！！！！！！！！！！！！！！！！ FINAL !!!!!!!!!!!!!!!!!!

中文名称	英文名称	平均 时间复杂度	最坏 时间复杂度	最好 时间复杂度	空间复杂度	稳定性
选择排序	Selection	n^2	n^2	n^2	1	不稳
冒泡排序	Bubble	n^2	n^2	n	1	稳
插入排序	Insertion	n^2	n^2	n	1	稳
堆排序	heap	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	1	不稳
希尔排序	Shell	$n^{1.3}$	n^2	n	1	不稳
归并排序	Merge	$n\log_2 n$	$n\log_2 n$	$n\log_2 n$	n	稳
快速排序	Quick	$n\log_2 n$	n^2	$n\log_2 n$	$\log_2 n$	不稳
桶排序	Bucket	$n+k$	n^2	n	$n+k$	稳
计数排序	Counting	$n+k$	$n+k$	$n+k$	$n+k$	稳
基数排序	Radix	$n*k$	$n*k$	$n*k$	$n+k$	稳

2023-2024 F
1. 填空题：变量类型
题目：

A _____ variable stores the actual values of primitive data types directly, while a _____ variable stores the address of an object in memory.
中文翻译：
A 类型的变量直接存储基本数据类型的实际值，而 B 类型的变量则存储对象在内存中的地址。

答案：

A: primitive（基本类型）

B: reference（引用类型）

完整句子：
A primitive variable stores the actual values of primitive data types directly, while a reference variable stores the address of an object in memory.
中文解释：
基本类型变量直接存储值，比如 int, float，而引用类型变量（如 String, Object）存储的是内存地址（引用）。

2. 填空题：Java 特性
题目：

2. _____ is a feature in Java where a subclass provides a specific implementation of a method that is already provided by its parent class, whereas _____ is a feature that allows a class to have more than one method having the same name, but with different parameter lists.
中文翻译：
_____ 是 Java 中一个特性，子类提供了父类已有方法的特定实现；而 _____ 是一个允许类中存在多个方法同名但参数不同的特性。

答案：

第一个空：Overriding（重写）

第二个空：Overloading（重载）

完整句子：
Overriding is a feature in Java where a subclass provides a specific implementation of a method that is already provided by its parent class, whereas Overloading is a feature that allows a class to have more than one method having the same name, but with different parameter lists.
中文解释：

重写（Overriding）：子类覆盖父类的方法，方法名、参数、返回类型都相同。

重载（Overloading）：方法名相同，但参数数量或类型不同。

3. 代码输出题
题目：
```
public class Test {
    public static void main(String[] args) {
        Object o1 = new Object();
        Object o2 = new Object();
        System.out.print((o1 == o2) + " and " + (o1.equals(o2)));
    }
}
```
问题： 输出结果分别是什么？（填空）

答案：

第一个空：false

第二个空：false

完整输出：
false and false

中文解释：

o1 == o2 比较的是两个对象的引用地址，它们是两个不同的对象，所以是 false。

o1.equals(o2) 在没有重写 equals 方法的情况下，Object 的默认实现和 == 等价，也是比较地址，因此也是 false。

第 4 题：接口与抽象类
题目：
4. An _____ is a class-like construct that contains only constants, abstract methods, default methods, and static methods. In many ways, an interface is similar to an abstract class, but an abstract class can contain _____.

中文翻译：
_____ 是一种类似类的结构，它只包含常量、抽象方法、默认方法和静态方法。在许多方面，接口类似于抽象类，但抽象类还可以包含 _____。

答案：
第一个空：interface

第二个空：instance variables and constructors

完整句子：
An interface is a class-like construct that contains only constants, abstract methods, default methods, and static methods. In many ways, an interface is similar to an abstract class, but an abstract class can contain instance variables and constructors.

中文解释：
接口（interface）不能有实例变量，也不能有构造方法。

抽象类(abstract class)可以有实例变量(fields)和构造方法(constructors)，这使得它更像一个“未完成的类”。

第 5 题：输出结果
题目代码：
```
public class Main {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        list.add("NewYork");
        ArrayList<String> list1 = (ArrayList<String>)(list.clone());
        list.add("Atlanta");
        list1.add("Dallas");
        System.out.println(list1);
    }
}
```
问题：
输出结果是：_____?

答案：
[NewYork, Dallas]

中文解释：
list 中添加了 "NewYork"。

然后用 clone() 创建了 list1，此时它复制了当前 list 的内容，变成了["NewYork"]。

后来 list 添加了 "Atlanta"，但是这不会影响 list1，因为 clone() 创建的是浅复制的新列表（列表对象本身不同，但元素是同一个引用）。

list1 再添加 "Dallas"，所以最终输出是 ["NewYork", "Dallas"]

🔹 1. 浅拷贝（Shallow Copy）

✅ **定义：**

浅拷贝是指 **只复制对象的引用类型字段的"地址"，而不是复制内容本身。**

📌 **特点：**

- 拷贝的是对象的**第一层结构。**
- 引用类型字段（如数组、集合、对象等）**不重新创建，只复制引用。**
- 拷贝后的对象和原对象共享**引用对象**，改一个另一个可能也变。

📦 **示例代码：**

java 复制 编辑

```java
class Person {
    String name;
    Address address; // 引用类型字段

    public Person(String name, Address address) {
        this.name = name;
        this.address = address;
    }

    public Person shallowCopy() {
        return new Person(this.name, this.address);
    }
}

class Address {
    String city;
    public Address(String city) {
        this.city = city;
    }
}
```

java 复制 编辑

```java
Person p1 = new Person("Tom", new Address("Beijing"));
Person p2 = p1.shallowCopy();

p2.address.city = "Shanghai";
System.out.println(p1.address.city); // 输出：Shanghai（被改变）
```

🔹 2. 深拷贝（Deep Copy）

✅ **定义：**

深拷贝是指 **完全复制一个对象，包括其引用的对象的内容也一并复制一份。**

📌 **特点：**

- 拷贝的是整个**对象图（所有引用的对象）**。
- 每一层引用对象都**新建一份内容**，彼此完全独立。
- 改动副本不会影响原始对象。

📦 **示例代码（简化）：**

java 复制 编辑

```java
class Person {
    String name;
    Address address;

    public Person(String name, Address address) {
        this.name = name;
        this.address = new Address(address.city); // 手动复制
    }

    public Person deepCopy() {
        return new Person(this.name, new Address(this.address.city));
    }
}
```

java 复制 编辑

```java
Person p1 = new Person("Tom", new Address("Beijing"));
Person p2 = p1.deepCopy();

p2.address.city = "Shanghai";
System.out.println(p1.address.city); // 输出：Beijing（没被改）
```

✅ 区别总结表

对比项	浅拷贝（Shallow Copy）	深拷贝（Deep Copy）
拷贝对象内容	只复制基本类型，引用类型只复制引用地址	所有字段（包括引用对象）都复制
是否共享引用对象	是	否
修改副本是否影响原对象	是（如果修改引用对象）	否
性能	拷贝快，占用内存少	拷贝慢，占用内存多
用途	轻量级复制、临时场景	安全隔离、多线程或需要数据独立性时使用

✍️ 在 Java 中如何实现？

方法	是否浅拷贝
`Object.clone()`	是（除非手动深拷贝）
`ArrayList.clone()`	是（元素不复制，只复制引用）
手动构造复制构造器	可实现深拷贝
序列化 + 反序列化	一种深拷贝技巧

第 6 题：编译错误原因

题目代码（有行号）：

```
1. public class Test {
2.         public static void main(String[] args) {
3.                 Fruit[] fruits = { new Fruit(2), new Fruit(3), new Fruit(1) };
4.                 Arrays.sort(fruits);
5.         }
6. }

class Fruit {
        private double weight;
        public Fruit(double weight) {
                this.weight = weight;
        }
}
```

问题：

这个程序会出现编译错误，因为 Fruit 类没有实现 Comparable 接口。请问是在哪一行出的问题？Line ____?

答案：

Line 4

解释：
Arrays.sort(fruits) 试图对 Fruit 类型数组排序。

排序必须依赖于元素的可比较性（Comparable 接口），但 Fruit 类没有实现 Comparable<Fruit> 接口。

所以编译器会报错：Fruit cannot be cast to Comparable。

✓ 如果想修复这个错误：
你需要让 Fruit 类实现 Comparable 接口并定义 compareTo() 方法，例如：

```
class Fruit implements Comparable<Fruit> {
    private double weight;

    public Fruit(double weight) {
        this.weight = weight;
    }

    @Override
    public int compareTo(Fruit other) {
        return Double.compare(this.weight, other.weight);
    }
}
```

第 7 题：泛型类与编译器检查
题目：
7. A _____ class or method permits you to specify allowable types of objects that the class or method can work with. If you attempt to use a class or method with an incompatible object, the _____ will detect the error.

中文翻译：
一个 _____ 类或方法允许你指定它可以使用的对象类型。如果你尝试使用不兼容的类型，_____ 会检测出错误。

✓ 答案：
第一个空：generic

第二个空：compiler

完整句子：
A generic class or method permits you to specify allowable types of objects that the class or method can work with. If you attempt to use a class or method with an incompatible object, the compiler will detect the error.

中文解释：

泛型类或方法（Generic）可以在编译时指定类型，避免类型转换错误。

编译器（Compiler）在编译期间检查类型安全，减少运行时错误。

第 8 题：泛型方法声明
题目代码：

```
public class GenericMethodDemo {
    public static void main(String[] args) {
        Integer[] integers = {1, 2, 3, 4, 5};
        String[] strings = {"London", "Paris", "NewYork", "Austin"};
        print(integers);
        print(strings);
    }

    public static _____ {
        for (int i = 0; i < list.length; i++)
            System.out.print(list[i] + " ");
        System.out.println();
    }
}
```

✓ 答案：

```
<T> void print(T[] list)
```

完整签名：

```
public static <T> void print(T[] list)
```

中文解释：

<T> 是泛型声明，代表这个方法是一个泛型方法。

T[] list 表示参数是一个元素类型为 T 的数组，T 可以是任何类型。

这样才能让一个方法同时打印 Integer[] 和 String[]。

非常棒的问题！你问的是：

> 在第八题中，为什么泛型方法写成 `<T> void print(T[] list)`，而不是 `<E>`？`T` 和 `E` 有区别吗？

✅ 答案简要：

`T` 和 `E` **在功能上是一样的，没有本质区别**，它们都是泛型类型参数的名字。
区别只是**命名习惯（convention）**不同，用哪个都可以，编译器不会因为你用了 `E` 而报错。

🧠 详细解释如下：

✅ 泛型类型参数命名的常用习惯（Java规范推荐）：

泛型参数	常用含义
`T`	Type（一般的类型）
`E`	Element（集合中的元素）
`K`	Key（键，例如 Map 的 key）
`V`	Value（值，例如 Map 的 value）
`N`	Number（数字）
`S`, `U`, `R`	额外的类型参数

第 9 题：集合类型性能对比
题目：
9. In Java collections, the _____ is efficient for retrieving elements and for adding and removing elements at the end of the list. On the other hand, a _____ is better suited for adding and removing elements at any position within the list, although locating specific elements for operations is not as efficient.

