??? ??

Java Coding Test Cheatsheet

0. Frequently Used Libraries

```
import java.util.*;      // Data structures
import java.io.*;        // Fast I/O
import java.math.*;      // BigInteger, BigDecimal
```

1. Variable & Array Declaration

```
String[] arr1 = new String[5];
int[] arr2 = {1, 2, 3};
int N = 3;
int[] arr3 = new int[N];
```

2. Arrays Utility

```
Arrays.sort(arr);                                  // Ascending
Arrays.sort(arr, Collections.reverseOrder());     // Descending
Arrays.sort(arr, 0, 4);                            // Partial sort
Arrays.binarySearch(arr, 2);                       // Binary search
List<String> list = Arrays.asList(arr1);           // Convert to List
int[] tmp = Arrays.copyOfRange(arr, 0, 3);         // Slice
```

3. length / length() / size()

```
arr.length        // Arrays
str.length()      // String
list.size()       // Collections
```

4. String Handling

```
str.split(" ");
str.substring(0, 5);
str.charAt(i);
str.toUpperCase();
str.toLowerCase();
String[] letters = str.split("");
String newStr = str.substring(0,4) + "X" + str.substring(5);
```

5. HashMap

```
Map<String, Integer> map = new HashMap<>();
map.put("key", 1);
map.get("key");
map.containsKey("key");
map.getOrDefault("key", 0);
for (String k : map.keySet()) map.get(k);
```

6. ArrayList

```
List<String> list = new ArrayList<>();
list.add("a");
list.set(0, "b");
list.remove("b");
list.contains("a");
list.indexOf("a");
```

7. Queue (LinkedList)

```
Queue<Integer> q = new LinkedList<>();
q.offer(1);
q.poll();
q.peek();
q.clear();
q.isEmpty();
```

8. PriorityQueue

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
PriorityQueue<Integer> maxPq = new PriorityQueue<>(Collections.reverseOrder());
```

9. Math

```
Math.max(a, b);
Math.min(a, b);
Math.abs(x);
Math.ceil(x);
Math.floor(x);
Math.round(x);
String.format("%.2f", d);  // Round to 2 decimal
Math.pow(a, b);
Math.sqrt(x);
```

10. HashSet

```
Set<String> set = new HashSet<>();
set.add("a");
set.remove("a");
set.contains("a");
```

11. Stack

```
Stack<Integer> stack = new Stack<>();
stack.push(1);
stack.pop();
stack.peek();
```

12. Deque (ArrayDeque)

```
Deque<Integer> dq = new ArrayDeque<>();
dq.addFirst(1);
dq.addLast(2);
dq.pollFirst();
dq.pollLast();
```

13. TreeSet

```
TreeSet<Integer> ts = new TreeSet<>();
ts.add(5);
ts.first();
ts.last();
ts.lower(5);
ts.higher(5);
```

14. TreeMap

```
TreeMap<String, Integer> tm = new TreeMap<>();
tm.put("apple", 3);
tm.firstKey();
tm.lastKey();
```

15. Fast I/O

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
StringTokenizer st = new StringTokenizer(br.readLine());
int a = Integer.parseInt(st.nextToken());
String[] parts = br.readLine().split(" ");

BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(System.out));
bw.write("Hello\n");
bw.flush();
bw.close();
```

16. DFS

```
void dfs(int node) {
    visited[node] = true;
    for (int next : graph.get(node)) {
        if (!visited[next]) dfs(next);
    }
}
```

17. BFS

```
Queue<Integer> q = new LinkedList<>();
q.offer(start);
visited[start] = true;
while (!q.isEmpty()) {
    int cur = q.poll();
    for (int next : graph.get(cur)) {
        if (!visited[next]) {
            q.offer(next);
            visited[next] = true;
        }
    }
}
```

18. Sorting

```
list.sort(Comparator.naturalOrder());
list.sort(Comparator.reverseOrder());
people.sort(Comparator.comparingInt(p -> p.age));
```

19. DP (Fibonacci)

```
int[] dp = new int[N+1];
dp[0] = 0; dp[1] = 1;
for (int i = 2; i <= N; i++) dp[i] = dp[i-1] + dp[i-2];
```

20. Union-Find

```
int find(int x) {
    if (x != parent[x]) parent[x] = find(parent[x]);
    return parent[x];
}
void union(int a, int b) {
    a = find(a); b = find(b);
    if (a != b) parent[b] = a;
}
```

21. Bitmask

```
int bit = 0;
bit |= (1 << 3);
bit &= ~(1 << 3);
boolean on = (bit & (1 << 3)) != 0;
Integer.bitCount(bit);
```

22. Greedy

```
int[] coins = {500, 100, 50, 10};
int count = 0;
for (int coin : coins) {
    count += target / coin;
    target %= coin;
}
```

