

Sorting

- Sorting is the process of arranging a list of items into either **ascending** (default in most cases) or **descending** order
 - Numerical order, alphabetical order

Sorting Algorithms

- **Selection sort** (in CS1101)
- **Insertion sort** (in CS1101)
- Bubble sort
- Merge sort
- Quick sort
- Heap sort
- ...

Selection Sort

zyBook Chap 7.11

Selection Sort

- Orders a list of values by **repeatedly**
 - **selecting** the **smallest** or **largest** value from the **unsorted** subarray, and
 - attaching it to the **end** of the **sorted subarray**

Selection Sort (Ascending Order)

- The algorithm:

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

Selection Sort (Ascending Order)

- The algorithm:
 - Traverse the array to find the smallest value

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

Selection Sort (Ascending Order)

- The algorithm:
 - Traverse the array to find the smallest value

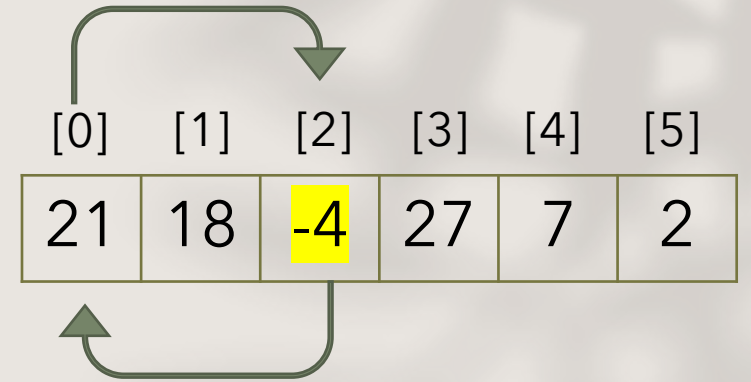
Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

Selection Sort (Ascending Order)

- The algorithm:
 - Traverse the array to find the smallest value
 - Swap it with the element at index 0

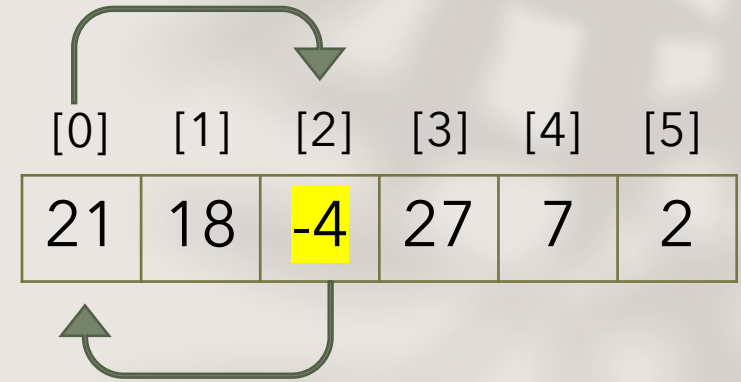
Initial Array



Selection Sort (Ascending Order)

- The algorithm:
 - Traverse the array to find the smallest value
 - Swap it with the element at index 0

Initial Array



The diagram shows an array with six elements: 21, 18, -4, 27, 7, and 2. The element -4 at index [2] is highlighted in yellow. A curved arrow points from index [2] to index [0], and another curved arrow points from index [0] back to index [2], indicating a swap between these two positions.

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

-4	18	21	27	7	2
----	----	----	----	---	---

Selection Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value

After 1st iteration

-4	18	21	27	7	2
----	----	----	----	---	---

OR, generally, we can take it as:

Traverse the unsorted part of the array to find the smallest value among the remaining elements

Selection Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value

After 1st iteration

-4	18	21	27	7	2
----	----	----	----	---	---

OR, generally, we can take it as:

Traverse the unsorted part of the array to find the smallest value among the remaining elements

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

-4	18	21	27	7	2
----	----	----	----	---	---

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

-4	18	21	27	7	2
----	----	----	----	---	---



The diagram illustrates the swap operation. A curved arrow starts from the cell containing -4 (index 0) and points to the cell containing 2 (index 5). Another curved arrow starts from the cell containing 2 (index 5) and points back to the cell containing -4 (index 0), indicating a swap of these two elements.