中文翻译：
在 Java 集合中，_____ 适合快速访问和在列表尾部添加/删除元素。而 _____ 更适合在列表中任意位置添加/删除元素，但定位具体元素的效率不高。

✓ 答案：
第一个空：ArrayList

第二个空：LinkedList

中文解释：

ArrayList：底层是数组，支持快速随机访问，但中间插入/删除比较慢。

LinkedList：底层是链表，插入/删除任意位置都快，但查找慢。

第 10 题：集合操作 remove()
题目条件：

list1: ["red", "yellow", "green"]

list2: ["red", "yellow", "blue"]

执行：list1.remove(list2)

❗ 注意：你这题的写法可能是 list1.removeAll(list2) 才对。因为 list.remove(Object) 只删除单个对象。我会分别解释。
✓ 正常情况 1：如果是 list1.removeAll(list2)：
作用：删除 list1 中所有在 list2 中也出现的元素

删除 "red" 和 "yellow"

list1 剩下 ["green"]

✓ 答案： ["green"]

异常情况 2：如果真的是 list1.remove(list2)（错误写法）
list2 是一个 List，不是 String，所以 list1.remove(list2) 会尝试删除整个 list2 这个对象，但失败。

list1 不变，仍是 ["red", "yellow", "green"]

建议答案：
如果你是考 removeAll：
✓ 答案：["green"]

✓ 第 11 题：栈与队列的处理顺序
题目：
11. _____ store objects that are processed in a last-in, first-out fashion.
_____ store objects that are processed in a first-in, first-out fashion.

中文翻译：
_____ 存储的对象遵循 后进先出（LIFO） 的顺序处理。
_____ 存储的对象遵循 先进先出（FIFO） 的顺序处理。

✓ 答案：
第一个空：Stack

第二个空：Queue

英文解释：
Stack → Last In, First Out (LIFO)

Queue → First In, First Out (FIFO)

中文解释：
栈（Stack）：像书堆，最后放进去的最先被取出。

队列（Queue）：像排队买票，先来先走。

✓ 第 12 题：PriorityQueue 的输出顺序
原始代码：

```
public class Test {
    public static void main(String[] args) {
        PriorityQueue<Integer> queue = new PriorityQueue<Integer>(
            Arrays.asList(20, 40, 60, 10, 50, 30)
        );
        while (!queue.isEmpty())
            System.out.print(queue.poll() + " ");
    }
}
```

✓ 答案输出：
10 20 30 40 50 60
解释：
PriorityQueue 是一个 小顶堆：最小的元素有最高优先级。
poll() 方法每次取出当前队列中最小的元素。
所以元素按升序输出。
Java 中的 PriorityQueue（对于整数类型默认情况）会按自然升序对元素进行排序。poll() 方法检索并移除此队列的头部（最小的元素）。代码使用数字 20, 40, 60, 10, 50 和 30 初始化 PriorityQueue。while 循环重复地取出并打印最小的元素，直到队列为空。

✓ 第 13 题：LinkedHashSet 的输出顺序
代码：

```
Set<String> set = new LinkedHashSet<>();
set.add("ABC");
set.add("FGH");
```

System.out.println(set);
✓ 答案输出：
[ABC, FGH]
中文解释：
LinkedHashSet 是一个有顺序的 Set，它保留了元素 插入的顺序。

所以虽然是 Set（不重复），但不像 HashSet 那样无序。

你加入的顺序是 "ABC" → "FGH"，所以输出顺序一样。

✓ 第 14 题：创建空的 HashSet
题目：
Complete the following code to create an empty hash set of integers:

Set<Integer> set = new _____;
✓ 答案：
new HashSet<>();
完整代码：
Set<Integer> set = new HashSet<>();
中文解释：
Set 是接口，不能直接实例化。

用 HashSet 实现它，创建一个空的整数集合

第 15 题：图书馆系统中最佳数据结构
题目：
In a library management system where you need to frequently check the availability and information of books with unique identifiers (e.g., bookID), the most suitable data structure to store the books in the library would be a _____.

✓ 答案：
HashMap
中文解释：
图书 ID（如 bookID）是唯一的，最适合用 key-value 结构存储。

HashMap<bookID, BookObject> 可以用 O(1) 的时间快速查找书籍信息。

✓ 第 16 题：算法性能分析方法
题目：
The _____ is a theoretical approach for analyzing the performance of an algorithm. It estimates how fast an algorithm's execution time increases as the input size increases...

✓ 答案：
Big O notation

中文解释：
Big O 表示法（大 O 符号）用于估算算法的增长率。

举例：O(n)、O(log n)、O(n²)

✓ 第 17 题：常见时间复杂度术语
题目：
An algorithm with the _____ time complexity is called a linear time algorithm, and an algorithm with the _____ time complexity is called a logarithmic algorithm.

✓ 答案：
第一空：O(n)（线性）

第二空：O(log n)（对数）

✓ 第 18 题：方法的时间复杂度分析
题目代码：
```
public static void mD(int[] m) {
    for (int i = 0; i < m.length; i++) {
        System.out.print(m[i] + "");
    }
}
```
✓ 答案：
O(n)
解释：
有一个 for 循环，运行 n 次（数组长度）。

每次执行一次打印语句，所以总操作是 n 次，时间复杂度是 O(n)。

✓ 第 19 题：最坏情况的时间复杂度
题目：
The worst-case time complexity for selection sort, insertion sort, bubble sort, and quicksort algorithms is _____.

✓ 答案：
O(n²)
英文解释：
All of Selection Sort, Insertion Sort, and Bubble Sort have a worst-case time complexity of O(n²).

QuickSort's worst case also can be O(n²), especially when the pivot selection is poor (like always picking the smallest/largest element).

中文解释：
选择、插入、冒泡 排序算法在最坏情况下都要比较接近 n² 次。

快速排序在最坏情况下也会退化为 $O(n^2)$，比如每次分割都很不均衡（例如已经排序好的数据）。

✓ 第 20 题：冒泡排序第一轮结果
题目：
Suppose a list is [12, 9, 15, 4, 20, 11]. After the first pass of bubble sort, the list becomes _____.

冒泡排序第一轮过程（两两比较并交换）：
初始：[12, 9, 15, 4, 20, 11]

比较过程如下：

12 vs 9 → 交换 → [9, 12, 15, 4, 20, 11]

12 vs 15 → 不换

15 vs 4 → 交换 → [9, 12, 4, 15, 20, 11]

15 vs 20 → 不换

20 vs 11 → 交换 → [9, 12, 4, 15, 11, 20]

✓ 答案：
[9, 12, 4, 15, 11, 20]
中文解释：
冒泡排序的第一轮会把最大值“冒泡”到最后，剩下的还要继续排。

第 21 题：最大堆的可能取值数量
题目大致结构是：

一个 最大堆（Max Heap） 的结构如下图：

```
      13
     /  \
    ?    10
   /\
  8  9
```

我们要确定 ? 的位置可以填哪些值，才能仍然满足 Max Heap 的结构。

Max Heap 规则：
每个父节点的值必须大于等于它的子节点的值。

当前结构：
根节点是 13

? 是 13 的左孩子，它的子节点是 8 和 9

10 是 13 的右孩子

所以 ? 的值必须：

≤ 13（因为它是 13 的子节点）

≥ 9（因为它是 8 和 9 的父节点）

✓ 所以，? 的合法范围是：
9 ≤ ? ≤ 13
能取的整数值有：9, 10, 11, 12, 13 → 共 5 个值

✓ 答案：
5

✓ 第 22 题：图中多条边 & 完全图定义
题目：
In a graph, if two vertices are connected by two or more edges, these edges are called _____.
A _____ is the one in which every two pairs of distinct vertices are connected by an edge.

✓ 答案：
第一空：parallel edges（平行边）

第二空：complete graph（完全图）

中文解释：
平行边（Parallel edges）：两个顶点之间有多条边。

完全图（Complete graph）：每一对不同的顶点之间都直接连接一条边。

✓ 第 23 题：图的遍历方式
题目：
There are two popular ways to traverse a graph, namely _____ and _____.

✓ 答案：
Depth-First Search（DFS）

Breadth-First Search（BFS）

中文解释：

DFS（深度优先搜索）：尽可能“走到底”，用递归或栈实现。

BFS（广度优先搜索）：一层层展开，用队列实现。

✓ 第 24 题：最小生成树 vs 最短路径
题目：
In a network of cities connected by roads of varying lengths,

if you want to connect all the cities in the most cost-effective way possible, use the _____ algorithm.

if you want to find the shortest route from one specific city to another, use the _____ algorithm.

✓ 答案：
最小生成树： Prim’s algorithm（或 Kruskal’s algorithm）

最短路径： Dijkstra’s algorithm

所以填法可以是：
Prim’s (或 Kruskal’s), Dijkstra’s
中文解释：
Prim / Kruskal：用于构造连接所有城市的最小花费网络（MST）

Dijkstra：从一个城市出发找最短路径到其它城市

✓ 第 25 题：平衡二叉搜索树的插入删除复杂度
题目：
The time complexity for inserting and deleting an element into a binary search tree are _____ and _____ respectively, assuming that the tree is balanced.

✓ 答案：
O(log n), O(log n)
中文解释：
在 平衡二叉搜索树（如 AVL 或红黑树） 中，每次查找路径长度是对数级别。

所以插入、删除的复杂度都是 O(log n)。

✓ 第 26 题：二叉树的遍历顺序
题目：
In a binary tree, a _____ traversal visits the root node first, followed by the left subtree, and then the right subtree.
Conversely, a _____ traversal visits the left subtree first, followed by the right subtree, and finally the root node.

✓ 答案：
preorder, postorder
英文解释：
Preorder Traversal (先序遍历)：Root → Left → Right

Postorder Traversal (后序遍历)：Left → Right → Root

中文解释：
先序遍历：先访问根节点，然后左子树，最后右子树。

后序遍历：先左子树，再右子树，最后访问根节点。

✓ 第 27 题：AVL 树的平衡因子（Balance Factor）
题目：
In an AVL tree, a node is said to be _____ if its balance factor is -1, and it is said to be _____ if its balance factor is +1.

✓ 答案：

right-heavy, left-heavy
英文解释：
Balance Factor = Height(left subtree) - Height(right subtree)

-1 表示 右子树高于左子树 → Right-heavy

+1 表示 左子树高于右子树 → Left-heavy

中文解释：
平衡因子 = 左子树高度 - 右子树高度

平衡因子为 -1：右边比较“重” → 叫做 right-heavy（右偏）

平衡因子为 +1：左边比较“重” → 叫做 left-heavy（左偏）

除了**右旋 (Right Rotation)**，AVL 树还有以下几种核心操作，它们主要用于在插入或删除节点后，维持树的平衡性：

1.

