIndex];
                a[smallestIndex] = temp;
            }
        }
        //Print values
        System.out.println("The sorted array is");
        for(int i=0;i<a.length;i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

# Computational Complexity of Selection Sort

- Worst case in number of comparisons?
- Best case in number of comparisons?

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example


| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |



# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 9 | 2 | 1 | 3 | 5 | 4 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example


| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 9 | 1 | 3 | 5 | 4 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 9 | 1 | 3 | 5 | 4 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example


| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 1 | 9 | 3 | 5 | 4 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example




| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 1 | 9 | 3 | 5 | 4 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 1 | 3 | 9 | 5 | 4 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example

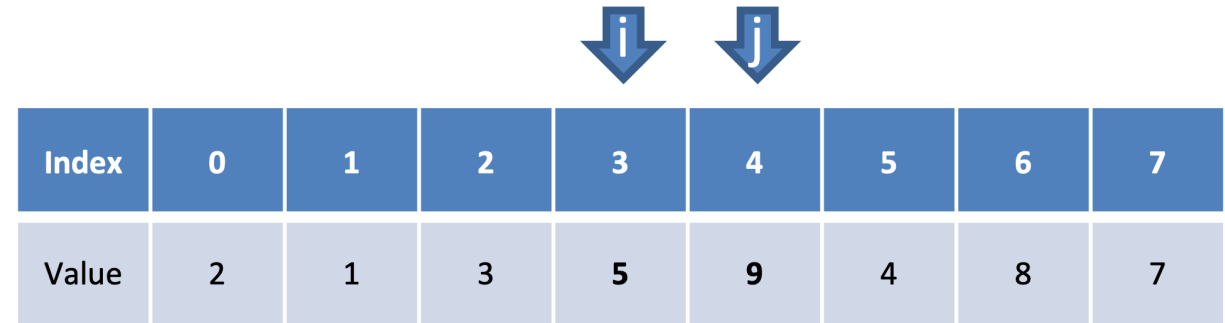
|       |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Value | 2 | 1 | 3 | 9 | 5 | 4 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 1 | 3 | 5 | 9 | 4 | 8 | 7 |

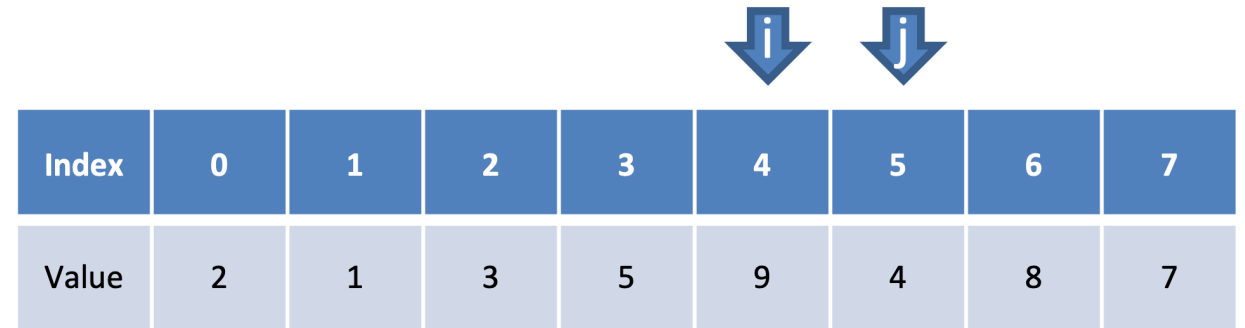
$$j = i + 1$$



# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



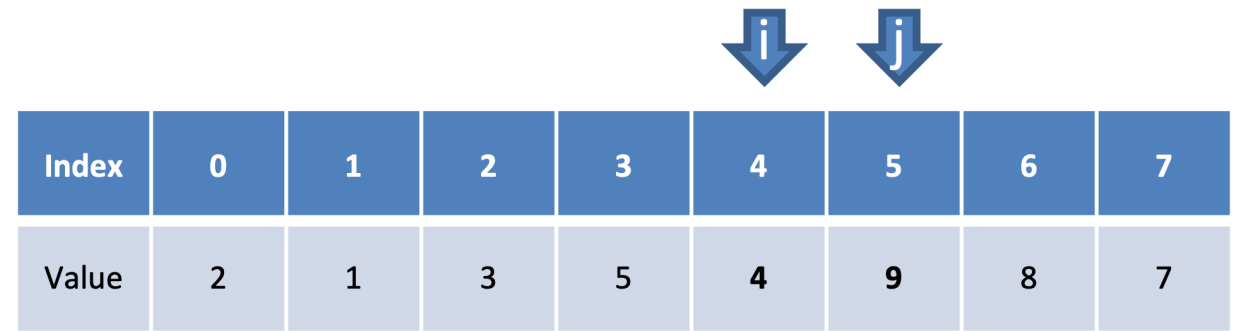
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 1 | 3 | 5 | 9 | 4 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



