1. 数组 (Arrays)

基本的数据结构，可以存储固定大小的相同类型的元素。

优点：随机访问元素效率高；缺点：大小固定，插入/删除元素相对较慢。

```
int[] array = new int[5];
```

2. 列表 (Lists)

ArrayList：`List<String> arrayList = new ArrayList<>();` O(n)

底层数据结构：动态数组。通过索引可直接计算出元素地址

动态数组，可变大小。

优点：高效的随机访问和快速尾部附插入。

缺点：中间插入和删除相对较慢。

LinkedList `List<Integer> linkedlist = new LinkedList<>();` O(1)

双向链表，元素之间通过指针连接。

优点：插入和删除元素高效，

缺点：随机访问相对较慢

底层结构：双向链表，元素在内存中分散存储

通过指针连接。

3. 集合 (Sets)

存储不重复的元素。

HashSet `Set<String> hashset = new HashSet<>();`

无序集合，基于 HashMap 实现。

优点：高效的查找和插入

缺点：不保证顺序。

TreeSet `Set<Integer> treeSet = new TreeSet<>();`

有序集合，底层基于红黑树，不允许出现重复值。

优点：自动排序，适用需要按顺序存储元素的场景。

缺点：不允许插入 null 值。

4. 映射 (Maps)

用于存储键值对。

HashMap `Map<String, Integer> hashMap = new HashMap<>();`

基于哈希表实现键值对

优点：高效的查找，插入、删除。缺点：无序，不保证顺序。

TreeMap `Map<String, Integer> treeMap = new TreeMap<>();`

基于红黑树实现有序键值对存储结构。

优点：有序，支持按照键的顺序遍历。

缺点：插入和删除相对较慢。

5. 栈 (Stack)

一种线性结构，按照后进先出原则。

6. 队列 (Queue)

先进先出。

7. 堆 (Heap)

堆作为队列基础，实现最大堆和最小堆。