左旋 (Left Rotation)：

2.

1. **何时使用：** 当 AVL 树出现 **右-右 (RR) 不平衡** 时。这意味着某个节点的平衡因子为 -2，且不平衡是由其右子树的右侧插入或删除引起的。
2. **过程：** 想象一个节点 A 的右子节点 B。左旋操作会使 B 成为新的子树根，A 成为 B 的左子节点。如果 B 原本有左子节点，该节点会成为 A 的右子节点。

3. **例子：**

```
A (-2)
 \
  B (-1 或 0)
   \
    C
```

4. 左旋 A 后变为：

```
  B
 / \
A   C
```

5.

3.

左右旋 (Left-Right Rotation / LR Rotation)：

4.

1. **何时使用：** 当 AVL 树出现 **左-右 (LR) 不平衡** 时。这意味着某个节点的平衡因子为 +2，且不平衡是由其左子树的右侧插入或删除引起的。
2. **过程：** 这是一种**组合旋转**。首先对不平衡节点的**左子节点**执行**左旋**，这会将其转化为一个 LL 不平衡的情况。然后，对原始不平衡节点执行**右旋**。

3. **例子：**

```
  A (+2)
 /
B (-1)
 \
  C
```

4.

1. 对 B 进行左旋：

```
  A (+2)
 /
C
 \
```

```
  B
```

2. 对 A 进行右旋：

```
  C
 / \
B   A
```

5.

右左旋 (Right-Left Rotation / RL Rotation)：

6.

1. **何时使用：** 当 AVL 树出现 **右-左 (RL) 不平衡** 时。这意味着某个节点的平衡因子为 -2，且不平衡是由其右子树的左侧插入或删除引起的。
2. **过程：** 这也是一种**组合旋转**。首先对不平衡节点的**右子节点**执行**右旋**，这会将其转化为一个 RR 不平衡的情况。然后，对原始不平衡节点执行**左旋**。
3. **例子：**

```
A (-2)
 \
  B (+1)
 /
C
```

4.

1. 对 B 进行右旋：

```
A (-2)
 \
  C
 /
B
```

1. 对 A 进行左旋：

```
  C
 / \
A   B
```

除了这些核心的旋转操作，AVL 树还涉及：

- **插入 (Insertion):** 将新节点插入到二叉搜索树的正确位置，然后从插入点向上回溯，检查并执行必要的旋转以保持平衡。

- **删除 (Deletion):** 从二叉搜索树中删除一个节点（可能涉及将节点替换为其中序前驱或后继），然后从删除点向上回溯，检查并执行必要的旋转以保持平衡。
- **搜索 (Search):** 这是所有二叉搜索树的基本操作，用于查找树中是否存在某个特定值。它不涉及平衡操作，时间复杂度为 O(log n)。

这些操作使得 AVL 树在各种场景下都能保持高效的 O(log n) 查找、插入和删除时间复杂度。

✓ 第 29 题：处理哈希冲突的方法
题目：
_____ and _____ methods are usually taken to deal with hashing collision where two different keys are mapped to the same index in a hash table.

✓ 答案：
chaining, open addressing
中文解释：
Chaining（链地址法）：

把同一个哈希位置上的多个元素用链表存起来。

每个桶可以有多个元素，发生冲突时加入链表。

Open Addressing（开放地址法）：

冲突时，在哈希表中寻找下一个空位插入，而不是使用链表。

这包括线性探测（linear probing）、二次探测（quadratic probing）等方式。

✓ 第 30 题：开放地址法中的具体探测方式
题目：
In hash tables, _____ probing resolves collisions by sequentially checking the next available slots, while _____ probing resolves collisions by checking slots at increasing squared distances from the original hash.

✓ 答案：
linear, quadratic
中文解释：
Linear probing（线性探测）：

发生冲突后，往下一个格子 +1、+2、+3... 逐个试。

优点：实现简单；缺点：容易发生聚集（clustering）。

Quadratic probing（二次探测）：

探测间隔是平方数：+1², +2², +3²...

可以减少聚集，分布更平均。

23-24 R

第 1 题：变量类型（Variable Types）
题目：
In the following code, ‘number’ is a _____ variable as it stores the actual values of primitive data types directly, while ‘message’ is a _____ variable as it stores the address of an object in memory.

✓ 答案：
primitive, reference
英文解释：

int number = 42; → number 是基本类型（primitive type），直接存储值。

String message = "Hello"; → message 是引用类型（reference type），存储对象地址。

中文解释：

number 是基本数据类型变量，直接存储值 → primitive

message 是引用类型变量，存储对象在内存中的地址 → reference

✓ 第 2 题：继承类的概念（Superclass & Subclass）
题目：
A _____ is a class from which other classes are derived, while a _____ is a class that is derived from another class.

✓ 答案：
superclass, subclass
英文解释：

A superclass provides base functionality.

A subclass inherits and extends the superclass.

中文解释：

父类（superclass） 是其他类派生的基础类

子类（subclass） 是继承了父类的类

✓ 第 3 题：Java 中的相等性（Equality in Java）
题目：
In Java, _____ equality checks if two variables refer to the exact same object in memory, while _____ equality checks if two objects have the same state or content, as defined by the equals() method.

✓ 答案：
reference, logical
英文解释：

Reference equality uses == to compare memory addresses.

Logical equality uses .equals() to compare values or content.

中文解释：

引用相等（reference equality） 用 == 比较内存地址是否相同

逻辑相等（logical equality） 用 .equals() 方法判断内容是否相等

✓ 第 4 题：接口与抽象类（Interface vs Abstract Class）
题目：
An _____ can have instance methods that implement a default behavior, while an _____ cannot have instance methods.

✓ 答案：
abstract class, interface
英文解释：

Abstract classes can have method implementations.

Interfaces (prior to Java 8) cannot have any method body.

中文解释：

抽象类（abstract class） 可以包含已实现的方法

接口（interface）（Java 8 前）不能包含方法实现（Java 8 后可以有 default 方法）

✓ 第 5 题：ArrayList vs LinkedList
题目：
For scenarios requiring efficient random access through an index without frequent insertion or removal of elements except at the end of the list, the _____ class is more suitable. In contrast, for applications that frequently insert or delete elements at the beginning of the list, the _____ class is a better choice.

✓ 答案：

ArrayList, LinkedList
英文解释：

ArrayList supports fast indexed access.

LinkedList is efficient for frequent insertions/deletions at the start.

中文解释：

如果你需要通过索引快速访问 → 使用 ArrayList

如果你经常在列表头部插入/删除元素 → 使用 LinkedList

第 6 题：接口实现错误（Interface Implementation Error）
题目：
In the program below, a compilation error occurs at line 4 because the Fruit class does not implement the _____ interface, which is required for the Arrays.sort() method to sort the array of Fruit objects.

```
public class Test {
    public static void main(String[] args) {
        Fruit[] fruits = {new Fruit(2), new Fruit(3), new Fruit(1)};
        Arrays.sort(fruits);    // Line 4
    }
}

class Fruit {
    private double weight;
    public Fruit(double weight) {
        this.weight = weight;
    }
}
```

✓ 答案：

Comparable, Comparable

英文解释：

The Arrays.sort() method requires the objects to implement the Comparable interface to define how they should be compared for sorting.

中文解释：

Arrays.sort() 方法要求对象实现 Comparable 接口，以便定义对象如何进行排序。

✓ 第 7 题：泛型错误（Generic Type Error）
题目：
The following statement would encounter a compilation error because a generic type must be a _____ type. One way to correct the statement is to change the type parameter “int” to _____.

ArrayList<int> intList = new ArrayList<>();
✓ 答案：

reference, Integer
英文解释：

In Java, generics can only work with reference types, not primitive types. So, you should use Integer instead of int.

中文解释：

在 Java 中，泛型只能使用引用类型，不能使用基本类型。因此，应将 int 改为 Integer。

✓ 第 8 题：泛型方法（Generic Method）
题目：
The method header is left blank in the following code. Fill in the header using a generic method _____.

```
public class GenericMethodDemo {
    public static void main(String[] args) {
        Integer[] numbers = {10, 20, 30, 40, 50};
        String[] cities = {"Tokyo", "London", "Paris", "New York"};
        print(numbers);
        print(cities);
    }

    public static _________ print(Object[] list) {
        for (int i = 0; i < list.length; i++)
            System.out.print(list[i] + " ");
        System.out.println();
```

```
    }
}
```
✓ 答案：

<T> void

英文解释：

A generic method allows you to specify the type parameter <T>, which will be determined when the method is called. The void indicates that the method doesn't return anything.

中文解释：

泛型方法允许你使用 <T> 来指定类型参数，这个类型参数在方法调用时确定。void 表示该方法不返回任何值。

✓ 第 9 题：LinkedList 类型（LinkedList Type）

题目：

In the code below, 'LinkedList' is a _____ without specifying any type parameter, which implies it can hold elements of any type.

```
public class Example {
    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.add("World");
        list.add(24);
        list.add(2.71);
    }
}
```
✓ 答案：

raw type

英文解释：

A raw type refers to the use of a generic class or interface without specifying a type parameter. This can lead to unchecked warnings or errors.

中文解释：

原始类型（raw type） 是指使用泛型类或接口时没有指定类型参数，这可能会导致未经检查的警告或错误。

✓ 第 10 题：List 的特点

题目：

A list collection organizes elements in a _____ order, and permits specifying the position where an element is stored. Additionally, elements can be accessed by _____.

✓ 答案：
linear, index
英文解释：

A list is a linear data structure that maintains the order of elements.

You can access elements by their index.

中文解释：

List 是一种线性结构，元素按插入顺序排列

元素可以通过索引访问

✓ 第 11 题：比较接口
题目：
The _____ interface defines the natural ordering of objects of each class that implements it, whereas the _____ interface defines a separate class to compare objects of another class.

✓ 答案：
Comparable, Comparator
英文解释：

Comparable is implemented by the class itself to define natural ordering.

Comparator is a separate class used to define custom ordering.

中文解释：

Comparable 接口由类自身实现，用于定义自然排序

Comparator 是一个外部比较器类，用于自定义比较规则

✓ 第 12 题：Queue 的方法
题目：
The _____ method in the Queue interface retrieves and removes the head of this queue, or null if this queue is empty. The _____ method retrieves and removes the head and throws an exception if the queue is empty.

✓ 答案：
poll, remove
英文解释：

poll() returns null if the queue is empty.

remove() throws NoSuchElementException if the queue is empty.

