WORKSHEET: Sorting and `ArrayList`

1. We have studied ***bubble***, ***selection***, and ***insertion*** sort. The following are brief descriptions of each sorting algorithm. Write the name of the algorithm next to its description. The description of the remaining algorithm is for ***merge*** sort, so also write that next to its description.

   a) builds a progressively longer sorted portion of the array by shifting elements and placing each element into its correct position.

   b) repeatedly compares and swaps adjacent elements to bring the largest element to the end (or smallest elements to the beginning).

   c) divides the array into halves, recursively sorts them, and merges them back together

   d) for each position, searches for the smallest (or largest) element and swaps it into the current position.

insertion
bubble
merge
selection

2. It is common to create arrays of primitive types (`int`, `double`, `boolean`, etc.). In contrast, because `ArrayList` can only accept an object in its *type parameter*, wrapper classes (`Integer`, `Double`, `Boolean`, etc.) must be used. Answer the following questions about wrapper classes and `ArrayList`.

   a) Write a statement to declare and initialize an empty `ArrayList` that can store real numbers.

```
ArrayList<Double> myList = new ArrayList<>();
```

   b) Write a statement to declare and initialize an empty `ArrayList` that can store integers.

```
ArrayList<Integer> myList = new ArrayList<>();
```

   c) In most cases, Java will automatically unbox wrapper types, such as `Integer` and `Double`. However, for the *AP Java Subset*, you must know how to explicitly get the `int` or `double` value from the `Integer` or `Double` object. Write a statement that declares a `double` value named `d`, and sets it to the value in the `Double` object named `myDouble`. Then write a second statement for an `int` value named `i`, setting it to the value in the `Integer` object named `myInteger`.

```
double d = myDouble.doubleValue();
int i = myInteger.intValue();
```

   d) When comparing two `Integer` values, comparison operators such as `<`, `<=`, `>`, and `>=` will automatically unbox the `int` values from the `Integer` objects. However, when using the equality operator (`==`), it will compare the references, which might produce unexpected results.
   Write a statement that will compare two `Integer` objects, `x` and `y`. *Hint*: this is how you've been told you should always compare two objects for equality.

```
x.equals(y)
```

WORKSHEET: Sorting and `ArrayList`

3. Given the following code for a sorting algorithm, write a version that replaces the array parameter with an `ArrayList` parameter. *Note*: you will not need to use the `intValue` method, as comparisons of wrapper types using `<`, `<=`, `>`, or `>=` are automatically unboxed.

```
1  public static void sort1(int[] a) {
2     for(int i=0; i < a.length-1; i++) {
3       for(int j=0; j<a.length-1 - i; j++) {
4          if( a[j] > a[j+1] ) {
5             swap(a, j, j+1);
6          }
7       }
8     }
9  }
```

```
1  public static void bubbleSort(ArrayList<Integer> a) {
2     for (int i = 0; i < a.size() - 1; i++) {
3       for (int j = 0; j < a.size()-1 - i; j++) {
4          if (a.get(j) > a.get(j + 1)) {
5             swap(a, j, j+1);
6          }
7       }
8     }
9  }
```

a) Write the name of this sorting algorithm in the box to the right.

`bubble`

b) In a complete English sentence, describe the purpose of the `if` statement (lines `4` through `6`).

The `if` statement compares each pair of adjacent elements in the unsorted portion of the array and swaps them if they are in the wrong order.

c) Why does the inner loop's range (in line `3`) decrease with each iteration of the outer loop?

After each iteration of the outer loop, the largest unsorted element is placed at the end of the array. Therefore, the inner loop's range decreases to avoid redundant comparisons with elements that are already sorted.

WORKSHEET: Sorting and `ArrayList`

4. Given the following code for a sorting algorithm, write a version that replaces the array parameter with an `ArrayList` parameter.

```
1  public static void sort2(double[] e) {
2     for (int j=0; j<e.length-1; j++) {
3        int minIndex = j;
4        for(int k=j+1; k<e.length; k++) {
5           if(e[k] < e[minIndex]) {
6              minIndex = k;
7           }
8        }
9        if (j != minIndex) {
10          swap(e, j, minIndex);
11       }
12    }
13 }
```

```
1  public static void selectionSort(ArrayList<Double> e) {
2     for (int j = 0; j < e.size() - 1; j++) {
3        int minIndex = j;
4        for (int k = j + 1; k < e.size(); k++) {
5           if (e.get(k) < e.get(minIndex)) {
6              minIndex = k;
7           }
8        }
9        if (j != minIndex) {
10          swap(e, j, minIndex);
11       }
12    }
13 }
```

a) Write the name of this sorting algorithm in the box to the right.

`selection`

b) Describe the purpose of the inner loop (lines 3 through 8) in a complete sentence.

The inner loop finds the index of the smallest element in the array (so that afterwards it can be swapped to the front).

b) Describe the purpose of lines 9.

There is no point in wasting time swapping the elements if the values are the same!

5. Given the following code for a sorting algorithm, write a version that replaces the array parameter with an `ArrayList` parameter.

```
1  public static void insertionSort(int[] e) {
2     for (int j = 1; j < e.length; j++) {
3        int temp = e[j];
4        int possibleIndex = j;
5        while (possibleIndex > 0 && temp < e[possibleIndex-1]) {
6           e[possibleIndex] = e[possibleIndex-1];
7           possibleIndex--;
8        }
9        e[possibleIndex] = temp;
10    }
11 }
```

```
1  public static void insertionSort(ArrayList<Integer> e) {
2     for (int j = 1; j < e.size(); j++) {
3       int temp = e.get(j);
4       int possibleIndex = j;
5       while (possibleIndex>0 && temp<e.get(possibleIndex-1)) {
6           e.set(possibleIndex, e.get(possibleIndex – 1));
7           possibleIndex--;
8        }
9        e.set(possibleIndex, temp);
10    }
11 }
```

a) In a complete English sentence, state the purpose of line 6 of the code given.

Line 6 shifts an element in the array right by one position. This makes space to insert each element into the correct position of the sorted array.

c) Notice that in line 7, we are decrementing the loop counter, meaning we are iterating from the end of the sorted portion of the list towards the beginning. Why do we iterate through the array in reverse order?

reverse iteration ensures that elements are moved to the right without overwriting any important data before it's placed in the correct spot.