23. Prefix Sum

```
int[] prefix = new int[N+1];
for (int i = 0; i < N; i++) prefix[i+1] = prefix[i] + arr[i];
```

24. Sliding Window

```
int sum = 0;
for (int i = 0; i < k; i++) sum += arr[i];
int max = sum;
for (int i = k; i < N; i++) {
    sum += arr[i] - arr[i-k];
    max = Math.max(max, sum);
}
```

25. Sieve of Eratosthenes

```
boolean[] isPrime = new boolean[N+1];
Arrays.fill(isPrime, true);
for (int i = 2; i*i <= N; i++) {
    if (isPrime[i]) {
        for (int j = i*i; j <= N; j += i) isPrime[j] = false;
    }
}
```

26. Dijkstra

```
PriorityQueue<int[]> pq = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
int[] dist = new int[N+1];
Arrays.fill(dist, INF);
dist[start] = 0;
pq.offer(new int[]{start, 0});
```

27. Tree / LCA

```
void dfs(int node, int par, int d) {
    parent[node] = par;
    depth[node] = d;
    for (int next : tree.get(node)) {
        if (next != par) dfs(next, node, d+1);
    }
}
```

28. Coding Test Patterns

- Use BufferedReader & BufferedWriter
- HashMap, HashSet, TreeMap, PriorityQueue
- Always validate inputs
- Use boolean[], Set, or visited[][] for visited states
- Practice common patterns (DFS/BFS, DP, Greedy, Two Pointers)

29. Backtracking (Permutation, Combination)

```
void permute(List<Integer> list, boolean[] used) {
    if (list.size() == N) {
        System.out.println(list);
        return;
    }
    for (int i = 0; i < N; i++) {
        if (!used[i]) {
            used[i] = true;
            list.add(arr[i]);
            permute(list, used);
            list.remove(list.size() - 1);
            used[i] = false;
        }
    }
}
```

30. Binary Search / Parametric Search

```
int left = 1, right = maxVal;
while (left <= right) {
    int mid = (left + right) / 2;
    if (condition(mid)) {
        answer = mid;
        right = mid - 1;
    } else {
        left = mid + 1;
    }
}
```

31. Recursion Optimization

- Tail recursion is not optimized in Java.
- Prefer loops for heavy stack recursion.
- Consider memoization or iterative conversion.

32. String Algorithms (KMP)

```
int[] makeTable(String pattern) {
    int[] table = new int[pattern.length()];
    int j = 0;
    for (int i = 1; i < pattern.length(); i++) {
        while (j > 0 && pattern.charAt(i) != pattern.charAt(j)) j = table[j-1];
        if (pattern.charAt(i) == pattern.charAt(j)) table[i] = ++j;
    }
    return table;
}
```

33. Permutations / Combinations / Bitmask

```
// Bitmask Combination
for (int mask = 0; mask < (1 << N); mask++) {
    for (int i = 0; i < N; i++) {
        if ((mask & (1 << i)) != 0) {
            // element i is selected
        }
    }
}
```

34. BigInteger / BigDecimal

```
BigInteger a = new BigInteger("12345678901234567890");
BigInteger b = new BigInteger("98765432109876543210");
BigInteger sum = a.add(b);
BigDecimal dec = new BigDecimal("1234.5678");
BigDecimal result = dec.setScale(2, RoundingMode.HALF_UP);
```

□□ : chatGPT

Java ?? ?? ?? ??? ??

1. ?? ??: `Arrays.equals()`

```
Arrays.equals(array1, array2);
```

[illegible]	[illegible]
[illegible] [illegible]	[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
[illegible]	`true`([illegible]), `false`([illegible])
[illegible]	`==` [illegible] `.equals()`[illegible] [illegible] [illegible] [illegible] [illegible] [illegible]

[illegible] [illegible]:

```
int[] a = {1, 2, 3};
int[] b = {1, 2, 3};
System.out.println(Arrays.equals(a, b)); // [illegible] true
System.out.println(a == b);              // [illegible] false ([illegible] [illegible])
```

2. ?? ??: `Arrays.copyOf()`

```
Arrays.copyOf([illegible], [illegible]);
```

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].
[illegible] [illegible], [illegible] [illegible] [illegible].

[illegible] [illegible]:

```
int[] original = {1, 2, 3, 4};
int[] copied = Arrays.copyOf(original, 2);  // [1, 2]
```

? ?? ??: ?? ?? ? `0`?? ??

```
int[] extended = Arrays.copyOf(original, 6);  // [1, 2, 3, 4, 0, 0]
```

3. ?? ??: `Arrays.copyOfRange()`

```
Arrays.copyOfRange([illegible], [illegible], [illegible]);
```

[illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] [illegible] ([illegible] [illegible])[illegible].

[illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible]:

```
int[] sub = Arrays.copyOfRange(original, 1, 3);  // [2, 3]
```

4. ?? ??: `Arrays.sort()`

```
Arrays.sort([illegible]);
```

[illegible] [illegible] [illegible] [illegible] [illegible]. (in-place [illegible], [illegible] [illegible] `void`)

void[illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible].