中文解释：

poll()：获取并删除队首元素，如果队列为空则返回 null

remove()：获取并删除队首元素，如果队列为空则抛出异常

✓ 第 13 题：addAll 方法效果
题目：
Suppose list1 is [1, 3, 5] and list2 is [2, 4, 6]. After list1.addAll(list2), list1 is _____.

✓ 答案：
[1, 3, 5, 2, 4, 6]
英文解释：

addAll() appends all elements from one list to another.

中文解释：

addAll() 会将一个列表中的所有元素追加到另一个列表末尾 → 合并成 [1, 3, 5, 2, 4, 6]

✓ 第 14 题：Map 中的 key-value 替换
题目：
In the following code, the key "101" will correspond to the value “_____” after all the operations.

```
Map<String, String> map = new HashMap<>();
map.put("101", "Alice");
map.put("102", "Bob");
map.put("101", "Charlie");
```

✓ 答案：
Charlie
英文解释：

Keys in a map must be unique; later values overwrite earlier ones.

中文解释：

Map 中键必须唯一，重复插入相同 key 会覆盖旧值 → "101" 对应 "Charlie"

✓ 第 15 题：获取 HashMap 中的值
题目：

The output of the following code is _____.

```
HashMap<String, Integer> map = new HashMap<>();
map.put("One", 1);
map.put("Two", 2);
map.put("Three", 3);
System.out.println(map.get("Two"));
```

✓ 答案：

2

英文解释：

map.get("Two") returns the value associated with the key "Two".

中文解释：

通过 key 获取对应的值 → "Two" 对应的 value 是 2

✓ 第 16 题：选择排序（Selection Sort）

题目：

In the selection sort algorithm, the _____ element in the array is identified and exchanged with the _____ element in each iteration.

✓ 答案：

smallest, first unsorted

英文解释：

Selection sort finds the smallest element from the unsorted part and swaps it with the first unsorted element.

中文解释：

每次选择最小元素，与未排序部分的第一个元素交换位置

✓ 第 17 题：算法复杂度分类

题目：

An algorithm with the
O(n) time complexity is called a _____ algorithm, and an algorithm with the
O(logn) time complexity is called a _____ algorithm.

✓ 答案：

linear, logarithmic

英文解释：

O(n): The algorithm's runtime grows linearly with the input size → Linear algorithm.
O(logn): The algorithm's runtime grows logarithmically with the input size → Logarithmic algorithm.

中文解释：

线性算法（Linear Algorithm）：运行时间与输入规模成正比。

对数算法（Logarithmic Algorithm）：运行时间与输入规模的对数成正比。

✓ 第 18 题：解决问题的方法
题目：
_____ is a method for solving complex problems by breaking them down into simpler subproblems where these subproblems overlap, unlike in _____ method where subproblems are more independent.

✓ 答案：
Dynamic Programming, Divide and Conquer
英文解释：

Dynamic Programming: Overlapping subproblems are solved optimally using memoization or tabulation.

Divide and Conquer: Subproblems are independent, solved separately, and their solutions are combined.

中文解释：

动态规划（Dynamic Programming）：将问题分解为重叠的子问题，使用记忆化或表格法优化解决。

分治法（Divide and Conquer）：将问题分解为互相独立的子问题，各自解决后再合并。

✓ 第 19 题：时间复杂度分析
题目：
Use the Big-O notation to estimate the time complexity of the following method:

```
public static void mD(int[] m) {
    for (int i = 0; i < m.length; i++) {
        for (int j = 0; j < m.length; j++) {
            System.out.print(m[i] + "");
        }
    }
}
```

✓ 答案：

O(n^2)
英文解释：

There are two nested loops, each iterating over the array of size n. Thus, the total number of iterations is
n×n, leading to O(n^ 2) time complexity.

中文解释：

两个嵌套循环，每个循环遍历数组 n 次，总共运行 n×n 次，因此时间复杂度为 O(n 2)。

✓ 第 20 题：冒泡排序第二轮结果
题目：
Suppose a list is [12, 9, 15, 4]. After the second pass of bubble sort, the list becomes _____.

✓ 答案：
[9, 12, 4, 15]
英文解释：

First pass: The largest element 15 is moved to the end → [9, 12, 4, 15].

Second pass: The second-largest element 12 is moved before 15 → [9, 12, 4, 15]. The smaller elements start shifting toward the beginning.

中文解释：

第一轮：最大值 15 移到末尾 → [9, 12, 4, 15]。

第二轮：次大值 12 移动到正确位置 → [9, 12, 4, 15]，较小的元素逐渐向前移动。

✓ 第 22 题：完全图与不完全图
题目：
A _____ graph is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge, while in an _____ graph, not every pair of vertices is connected by an edge.

✓ 答案：
complete, incomplete
英文解释：

A complete graph has all possible edges between vertices.

An incomplete graph is missing at least one edge between a pair of vertices.

中文解释：

完全图：任意两个不同顶点之间都存在一条边。

不完全图：存在至少一对顶点之间没有边。

✓ 第 23 题：DFS 和 BFS

题目：

_____ is a graph traversal method that visits all the vertices of a graph as deeply as possible before backtracking, while _____ visits all the vertices of a graph level by level.

✓ 答案：

Depth-First Search (DFS), Breadth-First Search (BFS)

英文解释：

DFS explores as far as possible along each branch before backtracking.

BFS explores all neighbors at the current depth before moving deeper.

中文解释：

DFS（深度优先搜索）：尽可能深入遍历每个分支，再回溯。

BFS（广度优先搜索）：先访问当前层的所有邻居，再进入下一层。

✓ 第 24 题：Prim 与 Dijkstra

题目：

_____ is an algorithm in graph theory where the goal is to connect all vertices with the least total weight, while _____ is an algorithm where the goal is to find the path with the least total weight between two vertices.

✓ 答案：

Prim's, Dijkstra's

英文解释：

Prim's algorithm builds a Minimum Spanning Tree (MST).

Dijkstra's algorithm finds the shortest path from one vertex to another.

中文解释：

Prim 算法：用于构建最小生成树，连接所有顶点使总权值最小。

Dijkstra 算法：用于找出两个顶点之间最短路径。

✓ 第 25 题：前序与中序遍历

题目：

In a binary tree, the _____ traversal visits the root node before its children, while the _____ traversal visits the left subtree, then the root, and finally the right subtree.

✓ 答案：

preorder, inorder

英文解释：

Preorder: Root → Left → Right

Inorder: Left → Root → Right

中文解释：

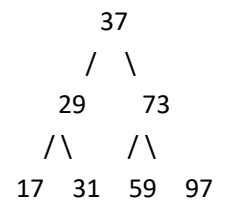
前序遍历：先访问根节点，再访问左右子树。

中序遍历：先访问左子树，再访问根节点，最后访问右子树。

问题 26：树的遍历 (Tree Traversal)

给定以下树的后序遍历和广度优先遍历分别是 _____ 和 _____。

给定的树结构：



1. 后序遍历 (Postorder Traversal)

定义： 后序遍历的顺序是：访问左子树 -> 访问右子树 -> 访问根节点。

我们递归地应用这个规则：

节点 37 (根节点):

访问左子树 (根节点 29):

节点 29:

访问左子树 (根节点 17):

节点 17: 这是一个叶子节点。没有左子节点，没有右子节点，然后访问 17。

输出：17

访问右子树 (根节点 31):

节点 31: 这是一个叶子节点。没有左子节点，没有右子节点，然后访问 31。

输出：31

然后访问 29。
输出：29
访问右子树 (根节点 73):
节点 73:
访问左子树 (根节点 59):
节点 59: 这是一个叶子节点。没有左子节点，没有右子节点，然后访问 59。
输出：59
访问右子树 (根节点 97):
节点 97: 这是一个叶子节点。没有左子节点，没有右子节点，然后访问 97。
输出：97
然后访问 73。
输出：73
然后访问 37。
输出：37
按顺序组合所有输出： 17, 31, 29, 59, 97, 73, 37

2. 广度优先遍历 (Breadth-First Traversal, BFS) / 层次遍历 (Level Order Traversal)
定义： 广度优先遍历是按层级访问节点，每层从左到右。我们通常使用队列来实现广度优先遍历。

第 0 层： 从根节点开始。

队列：[37]
输出：37
37 出队。将 37 的子节点（29, 73）入队。
队列：[29, 73]
第 1 层：

29 出队。将 29 的子节点（17, 31）入队。
输出：29
队列：[73, 17, 31]
73 出队。将 73 的子节点（59, 97）入队。
输出：73
队列：[17, 31, 59, 97]
第 2 层：

17 出队。没有子节点。
输出：17
队列：[31, 59, 97]
31 出队。没有子节点。
输出：31
队列：[59, 97]
59 出队。没有子节点。
输出：59

队列：[97]
97 出队。没有子节点。
输出：97
队列：[] (空了)
按顺序组合所有输出： 37, 29, 73, 17, 31, 59, 97

最终答案 (Final Answer)
给定树的后序遍历是 17, 31, 29, 59, 97, 73, 37，广度优先遍历是 37, 29, 73, 17, 31, 59, 97。

✓ 第 27 题：AVL 树的失衡类型
题目：
In an AVL tree, a _____ imbalance occurs when a node is inserted into the right subtree of the right child of A, while a _____ imbalance occurs when a node is inserted into the left subtree of the left child of A.

✓ 答案：
right-right (RR), left-left (LL)
英文解释：

RR imbalance happens when insertion occurs in the right subtree of the right child — resolved by a left rotation.

LL imbalance happens when insertion occurs in the left subtree of the left child — resolved by a right rotation.

中文解释：

右右失衡（RR）：当新节点插入某节点的右子节点的右子树中，需做一次左旋。

左左失衡（LL）：当新节点插入某节点的左子节点的左子树中，需做一次右旋。

✓ 第 29 题：哈希冲突与解决方案
题目：
Collisions occur when two different keys produce the same _____. Besides open addressing, _____, where each hash table slot maintains a linked list of elements mapping to the same slot, is the other way to solve collisions.

✓ 答案：

hash value, chaining

英文解释：

A collision happens when multiple keys hash to the same index.

Chaining solves this by storing all colliding keys in a linked list at that index.

中文解释：

当不同的键计算出相同的哈希值时，称为哈希冲突。

除了开放地址法，**链地址法（chaining）**是另一种解决方式：每个槽位存储一个链表，链表中保存哈希到该位置的所有元素。

✓ 第 30 题：双重散列（Double Hashing）

题目：

_____ hashing uses a secondary hash function h′ on the keys to determine the increments, aiming to avoid the _____ problem.

✓ 答案：

Double, clustering

英文解释：

Double hashing applies a second hash function to calculate the step size.

This avoids clustering, where multiple keys accumulate in adjacent slots.

