

NAME

DATE

PERIOD

UNIT 4 · TOPIC 4.15 · WORKSHEET

Sort It Out

Select, swap, shift, and insert, then catch the sorts that only look like they're working.

About 30 minutes

9 questions

Cumulative through 4.15

TOPIC

4.15



DIRECTIONS

Work each puzzle in pencil. Every sort here puts values in ascending order (smallest first). A *pass* is one run of the outer loop. Some puzzles include pieces you won't need.

PART A · 3 QUESTIONS

A

Warm up your brain

1

PENCIL CHECK

Sort 38 14 52 9 27 both ways. Write the whole array after each pass.

After pass	Selection sort	Insertion sort
1		
2		
3		
4		

2 YOU'RE THE JVM

This is selection sort, stopped after two passes. What prints?

```
int[] arr = {50, 20, 40, 10, 30};
for (int i = 0; i < 2; i++) {
    int minIndex = i;
    for (int j = i + 1; j < arr.length; j++) {
        if (arr[j] < arr[minIndex]) { minIndex = j; }
    }
    int temp = arr[i];
    arr[i] = arr[minIndex];
    arr[minIndex] = temp;
}
for (int v : arr) { System.out.print(v + " "); }
```

A 10 20 40 50 30

B 10 50 20 40 30

C 20 50 40 10 30

D 10 20 30 40 50

3 MATCH-UP

Each row shows the array 8 5 6 1 4 partway through a sort. Match each state to how it was made, from the list. Each choice is used once.

State	Array	How it was made
1	1 4 6 8 5	
2	5 8 6 1 4	
3	1 8 5 6 4	
4	1 5 6 8 4	
5	5 6 8 1 4	

- **P** selection sort, after 1 pass
- **Q** selection sort, after 2 passes
- **R** insertion sort, after 1 pass
- **S** insertion sort, after 2 passes
- **T** neither: a mistake that slides values instead of swapping

PART B · 2 PUZZLES

B Puzzles

4 CODE TILES

Selection sort's pass fell off the fridge. Put tiles in the outer loop so sorting {23, 41, 12, 8} prints the array after each pass as shown. One tile is a decoy.

```
public static void sort(int[] arr) {
    for (int i = 0; i < arr.length - 1; i++) {
        // your tiles go here
        for (int v : arr) { System.out.print(v + " "); }
        System.out.println();
    }
}
```

```
if (arr[j] < arr[minIndex]) { minIndex = j; }
```

```
int temp = arr[i]; arr[i] = arr[minIndex]; arr[minIndex] = temp;
```

```
int minIndex = i;
```

```
}
```

```
if (arr[j] > arr[minIndex]) { minIndex = j; }
```

```
for (int j = i + 1; j < arr.length; j++) {
```

Target output

```
8 41 12 23
```

```
8 12 41 23
```

```
8 12 23 41
```

YOUR TILES, IN ORDER



FAIR QUESTION

Why does the outer loop stop at `arr.length - 1`?

Once every other spot holds its smallest value, whatever is left in the last spot must be the largest, so it's already in place. Another pass would only swap it with itself.

5 SNIPPET POOL

Fill each numbered hole with a snippet from the pool so this insertion sort prints 1 2 5 8 9. Not every snippet is used.

```
int[] arr = {5, 2, 8, 1, 9};
for (int i = 1; i < arr.length; i++) {
    int current = (1);
    int j = i - 1;
    while ((2)) {
        arr[j + 1] = arr[j];
        j--;
    }
    arr[(3)] = current;
}
for (int v : arr) {
    System.out.print(v + " ");
}
```

THE POOL

Each snippet can be used only once, and not every snippet is needed.

<code>arr[i]</code>	<code>j > 0 && arr[j] > current</code>
<code>j</code>	<code>arr[j] > current && j >= 0</code>
<code>arr[i - 1]</code>	<code>j >= 0 && arr[j] > current</code>
<code>j + 1</code>	<code>i</code>



WATCH OUT

Insertion sort must save `arr[i]` in `current` **before** the first shift. The first shift copies a larger value into `arr[i]`, so reading `arr[i]` afterward gets that copy, and the value you meant to insert is gone.

PART C · 4 QUESTIONS

C Catch the mistake



THINK FIRST

Both sorts build a sorted part at the front. Is that part *finished* in selection sort? In insertion sort? Guess in the margin; question 7 settles it.

6 ONE-TOKEN FIX

This selection sort should print 7 12 18 29 45. It prints 45 7 12 18 29. Change exactly one token to fix it.

```
1 int[] arr = {29, 12, 45, 7, 18};
2 for (int i = 0; i < arr.length - 1; i++) {
3     int minIndex = 0;
4     for (int j = i + 1; j < arr.length; j++) {
5         if (arr[j] < arr[minIndex]) { minIndex = j; }
6     }
7     int temp = arr[i];
8     arr[i] = arr[minIndex];
9     arr[minIndex] = temp;
10 }
11 for (int v : arr) { System.out.print(v + " "); }
```

LINE, OLD TOKEN, NEW TOKEN

7 TWO TRUTHS AND A LIE

Insertion sort runs 2 passes on 9 4 6 1 7, and the array becomes 4 6 9 1 7. Two of the claims are true and one is a lie. Circle the lie, then rewrite it so it's true.

- 1 4, 6, and 9 are in order relative to each other.
- 2 4 is in its final position.
- 3 1 and 7 haven't been moved yet.

THE LIE, REWRITTEN

8 FACE-OFF

Tonight: Selection Sort and Insertion Sort argue about who works harder. Fill in the blanks.

SELECTION SORT	Each pass, I find the smallest value left and _____ it into place. Exactly _____ elements trade spots.
INSERTION SORT	I take the next value and _____ the larger ones right to make room for it.
SELECTION SORT	And once I place a value, it's in its _____ position. Yours might get bumped later.
INSERTION SORT	Maybe. But hand me an array that's already sorted and I make _____ shifts. I'm done fast.
SELECTION SORT	I can't know a value is the smallest without checking all the rest. Six sorted values still cost me _____ comparisons.

9 YOU'RE THE COMPILER

Sorting an ArrayList uses get and set. Each snippet runs after the setup code below. Decide whether it compiles. If it does, write what it prints. If not, say why.

```
import java.util.ArrayList;
ArrayList<Integer> n = new ArrayList<Integer>();
n.add(5); n.add(3);
ArrayList<String> w = new ArrayList<String>();
w.add("pear"); w.add("fig");
```

Snippet	Compiles?	Output, or why not
1 System.out.println(n.get(1) < n.get(0));		
2 System.out.println(w.get(1) < w.get(0));		
3 n[0] = n.get(1);		
4 int c = w.get(1).compareTo(w.get(0)); System.out.println(c < 0);		
5 n.set(0, n.get(1)); System.out.println(n.get(0) + n.get(1));		