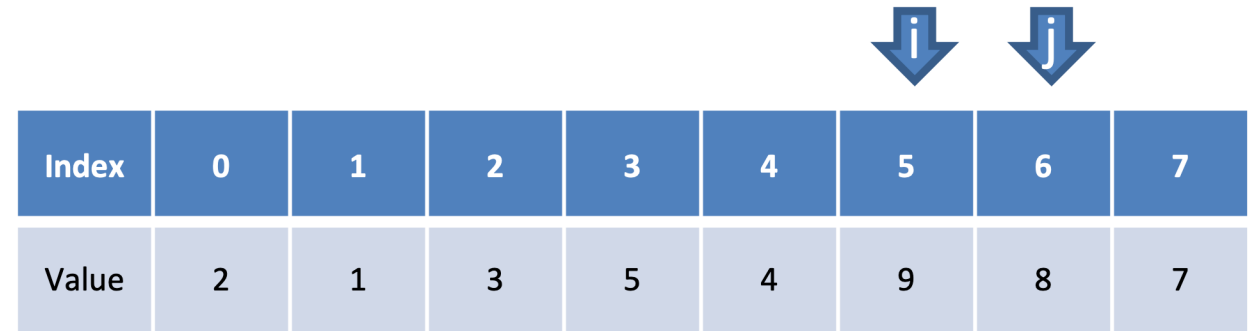
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 1 | 3 | 5 | 4 | 9 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



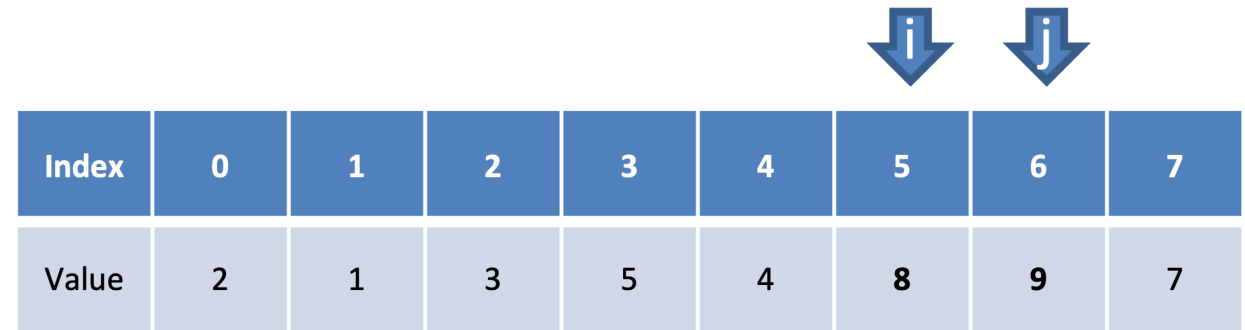
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 1 | 3 | 5 | 4 | 9 | 8 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 1 | 3 | 5 | 4 | 8 | 9 | 7 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



|       |   |   |   |   |   |   | ↓<br>i | ↓<br>j |
|-------|---|---|---|---|---|---|--------|--------|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6      | 7      |
| Value | 2 | 1 | 3 | 5 | 4 | 8 | 9      | 7      |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example


|       |   |   |   |   |   |   |  |  |
|-------|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6   | 7   |
| Value | 2 | 1 | 3 | 5 | 4 | 8 | 7   | 9   |

$$j = i+1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example



| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|---|---|---|---|---|---|---|---|
| Value | 2 | 1 | 3 | 5 | 4 | 8 | 7 | 9 |

$$j = i + 1$$

# Bubble Sort

- We will assume ascending order (smallest value to largest)
- Bubble Sort Algorithm
  1. Start at the first index
  2. Examine that index's *neighbor*
  3. If the neighbor has a smaller value, then swap values
  4. Move to the next index
  5. If the next index is the last index and there has been at least 1 swap, then repeat Step 2

## Bubble Sort Example

|       |   |   |   |   |   |   |   |   |
|-------|---|---|---|---|---|---|---|---|
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Value | 2 | 1 | 3 | 5 | 4 | 8 | 7 | 9 |

$$j = i + 1$$



# Bubble Sort

```
/*
 * Written by JJ Shepherd
 */
import java.util.Scanner;
public class BubbleSort {

    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Welcome to the Bubble Sort Program!\nEnter the size of the
Array.");
        int size = keyboard.nextInt();
        if(size <= 0)
        {
            System.out.println("That is an invalid size.");
            System.exit(0);
        }
        int[] a = new int[size];
        for(int i=0;i<a.length;i++)
        {
            System.out.println("Enter value at index "+i);
            a[i] = keyboard.nextInt();
        }
        //Bubble Sort
        boolean hasSwapped = true;
        while(hasSwapped)
        {
            hasSwapped = false;
            for(int i=0;i<a.length-1;i++)
            {
                if(a[i] > a[i+1])
                {
                    //Swap
                    int temp = a[i];
                    a[i] = a[i+1];
                    a[i+1] = temp;
                    hasSwapped = true;
                }
            }
        }
        //Print values
        System.out.println("The sorted array is");
        for(int i=0;i<a.length;i++)
        {
            System.out.println(a[i]);
        }
    }
}
```

# Computational Complexity of Bubble Sort

- Worst case in number of comparisons?
- Best case in number of comparisons?

# More Efficient Sorting Algorithms?