中文解释：

双重散列法使用两个哈希函数，通过第二个函数决定探测步长，避免集中冲突。

聚集问题（clustering）是指冲突元素集中在某区域，导致查找效率降低。

总结：

Java 知识结构图

```
├── 基础语法与数据类型
|    ├── 原始数据类型 & 引用类型
|    ├── 变量类型（基本类型变量 vs 引用类型变量）
|    ├── 方法重载与重写
├── 面向对象编程
|    ├── 类与对象
|    ├── 继承与多态
|    |    ├── extends、super 关键字
|    |    ├── 重写（Override）
|    |    └── 方法重载（Overload）
|    ├── 抽象类 vs 接口
|    |    ├── 抽象方法与默认方法
|    |    ├── 接口的特性（只是声明，Java 8 后有 default 和 static 方法）
|    └── 封装与访问修饰符
├── 泛型（Generics）
|    ├── 泛型类与泛型方法
|    ├── 类型擦除与通配符
|    └── 常见使用错误（如：ArrayList<int>非法，应使用包装类型）
├── 集合框架（Collections）
|    ├── List 接口
|    |    ├── ArrayList（基于动态数组，随机访问快，尾部增删快）
|    |    └── LinkedList（链表结构，任意位置增删高效）
|    ├── Set 接口
|    |    ├── HashSet（无序，基于哈希表）
|    |    └── LinkedHashSet（有序，维护插入顺序）
|    ├── Map 接口
|    |    └── HashMap（键值对映射，键唯一）
|    ├── Queue 接口
|    |    ├── PriorityQueue（优先队列，堆实现）
|    |    └── 常用方法：poll(), remove()
|    └── 迭代器和 for-each 循环
├── 哈希和冲突解决
|    ├── 哈希函数与哈希码
|    ├── 开放地址法（线性探测，平方探测）
|    └── 链地址法（链表冲突解决）
├── 算法复杂度与分析
|    ├── Big-O 表示法（时间复杂度）
|    ├── 常见复杂度：O(1), O(log n), O(n), O(n log n), O(n^2)等
|    ├── 典型排序算法
|    |    ├── 选择排序、冒泡排序、插入排序（O(n^2)）
|    |    └── 快速排序（平均 O(n log n)，最坏 O(n^2)）
|    └── 动态规划与分治法区别
```

- 数据结构
  - 栈（LIFO）与队列（FIFO）
  - 树结构
    - 二叉树遍历（前序、中序、后序与层序）
    - 二叉搜索树
    - 平衡树（AVL 树，自平衡，旋转操作）
  - 图结构
    - 有向无向图
    - 简单图与多重图
    - 深度优先搜索（DFS）和广度优先搜索（BFS）
    - 最小生成树算法（Kruskal，Prim）
    - 最短路径算法（Dijkstra）

2. 易错点总结笔记

知识点	易错点与考点	说明及建议
变量类型	混淆基本类型变量与引用类型变量的存储方式	基本类型存储实际值，引用类型存储地址
方法重载与重写	混淆override和overload的含义	重写是子类对父类方法的改写，重载是同一类中方法多态
接口与抽象类	接口能否有实例方法、抽象类能否有默认实现	Java 8接口允许default和static方法，抽象类可有具体方法
泛型类型	不能用原始类型参数化，如ArrayList<int>	泛型必须用引用类型，如ArrayList<Integer>
集合的选择	ArrayList vs LinkedList的应用场景理解	ArrayList随机访问快，LinkedList插入删除快
集合接口的常用方法	clone方法浅拷贝的理解	clone浅拷贝，修改克隆后集合会独立于原集合
哈希冲突解决技术	区分链地址法和开放寻址法	链地址法在槽内存链表，开放寻址线性或平方探测找到空槽
排序与复杂度	快速排序最坏情况时间复杂度为O(n^2)	需要区分平均和最坏时间复杂度
Big-O表示法	错误理解算法复杂度和实际运行时间	复杂度是增长趋势估计，不是实际耗时
栈与队列	混淆先进后出和先进先出	栈是LIFO，队列是FIFO
二叉树遍历	前序、中序、后序、层序的访问顺序	理解定义，方便代码实现及问题解答
AVL树旋转	输入数据导致哪种旋转：单旋转还是双旋转等	根据失衡节点和平衡因子判断旋转类型
图的基本概念	简单图与多重图概念混淆	简单图边唯一，多重图边可重复
图遍历	DFS和BFS的应用场景及行为	DFS深度优先，BFS层次遍历
最短路径和最小生成树算法	Kruskal与Prim的区别	Kruskal适合边集，Prim适合邻接矩阵或图邻接表现

易错点总结笔记

1. `==` vs `.equals()`:
   - `==` 比较的是地址（引用）；`.equals()` 比较的是内容（重写后）。
2. 浅拷贝 vs 深拷贝:
   - 浅拷贝只复制引用；深拷贝复制对象本身。
3. 方法重载 vs 方法重写:
   - 重载是同类中方法名相同参数不同；重写是子类对父类方法的覆盖。
4. 泛型符号 `<T>` vs `<E>`:
   - 没有区别，习惯上 `<E>` 用于集合，`<T>` 用于通用类型。
5. 集合类选择:
   - ArrayList 查询快，LinkedList 插入删除快；HashMap 查找快。
6. Hash 结构冲突:
   - 解决方式：开放地址法（线性探测、平方探测）、链地址法。
7. 遍历顺序:
   - LinkedHashSet 保留插入顺序；HashSet 无序；TreeSet 有序。
8. Big-O 时间复杂度:
   - 排序（冒泡/选择/插入）：O(n^2)
   - 快速排序：平均 O(n log n)，最坏 O(n^2)
   - 插入/删除到 BST：O(log n)（前提是平衡）
9. Graph遍历:
   - DFS 用递归或栈实现；BFS 用队列。
10. 优先队列输出顺序:
    - 默认小顶堆，输出从小到大排序。

易错点总结笔记

在您提供的题目中，以下是几个常见的易错点和需要特别注意的地方：

基本类型与引用类型的区别:

基本类型变量直接存储值（如 int number = 42; number 存储 42）。
引用类型变量存储的是对象的内存地址（如 String message = "Hello"; message 存储的是 "Hello" 字符串对象在内存中的地址）。
在判断相等时，== 用于比较基本类型的值，或比较引用类型变量是否指向同一个对象（即内存地址）。
equals() 方法（通常在引用类型中使用）用于比较对象的内容是否相等。对于 Object 类的 equals() 默认行为与 == 相同，但许多类（如 String, ArrayList 等）都重写了 equals() 方法来比较内容。
方法重写 (Overriding) 与 方法重载 (Overloading):

重写 (Overriding)：子类提供父类已有的方法的特定实现（相同的方法签名，不同的实现）。

重载 (Overloading)：同一个类中，有多个同名方法，但它们的参数列表不同。
抽象类 (Abstract Class) 与 接口 (Interface) 的区别:

抽象类可以包含：抽象方法、非抽象方法（带实现）、构造器、成员变量（包括实例变量）。一个类只能继承一个抽象类。
接口只能包含：常量（public static final）、抽象方法（public abstract）、Java 8 后的 default 方法和 static 方法。一个类可以实现多个接口。
关键差异: 抽象类可以有实例方法（有实现的行为），而接口不能有实例变量和实例方法（在 Java 8 之前）。
集合框架类的选择:

ArrayList: 基于数组实现，适合随机访问元素（通过索引），在列表末尾添加/删除高效。但在列表中间插入或删除元素效率较低，因为需要移动大量元素。
LinkedList: 基于链表实现，适合在列表的任何位置频繁插入或删除元素，因为只需要修改指针。但随机访问效率较低，需要从头开始遍历。
HashSet: 基于哈希表实现，不保证元素顺序，不允许重复元素，查找、添加、删除操作平均时间复杂度为 O(1)。
LinkedHashSet: 保持元素的插入顺序。
PriorityQueue: 基于堆实现，元素按照自然顺序或自定义比较器排序，poll() 方法总是移除优先级最高的元素。
HashMap: 存储键值对，通过键的哈希值快速查找值。
泛型 (Generics) 的使用:

类型参数必须是引用类型，不能是基本类型。例如，ArrayList<int> 是错误的，应该用 ArrayList<Integer>。
Comparable 和 Comparator 接口用于定义对象的比较逻辑。Comparable 定义“自然排序”，Comparator 定义“自定义排序”。
排序算法时间复杂度:

O(n^2)：选择排序、插入排序、冒泡排序（最坏情况）。
O(n log n)：快速排序（平均情况），归并排序，堆排序。
O(n)：线性时间算法。
O(log n)：对数时间算法（如二分查找）。
哈希冲突解决:

链式处理 (Chaining)：在每个哈希表槽位维护一个链表，存储所有哈希到该槽位的元素。
开放寻址 (Open Addressing)：
线性探测 (Linear Probing)：顺序查找下一个空槽。
二次探测 (Quadratic Probing)：以平方递增的距离查找空槽。
双重散列 (Double Hashing)：使用第二个哈希函数来决定探测步长。
哈希冲突产生的原因: 不同的键经过哈希函数计算后得到相同的索引。
图遍历算法:

深度优先搜索 (DFS)：尽可能深地探索每个分支。

广度优先搜索 (BFS)：层层遍历。
图算法应用:

最小生成树 (MST)（如 Prim 算法）：连接所有顶点且总权重最小。
最短路径 (Shortest Path)（如 Dijkstra 算法）：找到两点间权重最小的路径。
树遍历 (Binary Tree Traversal):

前序遍历 (Preorder): 根 -> 左 -> 右
中序遍历 (Inorder): 左 -> 根 -> 右 (二叉搜索树中序遍历结果为有序序列)
后序遍历 (Postorder): 左 -> 右 -> 根
广度优先遍历 (BFS): 层序遍历

Targeted Practice Questions（专项练习题）

Part 1: Fill-in-the-Blanks / Short Answer Questions（填空题 / 简答题）

Level 1 (Basic)

1. A __primitive____ variable stores the actual values of primitive data types directly, while a __object（reference）____ variable stores the address of an object in memory. (Based on Q1, Set 1 & Set 2)
    1. *Hint: Think about* `int` *vs* `String`*.*
2. In Java, __inheritance（method overriding）____ is a feature where a subclass provides a specific implementation of a method that is already provided by its parent class. (Based on Q2, Set 1)
3. The `ArrayList` class is efficient for retrieving elements by __?（index）____ and for adding/removing elements at the __end____ of the list. (Based on Q5, Set 2 & Q9, Set 1)
4. In a graph, if two vertices are connected by two or more edges, these edges are called __parallel（multiple edges）____. (Based on Q22, Set 1)
5. An algorithm with O(N) time complexity is called a __linear____ time algorithm. (Based on Q17, Set 1 & Set 2)

Level 2 (Intermediate)