Selection Sort (Ascending Order)

- The algorithm:

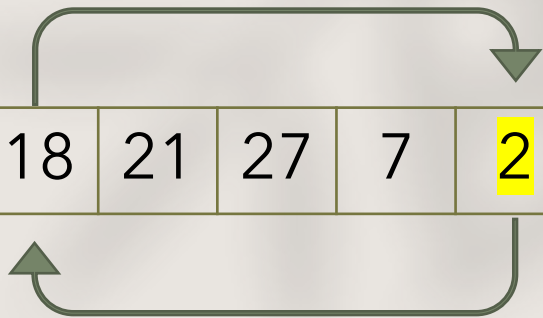
- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

-4	18	21	27	7	2
----	----	----	----	---	---



The diagram illustrates the swap operation after the first iteration. A curved arrow starts from the element -4 at index 0 and points to the element 2 at index 5. Another curved arrow starts from the element 2 at index 5 and points back to the element -4 at index 0, indicating a swap.

After 2nd iteration

-4	2	21	27	7	18
----	---	----	----	---	----

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1
- ...
- Repeat until all values are in their proper places

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	2	21	27	7	18
----	---	----	----	---	----

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1
- ...
- Repeat until all values are in their proper places

	[0]	[1]	[2]	[3]	[4]	[5]
Initial Array	21	18	-4	27	7	2

After 1st iteration	-4	18	21	27	7	2
---------------------	----	----	----	----	---	---

After 2nd iteration	-4	2	21	27	7	18
---------------------	----	---	----	----	---	----

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1
- ...
- Repeat until all values are in their proper places

Initial Array

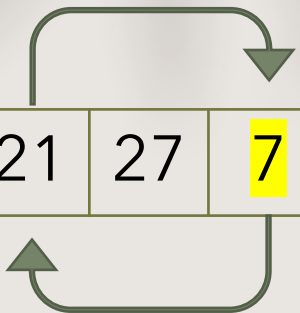
[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	2	21	27	7	18
----	---	----	----	---	----



The diagram illustrates the swap operation after the second iteration. A curved arrow points from the element '7' at index 4 to the element '2' at index 1. Another curved arrow points from the element '2' at index 1 to the element '7' at index 4, indicating their positions are swapped.

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1
- ...
- Repeat until all values are in their proper places

Initial Array

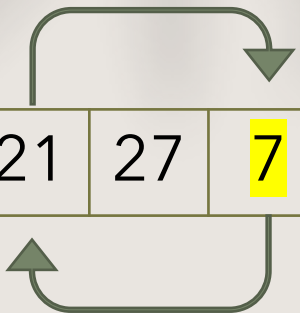
[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

-4	18	21	27	7	2
-----------	----	----	----	---	----------

After 2nd iteration

-4	2	21	27	7	18
-----------	----------	----	----	----------	----



After 3rd iteration

-4	2	7	27	21	18
-----------	----------	----------	----	----	----

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1
- ...
- Repeat until all values are in their proper places

	[0]	[1]	[2]	[3]	[4]	[5]
Initial Array	21	18	-4	27	7	2

After 1st iteration	-4	18	21	27	7	2
---------------------	----	----	----	----	---	---

After 2nd iteration	-4	2	21	27	7	18
---------------------	----	---	----	----	---	----

After 3rd iteration	-4	2	7	27	21	18
---------------------	----	---	---	----	----	----

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1
- ...
- Repeat until all values are in their proper places

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

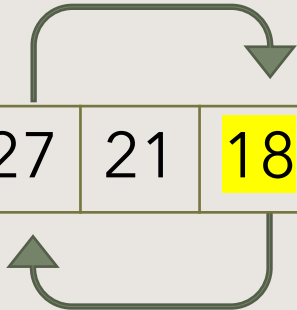
-4	18	21	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	2	21	27	7	18
----	---	----	----	---	----

After 3rd iteration

-4	2	7	27	21	18
----	---	---	----	----	----



The diagram shows two curved arrows indicating a swap between the elements at index 3 (27) and index 5 (18) in the array after the 3rd iteration.

