Searching & sorting

Java Reference

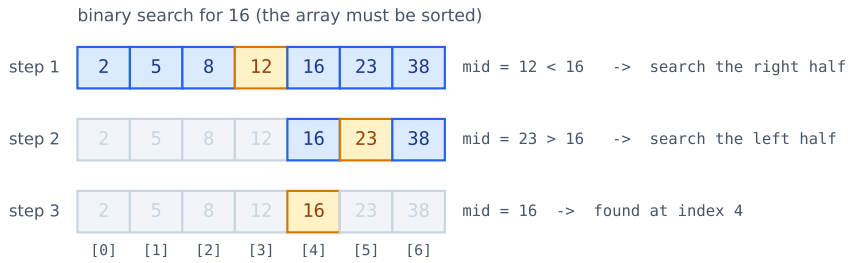
Linear & binary search

Linear search 线性查找 checks every element — works on any array. Binary search 二分查找 is much faster but needs a **sorted** 已排序 array: it looks at the middle, then throws away half each step. Both return the index, or `-1` if not found.

```
public class Main {
    public static void main(String[] args) {
        int[] a = {2, 5, 8, 12, 16, 23};      // sorted, so binary search
↪   works
        System.out.println(linear(a, 12));    // 3
        System.out.println(binary(a, 12));    // 3
        System.out.println(binary(a, 9));     // -1 (not found)
    }

    static int linear(int[] a, int target) {
        for (int i = 0; i < a.length; i++)
            if (a[i] == target) return i;
        return -1;
    }

    static int binary(int[] a, int target) {
        int lo = 0, hi = a.length - 1;
        while (lo <= hi) {
            int mid = (lo + hi) / 2;
            if (a[mid] == target) return mid;
            else if (a[mid] < target) lo = mid + 1;
            else hi = mid - 1;
        }
        return -1;
    }
}
```

11-binary-search

Binary search halves the range each step, so a sorted array is searched in O(log n)

Selection & insertion sort

Selection sort 选择排序 repeatedly finds the smallest remaining value and swaps it to the front. Insertion sort 插入排序 takes each value and slides it back into its place among the already-sorted values.

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] a = {5, 2, 9, 1, 7};
        for (int i = 0; i < a.length - 1; i++) {
            int min = i;
            for (int j = i + 1; j < a.length; j++)
                if (a[j] < a[min]) min = j;
            int t = a[min]; a[min] = a[i]; a[i] = t;   // swap into place
        }
        System.out.println(Arrays.toString(a));   // [1, 2, 5, 7, 9]
    }
}
```

```java
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] a = {5, 2, 9, 1, 7};
        for (int i = 1; i < a.length; i++) {
            int key = a[i], j = i - 1;
            while (j >= 0 && a[j] > key) {    // shift bigger values
↪   right
                a[j + 1] = a[j];
                j--;
            }
            a[j + 1] = key;                   // drop key into the gap
        }
        System.out.println(Arrays.toString(a));   // [1, 2, 5, 7, 9]
    }
}
```

How the speeds compare:

Algorithm	Time
linear search	O(n)
binary search	O(log n), sorted arrays only
selection / insertion sort	$O(n^2)$
merge sort (topic 10)	O(n log n)

Common mistakes

- Binary search only works on a **sorted** array.
- Linear search is O(n); binary search is O(log n) but needs the sort first.
- Selection and insertion sort are $O(n^2)$ — clear to learn, slow on big data.