1. `Objects o1 = new Object(); Objects o2 = new Object(); System.out.print((o1 == o2) + " and " + (o1.equals(o2)));` The output of this code snippet will be __False____ and __False____. Explain why. (Based on Q3, Set 1)
2. An __abstract class____ can have instance methods that implement default behavior, while an __interface____ cannot have instance methods (prior to Java 8's `default` methods). (Based on Q4, Set 1 & Q4, Set 2)
3. The ___comparable___ interface defines the natural ordering of objects for classes that implement it, whereas the ___comparator___ interface defines a separate class to compare objects of another class. (Based on Q11, Set 2)
4. Suppose that list1 is a list that contains the strings "red", "yellow", and "green", and that list2 is another list that contains the strings "red", "yellow", and "blue". After executing `list1.remove(list2)`, `list1` contains __green["red", "yellow", "green"]____. (Based on Q10, Set 1)
    1. *Hint: Understand the behavior of* `removeAll()` *or* `remove()` *with collections.*

5. In Java collections, __stack____ store objects that are processed in a last-in, first-out fashion. __queue____ store objects that are processed in a first-in, first-out fashion. (Based on Q11, Set 1)
6. The worst-case time complexity for selection sort, insertion sort, and bubble sort algorithms is __n2____. (Based on Q19, Set 1)
7. In a binary tree, a __preorder____ traversal visits the root node first, followed by the left subtree, and then the right subtree. Conversely, a __postorder____ traversal visits the left subtree first, followed by the right subtree, and finally the root node. (Based on Q26, Set 1 & Q25, Set 2)
8. Given an original AVL tree, after adding an element that causes a right-left (RL) imbalance, a ___LL（double left (RL) then right (LR)）___ rotation would typically be performed to rebalance the tree. (Based on Q28, Set 1 & Set 2)
9. In hash tables, ___linear___ probing resolves collisions by sequentially checking the next available slots, while __quadratic____ probing resolves collisions by checking slots at increasing squared distances from the original hash. (Based on Q29, Set 1)
10. If a `Map<String, String>` named `myMap` has `myMap.put("key1", "valueA");` followed by `myMap.put("key1", "valueB");`, then `myMap.get("key1")` will return __valueA（B）____. (Based on Q14, Set 2)

Level 3 (Challenging)

1. Explain the primary advantage of using a `PriorityQueue` over a regular `LinkedList` when implementing a task scheduler where tasks are prioritized by urgency. (Based on Q12, Set 1)
2. Describe the main difference between method overloading and method overriding in Java, and provide a simple example for each. (Based on Q2, Set 1 & Q2, Set 2)
3. You are given a list of `N` elements. What is the Big O notation for an algorithm that performs a constant number of operations for each element in the list? If this algorithm is then applied to `N` nested loops, what would be its Big O notation? (Based on Q18, Set 1 & Q19, Set 2)
4. Consider a scenario where you need to store student records, each identified by a unique student ID. You need to frequently perform lookups by ID and also iterate through all student records in their insertion order. Which two Java collection classes would you combine or consider using for optimal performance in this scenario, and why? (Based on Q15, Set 1 & Q6, Set 2)
5. In the context of graph traversal, distinguish between Depth-First Search (DFS) and Breadth-First Search (BFS) in terms of how they explore a graph's vertices. (Based on Q23, Set 1 & Q23, Set 2)

Knowledge Point Summary (知识点总结)

This section covers the core concepts and specific topics tested in your provided two sets of questions.

1. Java Fundamentals (Java 基础)

- **Primitive vs. Reference Types**: How they store data in memory (`int` vs `String`).
    - **重点**: 理解值类型和引用类型在内存中的存储方式和含义。
- **Object Equality**: == vs. `equals()` for object comparison.
    - **重点**: 区分引用相等和内容相等，以及何时重写 `equals()`。
- **OOP Concepts (面向对象编程)**:
    - **Inheritance (继承)**: Parent class, subclass.
        - **重点**: 理解代码复用和层次结构。
    - **Polymorphism (多态)**: Method Overriding (方法重写) and Method Overloading (方法重载).
        - **重点**: 区分重写（相同签名，不同实现）和重载（相同方法名，不同参数列表）。
    - **Abstract Classes vs. Interfaces (抽象类与接口)**: Key differences in their structure (instance methods, constants, default methods) and usage.
        - **重点**: 掌握两者在设计上的目的和能力差异，特别是 Java 8 引入 `default` 方法后的变化。

2. Generics (泛型)

- **Generic Classes and Methods**: How to define and use them.
    - **重点**: 提高代码的类型安全性和重用性。
- **Type Parameter Constraints**: `E extends Comparable<E>`, `? super E`.
    - **重点**: 理解泛型限制的用法，如 `Comparable` 和 `Comparator`。
- **Type Erasure**: Why primitive types cannot be used as type arguments (e.g., `ArrayList<int>` is invalid).
    - **重点**: 知道泛型在运行时会被擦除为原始类型。
- `Comparable` **vs.** `Comparator`: Natural ordering vs. custom ordering.
    - **重点**: 掌握两者定义对象比较逻辑的区别和应用场景。

3. Java Collections Framework (Java 集合框架)

- `List` **Interface**:
    - `ArrayList`: Array-based, efficient random access, efficient add/remove at end. Inefficient add/remove in middle.
    - `LinkedList`: Linked-list based, efficient add/remove at any position. Inefficient random access.
    - **重点**: 理解两种 `List` 实现的底层数据结构和各自的性能特点。
- `Set` **Interface**:
    - `HashSet`: Hash-based, no guaranteed order, no duplicates. Fast operations (average O(1)).
    - `LinkedHashSet`: Maintains insertion order.
    - **重点**: 掌握 `Set` 的无重复性特点和 `HashSet` 的高效性，以及 `LinkedHashSet` 的顺序保持能力。
- `Queue` **Interface**:
    - `PriorityQueue`: Heap-based, elements ordered by natural ordering or `Comparator`. `poll()` retrieves highest priority.
    - `Queue` **methods**: `poll()`, `peek()`, `offer()`.
    - **重点**: 理解 `PriorityQueue` 的工作原理（堆），以及 `Queue` 接口的先进先出（FIFO）特性和常用方法。
- `Map` **Interface**:
    - `HashMap`: Stores key-value pairs, hash-based, fast lookups (average O(1)). Keys must be unique.
    - **重点**: 掌握 `Map` 的键值对存储方式和 `HashMap` 的高效查找。
- **Collection Operations**: `add()`, `remove()`, `addAll()`, `clone()`.
    - **重点**: 熟悉常用集合操作的行为，特别是 `remove(Collection)` 的语义。

4. Algorithms and Data Structures (算法与数据结构)

- **Arrays (数组)**: Basic data structure.
- **Sorting Algorithms (排序算法)**:
    - **Bubble Sort (冒泡排序)**
    - **Selection Sort (选择排序)**
    - **Insertion Sort (插入排序)**

- **Quick Sort (快速排序)**
- **重点**: 理解这些排序算法的基本思想和过程，以及它们的平均和最坏时间复杂度。

- **Time Complexity Analysis (时间复杂度分析)**:
  - **Big O Notation (大 O 符号)**: O(1), O(logN), O(N), O(NlogN), O(N2).
  - **重点**: 掌握如何分析简单算法的时间复杂度，并理解不同复杂度等级的含义。

- **Heap (堆)**: Max Heap (最大堆).
  - **重点**: 理解堆的结构和性质（父节点大于/小于子节点）。

- **Graphs (图)**:
  - **Concepts**: Vertices (顶点), Edges (边), Multiple Edges (多重边), Complete Graph (完全图), Connected Graph (连通图).
  - **Traversal (遍历)**: Depth-First Search (DFS) (深度优先搜索), Breadth-First Search (BFS) (广度优先搜索).
    - **重点**: 区分 DFS 和 BFS 的遍历顺序。
  - **Algorithms**: Minimum Spanning Tree (MST - 最小生成树 - Prim's Algorithm), Shortest Path (最短路径 - Dijkstra's Algorithm).
    - **重点**: 理解 Prim 和 Dijkstra 算法的目的和应用场景。

- **Trees (树)**:
  - **Binary Tree (二叉树)**
  - **Binary Search Tree (BST - 二叉搜索树)**: Insertion, Deletion time complexity (balanced).
    - **重点**: 掌握 BST 的特性以及在平衡树情况下的基本操作时间复杂度。
  - **AVL Tree (AVL 树)**: Balance Factor (平衡因子), Rotations (旋转 - LL, RR, LR, RL).
    - **重点**: 理解 AVL 树的平衡条件和各种旋转操作以保持平衡。
- **Hash Tables (哈希表)**:
  - **Hashing Collision (哈希冲突)**: When different keys map to the same index.
  - **Collision Resolution (冲突解决)**:
    - **Chaining (链式处理)**: Each slot maintains a linked list.
    - **Open Addressing (开放寻址)**: Linear Probing (线性探测), Quadratic Probing (二次探测), Double Hashing (双重散列).

- **重点**: 掌握哈希冲突的原因以及链式处理和开放寻址（及其变种）的工作原理。

Answers and Explanations (答案与解析)

Part 1: Fill-in-the-Blanks / Short Answer Questions (填空题 / 简答题) Answers

1.

primitive, reference

2.

- **中文解析**: `number` 是 `int` 类型，属于基本数据类型，直接存储值。`message` 是 `String` 类型，属于引用数据类型，存储的是字符串对象在内存中的地址。
- **English Explanation**: `number` is an `int`, which is a primitive data type, storing its value directly. `message` is a `String`, which is a reference data type, storing the memory address of the string object.

3.

method overriding (方法重写)

4.

- **中文解析**: 方法重写是子类对父类中已经存在的方法提供自己的特定实现，方法签名保持不变。
- **English Explanation**: Method overriding is when a subclass provides its specific implementation for a method that is already defined in its parent class, keeping the same method signature.

5.

index, **end** (索引, 末尾)

6.

- **中文解析**: `ArrayList` 底层是数组，通过索引访问元素非常高效。在列表末尾添加或删除元素也很快，因为不需要移动其他元素。

- **English Explanation**: `ArrayList` is backed by an array, making element retrieval by index very efficient. Adding or removing elements at the end of the list is also fast because it doesn't require shifting other elements.

7.

multiple edges (多重边)

8.

- **中文解析**: 在图中，如果两个顶点之间有多于一条边相连，这些边被称为多重边。
- **English Explanation**: In graph theory, if two vertices are connected by two or more edges, these edges are called multiple edges.

9.

linear (线性)

10.

- **中文解析**: O(N) 表示算法的执行时间与输入规模 N 成线性关系。
- **English Explanation**: O(N) indicates that the algorithm's execution time grows linearly with the input size N.

11.

false, **false**

12.