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1
- ...
- Repeat until all values are in their proper places

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

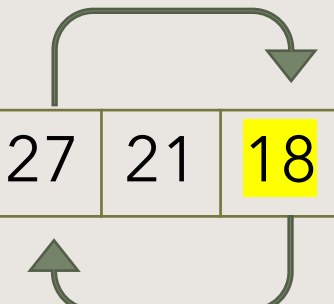
-4	18	21	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	2	21	27	7	18
----	---	----	----	---	----

After 3rd iteration

-4	2	7	27	21	18
----	---	---	----	----	----



The diagram shows two curved arrows indicating a swap. One arrow starts at the element '18' at index 5 and points to the element '21' at index 4. The other arrow starts at the element '21' at index 4 and points to the element '18' at index 5.

After 4th iteration

-4	2	7	18	21	27
----	---	---	----	----	----

DONE?

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1
- ...
- Repeat until all values are in their proper places

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	2	21	27	7	18
----	---	----	----	---	----

After 3rd iteration

-4	2	7	27	21	18
----	---	---	----	----	----

After 4th iteration

-4	2	7	18	21	27
----	---	---	----	----	----

Selection Sort (Ascending Order)

- The algorithm:

- Traverse the array to find the smallest value
- Swap it with the element at index 0
- Traverse the array to find the 2nd-smallest value
- Swap it with the element at index 1
- ...
- Repeat until all values are in their proper places

**** Single element array is naturally sorted**

	[0]	[1]	[2]	[3]	[4]	[5]
Initial Array	21	18	-4	27	7	2
After 1st iteration	-4	18	21	27	7	2
After 2nd iteration	-4	2	21	27	7	18
After 3rd iteration	-4	2	7	27	21	18
After 4th iteration	-4	2	7	18	21	27
After 5th iteration	-4	2	7	18	21	27

```
import java.util.Arrays;
```

```
public class SelectionSort {  
    public static void main(String[] args) {  
        int[] arr = {21, 18, -4, 27, 7, 2};  
        System.out.println("Initial Array: " + Arrays.toString(arr));  
        selectionSort(arr);  
    }  
  
    public static void selectionSort(int[] arr) {  
        // Notice the range is arr.length - 1, as size 1 array is naturally sorted  
        for (int i = 0; i < arr.length - 1; ++i) {  
            // Let the front most element be the current smallest value  
            int smallest = i;  
            // Traverse the remaining part of the array (j = i + 1), and...  
            for (int j = i + 1; j < arr.length; ++j) {  
                // Compare each element with the current smallest value  
                if (arr[j] < arr[smallest]) {  
                    smallest = j; // Update the index the smallest value  
                }  
            }  
            swap(arr, i, smallest); // Swap smallest to front  
            System.out.println("After iteration #" + i + ": " + Arrays.toString(arr));  
        }  
    }  
}
```

```
    public static void swap(int[] arr, int i, int j) {  
        int temp = arr[i];  
        arr[i] = arr[j];  
        arr[j] = temp;  
    }  
}
```

Selection Sort Implementation

```
$ javac SelectionSort.java  
$ java SelectionSort  
Initial Array: [21, 18, -4, 27, 7, 2]  
After iteration #0: [-4, 18, 21, 27, 7, 2]  
After iteration #1: [-4, 2, 21, 27, 7, 18]  
After iteration #2: [-4, 2, 7, 27, 21, 18]  
After iteration #3: [-4, 2, 7, 18, 21, 27]  
After iteration #4: [-4, 2, 7, 18, 21, 27]
```


Insertion Sort

zyBook Chap 7.10

Insertion Sort

- Orders a list of values by **repeatedly**
 - picking the first value from the **unsorted subarray**, and
 - **inserting** it into its **proper** place in a **sorted subarray**

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:
 1. Pick the first value from the unsorted portion of the array

