Exercise walk-through

4.4 Array Traversals

AP Computer Science A

The five standard array algorithms

sum or average	an accumulator declared outside the loop; divide by `length` at the end
max or min	start from `arr[0]`, not from zero – an array of negatives has no element above zero
count matching	a counter, incremented only when the element passes the test
check for a duplicate	a nested loop, with the inner one starting at `i+1` so no pair is checked twice
reverse or copy	build into a new array, or swap ends inward for an in-place reverse

> Every one of these is a traversal plus a small body. Recognising which one a question wants is most of the work.

Why a data structure at all

the problem	a single variable holds one value; real problems need many related values – a class roster, pixels, sensor readings
a data structure	holds many values under one name, reached by index
an array	fixed length, set when it is created, and `arr.length` is a field, not a method
the bounds	valid indices run 0 to `length-1`; anything outside throws an `ArrayIndexOutOfBoundsException`
the run-time cost	reaching any element takes the same time whatever the index – that is the point of an array

> Off-by-one at the top end is the classic array bug: `i <= arr.length` throws, `i < arr.length` does not.

Removing from a list while looping over it

the problem	`remove(i)` shifts every later element down, so the element after the removed one is skipped
fix one	loop backwards, from `size()-1` down to `0` – later shifts cannot affect indices you have passed
fix two	do not increment `i` after a removal, so the next pass re-examines the same index
never	remove inside an enhanced `for` loop – it throws a `ConcurrentModificationException`
the rest	the array algorithms carry over unchanged, plus insertion and deletion, which arrays cannot do easily

> This is the single most-tested ArrayList point. If a question removes items in a loop, it is testing exactly this.

Exercises 2.1–2.2

2.1 Write an enhanced `for` loop that prints every element of `d` on its own line. [2]

2.2 State the values visited by `for (int i = 1; i < d.length; i += 2)`. [1]

Exercises 2.1·2.2

2.1 Write an enhanced `for` loop that prints every element of `d` on its own line. [2]

2.1

```
for (int v : d)
{
    System.out.println(v);
}
```

- enhanced `for` header with type `int`
- prints `v` inside the loop

2.2 State the values visited by `for (int i = 1; i < d.length; i += 2)`. [1]

2.2

- indexes 1, 3, 5: **3, 1, 2**

Exercises 2.3·2.4

2.3 State why `for (int i = 0; i <= d.length; i++)` causes an error. [1]

2.4 Explain why an enhanced `for` loop cannot be used to **double** every element of an array. [1]

Exercises 2.3·2.4

2.3 State why `for (int i = 0; i <= d.length; i++)` causes an error. [1]

2.3

- when `i` equals `d.length` (6), `d[6]` is **out of bounds**: an `ArrayIndexOutOfBounds Exception`

2.4 Explain why an enhanced `for` loop cannot be used to **double** every element of an array. [1]

2.4

- the loop variable is a **copy** of each value; assigning to it does not change the array

Exercise 3.1

3.1 What is printed? [1]

```
int[] arr = {4, 7, 2, 9};
int total = 0;
for (int i = arr.length - 1; i > 0; i--)
{
    total += arr[i];
}
System.out.println(total);
```

- ☐ **A** 22
- **B** 18
- **C** 13
- **D** 11

Exercise 3.1

3.1 What is printed? [1]

```
int[] arr = {4, 7, 2, 9};
int total = 0;
for (int i = arr.length - 1; i > 0; i--)
{
    total += arr[i];
}
System.out.println(total);
```

- ☐ **A** 22
- **B** 18
- **C** 13
- **D** 11

3.1 B

- visits indexes 3, 2, 1 (stops before 0): $9 + 2 + 7 = 18$

Exercise 3.2

3.2 What are the contents of `vals` after the code runs? [1]

```
int[] vals = {1, 2, 3, 4};
for (int i = 1; i < vals.length; i++)
{
    vals[i] = vals[i - 1] + vals[i];
}
```

- **A** `{1, 3, 5, 7}`
- **B** `{1, 3, 6, 10}`
- **C** `{1, 2, 3, 4}`
- **D** `{2, 4, 6, 8}`

Exercise 3.2

3.2 What are the contents of `vals` after the code runs? [1]

```
int[] vals = {1, 2, 3, 4};
for (int i = 1; i < vals.length; i++)
{
    vals[i] = vals[i - 1] + vals[i];
}
```

- ☐ **A** `{1, 3, 5, 7}`
- **B** `{1, 3, 6, 10}`
- **C** `{1, 2, 3, 4}`
- **D** `{2, 4, 6, 8}`

3.2 B

- each element becomes itself plus the **new** previous one: 1, 3, 6, 10

Exercise 3.3

3.3 Which loop prints the elements of `int[] a` from **last** to **first**? [1]

- [ ] **A** `for (int i = a.length; i >= 0; i--) System.out.println(a[i]);`
- [ ] **B** `for (int i = a.length - 1; i >= 0; i--) System.out.println(a[i]);`
- [ ] **C** `for (int i = a.length - 1; i > 0; i--) System.out.println(a[i]);`
- [ ] **D** `for (int v : a) System.out.println(a[a.length - v]);`

Exercise 3.3

3.3 Which loop prints the elements of `int[] a` from **last** to **first**? [1]

- [ ] **A** `for (int i = a.length; i >= 0; i--) System.out.println(a[i]);`
- [ ] **B** `for (int i = a.length - 1; i >= 0; i--) System.out.println(a[i]);`
- [ ] **C** `for (int i = a.length - 1; i > 0; i--) System.out.println(a[i]);`
- [ ] **D** `for (int v : a) System.out.println(a[a.length - v]);`

3.3 B

- A starts out of bounds; C misses index 0; D uses values as indexes

Exercise 3.4 (a)

3.4 A weather station stores the rainfall, in mm, for each day of a month in `double[] rain`.

(a) Write code that sets `wetDays` to the number of days with **more than 5.0** mm of rain. [3]

Exercise 3.4 (a)

3.4 A weather station stores the rainfall, in mm, for each day of a month in `double[] rain`.

(a) Write code that sets `wetDays` to the number of days with **more than 5.0** mm of rain. [3]

(a)

```
int wetDays = 0;
for (double r : rain)
{
    if (r > 5.0)
    {
        wetDays++;
    }
}
```

- counter set to 0 before the loop
- traverses every element with the test `> 5.0`
- increments the counter inside the `if`

Exercise 3.4 (b)

3.4 A weather station stores the rainfall, in mm, for each day of a month in `double[] rain`.

(b) Write code that changes every value in `rain` from mm to **cm** (divide by 10). [2]

Exercise 3.4 (b)

3.4 A weather station stores the rainfall, in mm, for each day of a month in `double[] rain`.

(b) Write code that changes every value in `rain` from mm to **cm** (divide by 10). [2]

(b)

```
for (int i = 0; i < rain.length; i++)
{
    rain[i] = rain[i] / 10;
}
```

- indexed loop over every element
- assigns `rain[i] / 10` back to `rain[i]`

Exercise 3.4 (c)

3.4 A weather station stores the rainfall, in mm, for each day of a month in `double[] rain`.

(c) Explain why your loop in (b) must be an **indexed `for` loop.** [1]

Exercise 3.4 (c)

3.4 A weather station stores the rainfall, in mm, for each day of a month in `double[] rain`.

(c) Explain why your loop in (b) must be an **indexed `for`** loop. [1]

(c)

- only assigning to `rain[i]` changes the array; an enhanced loop changes only a copy