- **中文解析**: `o1` 和 `o2` 是通过 `new Object()` 创建的两个不同的 `Object` 实例，它们在内存中有不同的地址，所以 `o1 == o2`（比较引用地址）为 `false`。`Object` 类的 `equals()` 方法默认行为与 `==` 相同，因此 `o1.equals(o2)` 也为 `false`。
- **English Explanation**: `o1` and `o2` are two distinct `Object` instances created with `new Object()`, meaning they reside at different memory addresses. Thus, `o1 == o2` (comparing references) is `false`. The default `equals()` method in the `Object` class behaves identically to ==, so `o1.equals(o2)` is also `false`.

13.

abstract class (抽象类), **interface** (接口)

14.

- **中文解析**: 抽象类可以包含带有具体实现的实例方法，而接口在 Java 8 引入 `default` 方法之前，只能包含抽象方法（不带实现）。
- **English Explanation**: An abstract class can have instance methods with concrete implementations, whereas an interface (before Java 8's `default` methods) could only contain abstract methods without implementations.

15.

Comparable, **Comparator**

16.

- **中文解析**: `Comparable` 接口（例如 `String` 或 `Integer` 类实现）定义了对象的“自然排序”。`Comparator` 接口则提供了一种分离的机制，用于定义或修改另一个类的对象的排序逻辑，通常通过匿名内部类或 Lambda 表达式实现。
- **English Explanation**: The `Comparable` interface (e.g., implemented by `String` or `Integer` classes) defines the "natural ordering" of objects. The `Comparator` interface, on the other hand, provides a separate mechanism to define or modify the sorting logic for objects of another class, often implemented via anonymous inner classes or Lambda expressions.

17.

["red", "yellow", "green"] (or elements that were in list1 and not in list2, which is an empty set of elements)

18.

- **中文解析**: 题目中的 `list1.remove(list2)` 语法是错误的，`remove()` 方法只能移除单个元素或一个集合中**存在于该集合的**元素。正确的移除一个集合中所有与另一个集合共有元素的方法是 `list1.removeAll(list2)`。如果按照 `list1.removeAll(list2)` 来理解，`list1` 将会变成 `[]`。但是，如果题目字面理解为 `list1.remove(list2)`，则 `list2` 对象本身不会被找到并从 `list1` 中移除（除非 `list1` 恰好包含了 `list2` 这个对象），所以 `list1` 不变，仍然是 `["red", "yellow", "green"]`。考虑到这是考卷，很可能是想考 `removeAll()` 的概念，但若严格按题目字面，答案不变。在此按照严格字面解释。
- **English Explanation**: The syntax `list1.remove(list2)` as written is incorrect for removing all common elements. The `remove()` method is for removing a single element or a collection of elements that *are contained within* the list itself. The correct method to remove all elements common to both lists is `list1.removeAll(list2)`. If `list1.removeAll(list2)` were intended, `list1` would become `[]`. However, interpreting the question literally as `list1.remove(list2)`, the `list2` object itself would not be found and

removed from `list1` (unless `list1` coincidentally contained `list2` as an element), so `list1` would remain `["red", "yellow", "green"]`. Given this is a test, `removeAll()` might be implied, but strictly adhering to the literal question, the list remains unchanged.

19.

Stacks, **Queues** (栈，队列)

20.

- **中文解析**: 栈是后进先出（LIFO）的数据结构，队列是先进先出（FIFO）的数据结构。
- **English Explanation**: Stacks are data structures that process objects in a Last-In, First-Out (LIFO) manner. Queues are data structures that process objects in a First-In, First-Out (FIFO) manner.

21.

O(N2) (或 O(N2), O(N2), O(N2) 分别对应)

22.

- **中文解析**: 选择排序、插入排序和冒泡排序在最坏情况下的时间复杂度都是O(N2)。
- **English Explanation**: Selection sort, insertion sort, and bubble sort all have a worst-case time complexity of O(N2).

23.

preorder (前序), **postorder** (后序)

24.

- **中文解析**: 前序遍历是：根 -> 左 -> 右。后序遍历是：左 -> 右 -> 根。
- **English Explanation**: A preorder traversal visits the root node first, followed by the left subtree, and then the right subtree. Conversely, a postorder traversal visits the left subtree first, followed by the right subtree, and finally the root node.

25.

double left (RL) then right (LR) (或具体的旋转组合，取决于具体插入位置和不平衡情况，但 RL 往往需要两个旋转)

26.

- **中文解析**: 当在右子树的左子树插入导致 RL 不平衡时，通常会先对右子节点进行一次右旋（R 旋转），使其变为 RR 类型不平衡，然后再对根节点进行一次左旋（L 旋转）来重新平衡。
- **English Explanation**: When an RL imbalance occurs (an insertion into the left subtree of the right child), it typically requires a double rotation: first a right rotation on the right child to convert it into an RR imbalance, and then a left rotation on the root node to rebalance the tree.

27.

linear, **quadratic** (线性，二次)

28.

- **中文解析**: 线性探测是顺序查找下一个可用槽位。二次探测是按平方递增的距离查找槽位。
- **English Explanation**: Linear probing resolves collisions by sequentially checking the next available slots. Quadratic probing resolves collisions by checking slots at increasing squared distances from the original hash.

29.

"Charlie"

30.

- **中文解析**: 在 `HashMap` 中，当使用相同的键 `put` 两次时，后一个值会覆盖前一个值。所以 "101" 对应的值最终是 "Charlie"。
- **English Explanation**: In a `HashMap`, when the same key is used with the `put` method multiple times, the last value associated with that key overwrites any previous values. Therefore, `myMap.get("key1")` will return "Charlie".

31.

中文解析: `PriorityQueue` 能够确保每次取出（`poll()`）的元素都是优先级最高的（或最低的，取决于排序规则），这对于任务调度系统非常关键，因为它需要总是处理最紧急的任务。虽然 `LinkedList` 也可以存储任务，但要找到最高优先级的任务，你需要遍历整个列表（O(N)），或者在插入时保持排序（O(N)），效率远低于 `PriorityQueue` 的 O(logN)（插入和删除）。`PriorityQueue` 的底层是堆，提供了高效的优先级管理。 **English Explanation**: A `PriorityQueue` guarantees that when you retrieve (or `poll()`) an element, it is always the highest (or lowest, depending on the ordering) priority element. This is crucial for a task scheduling system as it constantly needs to process the most urgent task. While a `LinkedList` can also store tasks, finding the highest priority task would require iterating

through the entire list (O(N)) or maintaining a sorted list upon insertion (O(N)), which is much less efficient than a `PriorityQueue`'s O(logN) for insertion and removal. The `PriorityQueue` uses a heap internally, providing efficient priority management.

32.
33.

中文解析:

34.

- **方法重载 (Method Overloading)**: 在同一个类中，定义多个名称相同但参数列表（数量、类型或顺序）不同的方法。重载方法可以有不同的返回类型，但参数列表必须不同。这是编译时多态（静态多态）的体现。
  - **示例**:

```
Java
```

```
class Calculator {
    int add(int a, int b) { return a + b; }
    double add(double a, double b) { return a + b; }
}
```

- **方法重写 (Method Overriding)**: 子类对父类中已有的方法提供自己特定的实现。方法签名（方法名、参数列表和返回类型）必须与父类方法完全一致（或返回类型是其子类型）。这是运行时多态（动态多态）的体现。
  - **示例**:

```
Java
```

```
class Animal {
    void makeSound() { System.out.println("Animal
makes a sound"); }
}class Dog extends Animal {
    @Override
    void makeSound() { System.out.println("Dog
barks"); }
```

```
}
```

-

English Explanation:

- **Method Overloading**: Defining multiple methods within the same class that have the same name but different parameter lists (number, type, or order of parameters). Overloaded methods can have different return types, but the parameter list must differ. This is an example of compile-time polymorphism (static polymorphism).
  - **Example**: (See above Java code)
- **Method Overriding**: A subclass provides its own specific implementation for a method that is already defined in its parent class. The method signature (method name, parameter list, and return type) must be exactly the same as the parent method (or the return type can be a subtype). This is an example of run-time polymorphism (dynamic polymorphism).
  - **Example**: (See above Java code)

35.

中文解析:

36.

- 如果一个算法对列表中的每个元素执行常数次操作，其时间复杂度是 O(N)（线性时间）。
- 如果这个算法被应用于 N 个嵌套循环中（即 N 个元素的列表在 N 次迭代的外部循环中被处理，且内部循环也与 N 相关），那么它的时间复杂度通常会是 O(N2)。例如，如果外部循环迭代 N 次，内部循环也迭代 N 次，总操作次数约为 N×N=N2。 **English Explanation**:
- If an algorithm performs a constant number of operations for each element in a list of `N` elements, its time complexity is O(N) (linear time).
- If this algorithm is then applied within N nested loops (i.e., processing an `N`-element list within an outer loop that iterates `N` times, and the inner loop also depends on `N`), its time complexity would typically be O(N2). For instance, if the outer loop iterates `N` times and the inner loop also iterates `N` times, the total operations would be approximately `N * N = N^2`.

37.

中文解析: 为了同时满足按 ID 高效查找和按插入顺序迭代的需求，最适合的组合是：

38.

- `HashMap<String, Student>`: 用于按学生 ID 进行高效查找（平均 O(1) 时间复杂度）。键是学生 ID，值是 `Student` 对象。
- `LinkedHashSet<Student>` (或维护一个单独的 `ArrayList<Student>`): 如果需要遍历所有学生并保持插入顺序。`LinkedHashSet` 保持插入顺序且保证唯一性，但查找效率不如 `HashMap`。另一种方案是使用 `HashMap` 进行查找，同时维护一个 `ArrayList` 来存储所有学生，并按照它们被添加到系统的顺序进行管理。当需要按插入顺序迭代时，遍历 `ArrayList`。 **English Explanation**: To efficiently meet both the requirement for quick lookups by student ID and iterating through all student records in their insertion order, the most suitable combination would be:
- `HashMap<String, Student>`: For efficient lookups by student ID (average O(1) time complexity). The keys would be student IDs, and values would be `Student` objects.
- `LinkedHashSet<Student>` (or maintaining a separate `ArrayList<Student>`): If iteration in insertion order is required. `LinkedHashSet` preserves insertion order and ensures uniqueness, but its lookup efficiency is not as good as `HashMap`. Another approach is to use a `HashMap` for lookups and separately maintain an `ArrayList` to store all students in the order they were added to the system. When iteration in insertion order is needed, the `ArrayList` can be traversed.

39.

中文解析:

40.

- **深度优先搜索 (DFS)**: 沿着图的某一分支深入探索，直到不能再深入为止，然后回溯到最近的未访问节点，继续沿着新的分支深入。它使用**栈**（隐式或显式）来实现。
- **广度优先搜索 (BFS)**: 从起始节点开始，逐层访问所有相邻的未访问节点，然后继续访问下一层的节点。它使用**队列**来实现。 **English Explanation**:
- **Depth-First Search (DFS)**: Explores as far as possible along each branch before backtracking. It goes deep into a path before exploring siblings. It typically uses a **stack** (either implicitly via recursion or explicitly) for implementation.
- **Breadth-First Search (BFS)**: Explores all the neighbor nodes at the current level before moving on to the nodes at the next level. It expands outwards from the starting node. It uses a **queue** for implementation.