**** The first element is considered sorted**

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:
 1. Pick the first value from the unsorted portion of the array

**** The first element is considered sorted**

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:
 1. Pick the first value from the unsorted portion of the array
 2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value

Insertion Sort (Ascending Order)

Initial Array

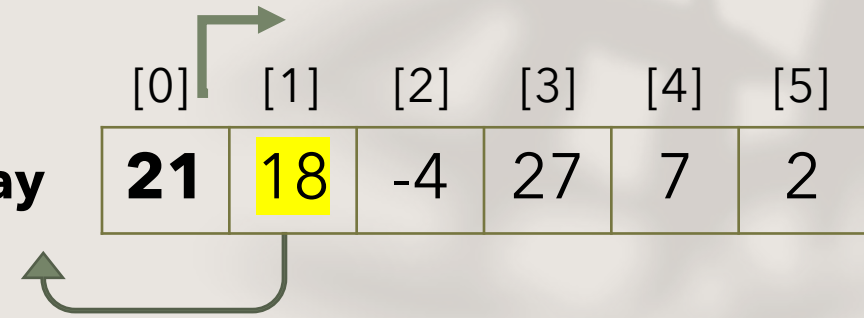
[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2



- The algorithm:
 1. Pick the first value from the unsorted portion of the array
 2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value

Insertion Sort (Ascending Order)

Initial Array



[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

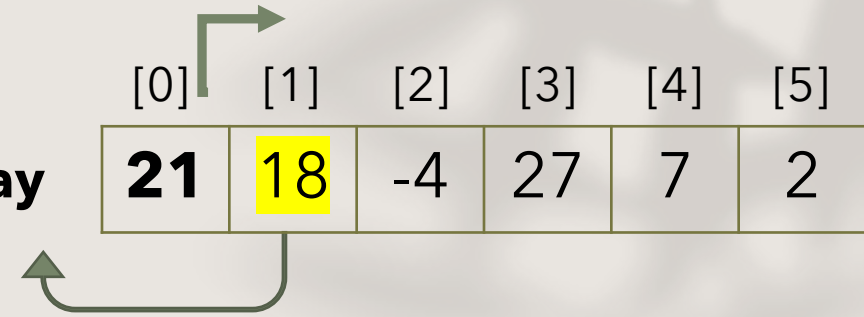
- The algorithm:
 1. Pick the first value from the unsorted portion of the array
 2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
 3. Shift items as needed to make room to insert the new addition

**** Technically, the algorithm repeatedly swaps the target with the element on its left, until**

1. It is no longer less than the element on its left, OR
2. It becomes the left-most element

Insertion Sort (Ascending Order)

Initial Array



[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

Insertion Sort (Ascending Order)

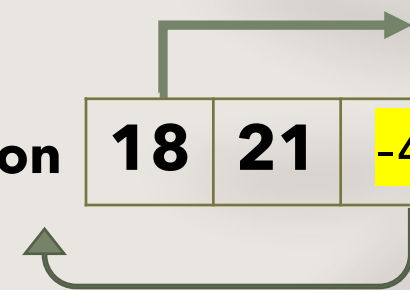
- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

After 1st iteration



18	21	-4	27	7	2
----	----	----	----	---	---

Technically, the algorithm
swaps arr[2] with arr[1] // [18, -4, 21, ...
swaps arr[1] with arr[0] // [-4, 18, 21, ...

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 3rd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 3rd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 3rd iteration

-4	18	21	27	7	2
----	----	----	----	---	---



Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 3rd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 4th iteration

-4	7	18	21	27	2
----	---	----	----	----	---



Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 3rd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 4th iteration

-4	7	18	21	27	2
----	---	----	----	----	---

Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 3rd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 4th iteration

-4	7	18	21	27	2
----	---	----	----	----	---



Insertion Sort (Ascending Order)

Initial Array

[0]	[1]	[2]	[3]	[4]	[5]
21	18	-4	27	7	2

- The algorithm:

