Data Collections

AP Computer Science A ▪ Unit 4

AP Computer Science A

4	8	2	9

What we will cover

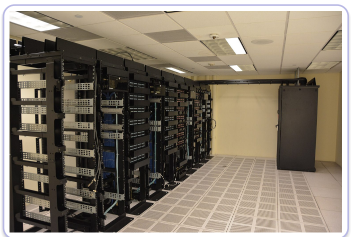
4	8	2	9	5
0	1	2	3	4

1 Ethics

The ethics of collecting data

Programs that collect and process personal data carry real responsibility.

> Collect only what you need, protect **privacy** 隐私, be honest about how data is used, and guard against **bias** 偏差 in what you gather. Legal is not always ethical.



Data centre

2 Arrays

Arrays

4	8	2	9	5	7
0	1	2	3	4	5

An **array** 数组 stores a fixed number of same-type values, reached by **index** 索引 (from **0**). Visit every element with a loop:

```
for (int i = 0; i < a.length; i++) ...
```

Standard algorithms: find the **max/min**, **sum/average**, or count matches by traversing once.

Filing cabinet

3 ArrayList

The ArrayList toolbox

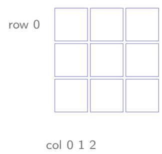
An **ArrayList** 动态数组 grows and shrinks, unlike a fixed array. It stores objects, so numbers are **wrapped** 包装 (`Integer`, `Double`).

`add(x)`	`get(i)`	`set(i,x)`	`remove(i)`
append	read	replace	delete

Traverse it with a `for` or an enhanced for-each loop: `for (int x : list)`.

4 2D arrays

2D arrays (grids)



A **2D array** 二维数组 is a grid of rows & columns: `grid[r][c]`.

Walk it with **nested loops** – outer over rows, inner over columns:

```
for r ...
  for c ...
    grid[r][c]
```

5 Search & sort

Searching & sorting

Linear search 线性查找 checks every element; **binary search** 二分查找 halves a **sorted** array

Selection 选择排序 & **insertion sort** 插入排序 order a list; **merge sort** 归并排序 is faster

Binary search needs **sorted** data and does $\sim \log_2 n$ comparisons, far fewer than linear search's n.

Recursion

A **recursive** 递归 method calls itself on a smaller input, stopping at a **base case** 基本情形.

Factorial

`factorial(n) = n * factorial(n-1)`, with `factorial(0) = 1`.
Each call shrinks *n* until the base case, then the results multiply back up. **Merge sort** 归并排序 uses the same divide-and-conquer idea.

Key points

- **Arrays** are fixed; **ArrayLists** grow & hold objects
- A **2D array** is a grid, walked with nested loops
- **Binary search** & **merge sort** beat the simple versions
- **Recursion** solves a problem via a smaller copy

Worked example – tracing recursion

Trace `factorial(4)`

Each call defers to a smaller one:

$$4 \cdot f(3) = 4 \cdot 3 \cdot f(2) = 4 \cdot 3 \cdot 2 \cdot f(1)$$

`factorial(1)` hits the base case and returns 1, so the calls unwind inward: $2 \cdot 1 = 2$, then $3 \cdot 2 = 6$, then $4 \cdot 6 = 24$.

Writing each call above its returned value is the reliable way to trace recursion – and the base case is what stops it.

Check yourself

1. `factorial(4)` returns what?
2. What happens to recursion with no base case?
3. Binary search needs the array to be …?
4. Array or ArrayList: which resizes itself?

Answers **1.** 24 **2.** it never stops – a StackOverflowError **3.** sorted **4.** ArrayList