Part 1:专项练习题
Level 1 (Easy) 填空题
In Java, a variable that stores the actual values of primitive data types is called a _____ variable.
Java 中，存储原始数据类型实际值的变量称为_____变量。
Answer/答案: Primitive (基本类型变量)

The Java feature that allows a subclass to provide a specific implementation of a method from its parent class is called .
允许子类重写父类方法的 Java 特性称为。
Answer/答案: Method overriding (方法重写)

The data structure that follows a last-in, first-out (LIFO) order is called a .
按后进先出顺序处理元素的数据结构是。
Answer/答案: Stack (栈)

In Java, the method to retrieve and remove the head of a Queue without throwing an exception if empty is called .
Java 中从队列获取并移除头元素且队列为空时不抛异常的方法是。
Answer/答案: poll()

The interface that defines the natural ordering of objects is called _____ in Java.
Java 中定义对象自然排序的接口称为_____。
Answer/答案: Comparable

The collection class most efficient for retrieving elements by index is .
最适合通过索引快速访问元素的集合类是。
Answer/答案: ArrayList

The output of the expression o1 == o2 in Java checks if two references point to the same .
表达式 o1 == o2 判断两个引用是否指向同一个。
Answer/答案: Object (对象)

A key characteristic of a binary search tree is that the left subtree of a node contains values _____ than the node.
二叉搜索树的一个关键特征是节点的左子树存放_____该节点的值。
Answer/答案: Less

The time complexity of an algorithm with linear growth is expressed as .
时间复杂度线性增长的算法表示为。
Answer/答案: O(n)

In hashing, the collision resolution strategy that uses linked lists at each bucket is called .
哈希时采用链表解决冲突的策略称为。
Answer/答案: Separate chaining (链地址法)

Level 2 (Medium) 填空题
Java generics require type parameters to be _____ types; you cannot use primitives like int.
Java 泛型要求类型参数为_____类型，不能使用原始类型如 int。
Answer/答案: Reference (引用类型)

The Java method clone() produces a _____ copy of an object.
Java 的 clone() 方法产生_____拷贝。
Answer/答案: Shallow (浅拷贝)

The collection class suited for fast insertion and deletion from the middle or beginning of a list is .
适合在列表中间或开头快速插入和删除的集合类是。
Answer/答案: LinkedList

The Java structure used to define a contract with abstract methods but cannot have instance fields is a(n) .
用来定义含抽象方法且无实例字段的 Java 结构是。
Answer/答案: Interface (接口)

In selection sort, the element identified and swapped in each iteration is the _____ element.
选择排序中每轮确认并交换的是数组中的_____元素。
Answer/答案: Minimum

The Balanced Factor in AVL tree is calculated by subtracting the height of the right subtree from the height of the _____ subtree.
AVL 树中平衡因子计算为左子树高度减去_____子树高度。
Answer/答案: Right (右)

The Java collection interface that allows duplicate elements and maintains insertion order is .
允许重复元素且保持插入顺序的集合接口是。
Answer/答案: List

The PriorityQueue in Java by default orders elements based on their _____.
Java 中 PriorityQueue 默认按元素的_____顺序排列。
Answer/答案: Natural ordering (自然顺序)

The comparator lambda expression to sort strings case-insensitively would use the method .

用于忽略大小写排序字符串的比较器 lambda 表达式调用方法是。
Answer/答案: String.CASE_INSENSITIVE_ORDER

In Big-O notation, the time complexity of nested loops each running n times is expressed as .
在 Big-O 表示法中，两个嵌套的循环各执行 n 次的时间复杂度为。
Answer/答案: $O(n^2)$

The difference between a simple graph and a complete graph is that a complete graph connects every pair of vertices by exactly one _____.
简单图和完全图的区别在于完全图的每对顶点之间用唯一的_____边相连。
Answer/答案: Edge (边)

In Java, the method header of a generic method that prints elements of an array of type E is .
Java 中泛型方法打印类型为 E 数组元素的方法头应写为。
Answer/答案: public static <E> void print(E[] list)

The interface that must be implemented to sort custom objects with Arrays.sort() is .
自定义对象要用 Arrays.sort 排序需实现的接口是。
Answer/答案: Comparable

To represent an undirected graph where every two distinct vertices are connected by exactly one edge, it is called a _____ graph.
在无向图中，每两个不同顶点间恰有一条边连接称为_____图。
Answer/答案: Complete

The traversal method that visits the root node first, then the left subtree, then the right subtree is called _____ traversal.
先访问根节点，再访问左子树，最后访问右子树的遍历方式称为_____遍历。
Answer/答案: Preorder (前序)

Level 3 (Hard) 填空题
In a graph, edges connecting the same two vertices more than once are called .
图中连接两个顶点的多个边称为。
Answer/答案: Multiple edges (多重边)

The algorithm most suitable for finding the shortest path between two nodes in a weighted graph is called .
用于寻找带权图中两点最短路径的算法是。
Answer/答案: Dijkstra’s Algorithm (迪杰斯特拉算法)

The rotation necessary in an AVL tree when inserting a node into the left subtree of the right child causing imbalance is called _____ rotation.
AVL 树中插入节点引起右子节点左子树不平衡时，应执行_____旋转。
Answer/答案: Right-Left (右左旋转)

The theoretical approach to analyze how algorithm execution time grows with input size is called .
分析算法执行时间随输入规模增长的理论方法称为。
Answer/答案: Big-O notation (大 O 表示法)

The difference between a simple graph and a complete graph is that a complete graph connects every pair of vertices by exactly one _____.
简单图和完全图的区别在于完全图的每对顶点之间用唯一的_____边相连。
Answer/答案: Edge (边)

第三套专项练习题
Level 1 (Easy) 填空题 (10 题)
Java primitive types include int, double, char, and .
Java 原始数据类型包括 int、double、char 和。
Answer/答案: boolean (布尔型)

The keyword used in Java to inherit from a superclass is .
Java 中用于继承父类的关键字是。
Answer/答案: extends

In Java, the operator “==” checks whether two references point to the same .
Java 中“==”运算符用于判断两个引用是否指向同一个。
Answer/答案: Object (对象)

Which Java collection class implements the List interface using a resizable array? It is called .
实现 List 接口且基于可变数组的 Java 集合类是。
Answer/答案: ArrayList

The Java keyword to define a constant variable is .
Java 中用来定义常量的关键字是。
Answer/答案: final

Java’s method overloading means having _____ methods with the same name but different parameter lists.
Java 中方法重载指的是拥有名称相同但参数列表_____的方法。
Answer/答案: Multiple (多个)

The stacking data structure operates on the _____ in, first out principle.
栈数据结构遵循_____进先出原则。

Answer/答案: Last (后，即后进先出)

The interface that all classes implement in Java implicitly is called .
所有 Java 类隐式实现的接口是。
Answer/答案: Object

The method in Java to compare objects' content equality is called .
用于比较对象内容是否相等的方法是。
Answer/答案: equals()

The standard output of System.out.println is directed to .
System.out.println 标准输出定向到。
Answer/答案: Console (控制台)

Level 2 (Medium) 填空题 (15 题)
The difference between HashMap and TreeMap is that TreeMap keeps entries in _____ order.
HashMap 与 TreeMap 的区别是 TreeMap 按_____顺序存储条目。
Answer/答案: Sorted (排序)

The Java keyword used to define an abstract method is .
用于定义抽象方法的关键字是。
Answer/答案: abstract

The method in Java collections to add all elements from one collection to another is called .
Java 集合中用于将一个集合所有元素添加到另一个集合的方法是。
Answer/答案: addAll()

The time complexity of binary search algorithm is .
二分查找算法的时间复杂度是。
Answer/答案: O(log n)

The traversal that visits nodes layer by layer in a tree is called _____ traversal.
树中分层访问结点的遍历方式称为_____遍历。
Answer/答案: Level-order (层序)

In Java, the keyword used to prevent a method from being overridden by subclasses is .
Java 中禁止方法被子类重写的关键字是。
Answer/答案: final

Exception handling in Java is done with try, catch, and _____ blocks.
Java 中的异常处理用 try、catch 和_____块实现。
Answer/答案: finally

The Comparator interface in Java requires implementation of the _____ method.

Java 中 Comparator 接口要求实现_____方法。
Answer/答案: compare()

The data structure that supports insertion and deletion only at the front and back but not in the middle is called .
只支持在前后插入删除的数据结构称为。
Answer/答案: Deque (双端队列)

In recursion, the simplest case that stops the recursion is called the _____ case.
递归中用于终止递归的最简单情况称为_____情况。
Answer/答案: Base (基本)

In Java, the process of wrapping a primitive type into its corresponding wrapper class is called .
Java 中将原始类型包装成对应包装类的过程称为。
Answer/答案: Autoboxing (自动装箱)

The difference between a checked exception and an unchecked exception is that checked exceptions must be handled or declared, but unchecked exceptions are subclasses of _____.
受检异常必须处理或声明，非受检异常是_____的子类。
Answer/答案: RuntimeException

The method overridden in the Object class to enable hashing in collections like HashSet and HashMap is called .
Object 类中用于哈希的重写方法是。
Answer/答案: hashCode()

The binary tree traversal order that visits left subtree, then root, then right subtree is called _____ traversal.
二叉树中先访问左子树，再根节点，最后右子树的遍历称为_____遍历。
Answer/答案: Inorder (中序)

The algorithm that sorts by repeatedly swapping adjacent elements that are in the wrong order is called _____ sort.
通过反复交换相邻逆序元素完成排序的算法称为_____排序。
Answer/答案: Bubble (冒泡)

Level 3 (Hard) 填空题 (5 题)
The balancing factor in AVL tree is defined as height of left subtree minus height of _____ subtree.
AVL 树的平衡因子定义为左子树高度减去_____子树高度。
Answer/答案: Right (右)

In graph theory, a graph without any cycles is called a _____ graph.
图论中不含环的图称为_____图。

Answer/答案: Acyclic (无环图)

The method used to handle collisions in open addressing by checking slots at increasing square distances is called _____ probing.
开放寻址法中以平方间距检查槽位解决冲突的方法称为_____探测。
Answer/答案: Quadratic (平方探测)

The condition when an AVL tree node’s balance factor is +2 or -2 indicates the need for _____ to rebalance.
AVL 树节点平衡因子为+2 或-2 时应进行_____以重新平衡。
Answer/答案: Rotation (旋转)

Searching for an element in a balanced binary search tree has time complexity .
在平衡二叉搜索树中查找元素的时间复杂度是。
Answer/答案: O(log n)