1. Pick the first value from the unsorted portion of the array
2. Traverse the sorted portion and look for an element smaller than what you just picked
 - If found, insert the new value after it
 - Otherwise, insert as the first value
3. Shift items as needed to make room to insert the new addition
4. Repeat until all values are in their proper places

After 1st iteration

18	21	-4	27	7	2
----	----	----	----	---	---

After 2nd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 3rd iteration

-4	18	21	27	7	2
----	----	----	----	---	---

After 4th iteration

-4	7	18	21	27	2
----	---	----	----	----	---



After 5th iteration

-4	2	7	18	21	27
----	---	---	----	----	----

```
import java.util.Arrays;
```

```
public class InsertionSort {  
    public static void main(String[] args) {  
        int[] arr = {21, 18, -4, 27, 7, 2};  
        System.out.println("Initial Array: " + Arrays.toString(arr));  
        insertionSort(arr);  
    }  
}
```

```
public static void insertionSort(int[] arr) {  
    // Notice the for loop starts at index 1, as the first element is naturally sorted  
    for (int i = 1; i < arr.length; ++i) {  
        // Starting with the first element in the unsorted part  
        int j = i;  
        // Traverse the sorted portion, and  
        // repeatedly swap the target with the element on its left, until  
        // 1. It is no longer less than the element on its left, OR  
        // 2. It becomes the left-most element  
        while (j > 0 && arr[j] < arr[j - 1]) {  
            swap(arr, j, j - 1);  
            --j;  
        }  
        System.out.println("After iteration #" + i + ": " + Arrays.toString(arr));  
    }  
}
```

```
public static void swap(int[] arr, int i, int j) {  
    int temp = arr[i];  
    arr[i] = arr[j];  
    arr[j] = temp;  
}
```

Insertion Sort Implementation

```
$ javac SelectionSort.java  
$ java SelectionSort  
Initial Array: [21, 18, -4, 27, 7, 2]  
After iteration #1: [18, 21, -4, 27, 7, 2]  
After iteration #2: [-4, 18, 21, 27, 7, 2]  
After iteration #3: [-4, 18, 21, 27, 7, 2]  
After iteration #4: [-4, 7, 18, 21, 27, 2]  
After iteration #5: [-4, 2, 7, 18, 21, 27]
```

Q: Which sorting algorithm is used given the following output?

```
Initial Array: [25, -2, 9, 1, 8, 5, -5, 30]
After iteration #0: [-5, -2, 9, 1, 8, 5, 25, 30]
After iteration #1: [-5, -2, 9, 1, 8, 5, 25, 30]
After iteration #2: [-5, -2, 1, 9, 8, 5, 25, 30]
After iteration #3: [-5, -2, 1, 5, 8, 9, 25, 30]
After iteration #4: [-5, -2, 1, 5, 8, 9, 25, 30]
After iteration #5: [-5, -2, 1, 5, 8, 9, 25, 30]
After iteration #6: [-5, -2, 1, 5, 8, 9, 25, 30]
```

Selection Sort

Q: Sort the following array with 1) selection sort, and 2) insertion sort.
Show the resulting array after each iteration.

[3, 6, 2, 10, 5, 1, 9]

Selection Sort:

After iteration #0: [1, 6, 2, 10, 5, 3, 9]
After iteration #1: [1, 2, 6, 10, 5, 3, 9]
After iteration #2: [1, 2, 3, 10, 5, 6, 9]
After iteration #3: [1, 2, 3, 5, 10, 6, 9]
After iteration #4: [1, 2, 3, 5, 6, 10, 9]
After iteration #5: [1, 2, 3, 5, 6, 9, 10]

Insertion Sort:

After iteration #1: [3, 6, 2, 10, 5, 1, 9]
After iteration #2: [2, 3, 6, 10, 5, 1, 9]
After iteration #3: [2, 3, 6, 10, 5, 1, 9]
After iteration #4: [2, 3, 5, 6, 10, 1, 9]
After iteration #5: [1, 2, 3, 5, 6, 10, 9]
After iteration #6: [1, 2, 3, 5, 6, 9, 10]