

Hide and Seek

Searching algorithms



Find the locker

PAUSE AND TRY

The list is in no particular order. How do you find Rosa's locker?

LOCKER LIST

Theo	214
Mina	108
Rosa	331
Jules	095
Ari	262

1. Where do you start?
2. When can you stop?
3. When can you say "Rosa isn't on the list"?

By the end of class you can...

- 1 Write a linear search that returns the index of the target, or -1 if it is missing.
- 2 Explain why a search stops at the first match, and which copy it reports.
- 3 Spot searches that give up too soon or read past the end of the array.
- 4 Count the comparisons a search makes in the best and worst cases.
- 5 Search an ArrayList of strings, search from the end, and search a 2D array row by row.

A white rounded square with a thin black border, containing a large, bold, black number '1' in the center.

1

PART 1

Check each element in order

Linear search returns an index

```
public static int findIndex(int[] values, int target) {  
    for (int i = 0; i < values.length; i++) {  
        if (values[i] == target) {  
            return i;  
        }  
    }  
    return -1;  
}  
  
// in main  
int[] jerseys = {23, 7, 11, 30, 7};  
System.out.println("11: " + findIndex(jerseys, 11));  
System.out.println("99: " + findIndex(jerseys, 99));
```

OUTPUT

```
11: 2  
99: -1
```



A **linear search** checks each element in order until it finds the **target** or runs out.

Found: return the index right away. **Missing:** return -1 after the loop.

Why -1 means "not found"

Return value	What it could mean	Safe for "not found"?
0	The target is the first element	No: it is a real index
length - 1	The target is the last element	No: it is a real index
-1	Nothing: no array has index -1	Yes



A **sentinel value** can never be a real answer, so it can signal "not found." You have seen it before: `"hopper".indexOf("z")` is -1.

One mismatch does not prove it is missing

```
public static int findIndex(int[] values, int target) {  
    for (int i = 0; i < values.length; i++) {  
        if (values[i] == target) {  
            return i;  
        } else {  
            return -1;  
        }  
    }  
    return -1;  
}  
  
// in main  
int[] jerseys = {23, 7, 11, 30, 7};  
System.out.println("23: " + findIndex(jerseys, 23));  
System.out.println("30: " + findIndex(jerseys, 30));
```

OUTPUT

```
23: 0  
30: -1
```



"Not found" can
only be decided
after the loop.

Fix: delete the else. Keep
return -1; after the loop.

2

PART 2

Stop at the first match

A search that stops in its loop condition

```
int[] floors = {3, 8, 5, 12, 8};
int target = 12;
int index = 0;
while (index < floors.length && floors[index] != target) {
    index++;
}
if (index < floors.length) {
    System.out.println("Found at " + index);
} else {
    System.out.println("Not found");
}
```

OUTPUT

Found at 3



When the loop ends, **index < length** means found; **index == length** means every element was checked.



The search stops at the **first** match. If the target appears twice, you get the copy it reaches first.

Guard the array access



Check the index **before** you read the element: **`index < floors.length && floors[index] != target`**.

```
int[] floors = {3, 8, 5, 12, 8};
int target = 6;
int index = 0;
while (floors[index] != target && index < floors.length) {
    index++;
}
System.out.println("index is " + index);
```

The target is missing, so index reaches 5, and `floors[5]` is read first.

ERROR

```
java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
```

Which index?

PAUSE AND TRY

What prints? Then: what prints if target is 5?

```
int[] rolls = {4, 2, 6, 2, 4};  
int target = 2;  
int index = 0;  
while (index < rolls.length && rolls[index] != target) {  
    index++;  
}  
System.out.println(index);
```

Two answers. Which 2 does it find?

A white rounded square with a teal number '3' inside, tilted slightly to the right.

3

PART 3

Count the comparisons

Found at index k: k + 1 comparisons

```
public static int comparisons = 0;
public static int findIndex(int[] values, int target) {
    comparisons = 0;
    for (int i = 0; i < values.length; i++) {
        comparisons++;
        if (values[i] == target) {
            return i;
        }
    }
    return -1;
}

// in main
int[] ids = {40, 15, 62, 15, 88, 9};
findIndex(ids, 62);
System.out.println(comparisons);
findIndex(ids, 77);
System.out.println(comparisons);
```

OUTPUT

3
6

Target	Index	Comparisons
40	0	1
62	2	3
9	5	6
77	-1	6

Missing costs one comparison per element.

Best case, worst case

PAUSE AND TRY

A linear search runs on an array of 200 values.

- A** What is the fewest comparisons it can make? When?
- B** The target is at index 37. How many comparisons?
- C** What is the most it can make? Name two situations that cause it.

A large, bold, teal number '4' is centered within a white rounded square that has a slight 3D effect and a shadow.

PART 4

Strings, the other end, and 2D

Searching an ArrayList of strings

```
import java.util.ArrayList;
public static int find(ArrayList<String> list, String t) {
    for (int i = 0; i < list.size(); i++) {
        if (list.get(i).equals(t)) {
            return i;
        }
    }
    return -1;
}
// in main
ArrayList<String> songs = new ArrayList<String>();
songs.add("Glow");
songs.add("Static");
songs.add("Tides");
System.out.println("Tides: " + find(songs, "Tides"));
System.out.println("tides: " + find(songs, "tides"));
```

OUTPUT

```
Tides: 2
tides: -1
```



Compare strings with **equals**, never **==**. And equals is case-sensitive.

Start at the end to find the last copy

```
public static int comparisons = 0;
public static int lastIndex(int[] values, int target) {
    comparisons = 0;
    for (int i = values.length - 1; i >= 0; i--) {
        comparisons++;
        if (values[i] == target) {
            return i;
        }
    }
    return -1;
}
// in main
int[] temps = {61, 58, 64, 58, 70};
int spot = lastIndex(temps, 58);
System.out.println(spot + " after " + comparisons);
spot = lastIndex(temps, 61);
System.out.println(spot + " after " + comparisons);
```

OUTPUT

```
3 after 2
0 after 5
```



A linear search can start at **either end**. Start at **length - 1**; starting at length crashes.

A 2D array: search one row at a time

```
// findIndex is the method from Part 1
public static String locate(int[][] grid, int target) {
    for (int r = 0; r < grid.length; r++) {
        int c = findIndex(grid[r], target);
        if (c != -1) {
            return "row " + r + ", col " + c;
        }
    }
    return "not found";
}

// in main
int[][] bins = { {14, 3, 27},
                  {8, 41, 19},
                  {33, 6, 22} };

System.out.println(locate(bins, 41));
System.out.println(locate(bins, 50));
```

OUTPUT

```
row 1, col 1
not found
```



Access each **row**, then run a linear search on that row. Stop when a row has the target.

Count the comparisons in a grid

PAUSE AND TRY

A 2D array has 3 rows and 6 columns.

Its only copy of the target is at row 1, column 4. How many comparisons does a row-by-row search make before it finds it?

Words to know

Linear search

Check each element in order until the target is found or all are checked.

Sequential search

Another name for linear search.

Target

The value the search is looking for.

Sentinel value

A value that can never be a real answer, like -1 for "not found."

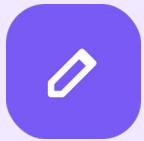
Comparison

One check of an element against the target.

Short-circuit

In a `a && b`, Java skips `b` when `a` is false.

When can a search say "not found"?



Explain to your partner when a linear search may stop, and when it may say "not found."

SENTENCE STARTER

A search can stop as soon as _____, but it can only return -1 after _____.

Next: the Hide and Seek puzzle worksheet.