[illegible] [illegible] ([illegible]: [illegible], [illegible]: [illegible] [illegible])

[illegible] [illegible]:

```
int[] arr = {5, 1, 3};
Arrays.sort(arr);  // arr → [1, 3, 5]
```

5. ?? ??: `Arrays.toString()`

```
Arrays.toString([illegible]);
```

[illegible] [illegible] [illegible] [illegible]. ([illegible])

[illegible] [illegible]:

```
System.out.println(Arrays.toString(arr)); // [1, 3, 5]
```

6. ?? ???: `Arrays.fill()`

```
Arrays.fill(□□, □);
```

□□ □□□ □□□ □□□ □□□□□ .

□□ □□ :

```
int[] arr = new int[5];
Arrays.fill(arr, 7); // arr → [7, 7, 7, 7, 7]
```

7. ?? ?? ??: `Arrays.binarySearch()`

```
Arrays.binarySearch(□□□□□, □□□);
```

□□□ □□□□ □□ □□□ □□□□ .
□□□□ □□□□□ .(□□□ □□)

□□ □□ :

```
int[] sorted = {1, 2, 3, 4};
int idx = Arrays.binarySearch(sorted, 3); // 2
```

? ??

□□	□□□
□□ □□ □□	`Arrays.equals()`
□□ □□	`Arrays.copyOf()`
□□ □□	`Arrays.copyOfRange()`
□□ □□	`Arrays.sort()`
□□ □□	`Arrays.toString()`

□□ □□□	`Arrays.fill()`
□□ □□	`Arrays.binarySearch`

? ???? ?? ?? ??

□□	□□ □□
□ □□□□ □□ □□□ □□□□□□ □□	`toCharArray()` → `Arrays.sort()` → `Arrays.equals()`
□□ □□ □□□□ □□	`Arrays.copyOf()` + `Arrays.sort()`
□□ □□□□	`Arrays.copyOfRange()`

Java ???(Expression)

1. ??????

> **□□□ (Expression)**□□ , □□ □□□□ □□ □□ □□ .
> □□ , □□ , □□ □□ , □□ □□ □ □□□□ □□□ □□ □□ □ □□□□□

2. ??? ?? ???

□□	□□	□□
`3 + 4`	`7`	□□ □□
`"Hi" + " there"`	`"Hi there"`	□□□ □□
`new String("abc")`	`"abc"`	□□ □□
`arr.length`	□□ □□	□□□ □□ □□
`x > 5`	`true/false`	□□ □□

3. ???? ??(statement)? ??

- □□□ : □□ □□□
- □□ : □□□ □□□□ (□□)

```
// □□□
3 + 4               // □: 7
"Hello" + "World"   // □: "HelloWorld"
x > 5               // □: true □□ false

// □□
int a = 3 + 4;                // □□ □□ (□□)
System.out.println("Hi");   // □□□ □□
if (x > 5) { ... }            // □□□
```

4. ???? ??? ??

□□	□□
return □	`return x + y;`
□□ □□	`int z = x * 2;`
□□□ □□	`if (a > b)`
□□□ □□	`System.out.println("Hi");`

5. ??

- □□□□ □□ □□ □□□.
- □□ □□□ □□ □□□ □□ □□□ □□□□ □□ □ □□ .
- □□ □□ , □□ , □□ □□ □ □□□ □□ □□ □ □□□□□ .

Java ????? ??? (Escape Sequences)

1. ????? ?????

> □□□□□ □□□ □ □□ □□□□ □□□ □ □□ □□□ □□ (□: □□□, □ □)□ □□□ □□□ □□□ □ □□□ □□□ □□□□ `\`□ □□□□ □□ □□ □□.

2. ?? ????? ??? ?

□□□□□	□□	□□ □□
`\n`	□□□ (newline)	□□ □□□
`\t`	□ (tab)	□□□□ 4~8□ (□ □□□ □□ □□)
`\"`	□□□□ (")	□□□ □□□ □□□□ □□ □□
`\'`	□□□□□ (')	□□□□□□ □□ □□ □□□ □□□ □□
`\\`	□□□□ ()	□□□□ □□ □□
`\r`	□□□ □□ (CR)	□□□ □ □□□ □□ (□□ □□ □ □)
`\b`	□□□□□	□□ □□ □□ (□□ □□ □ □)
`\f`	□ □□ (Form feed)	□□□ □□ (□□□ □□□, □□□ □□ □□ □ □)

3. ?? ?? ??

```
public class EscapeExample {
    public static void main(String[] args) {
        System.out.println("□□\t□□\t□□");
        System.out.println("□□\t20\t□");
        System.out.println("□□ □□□: \"□□□□□\"");
        System.out.println("C:\\Program Files\\Java");
    }
}
```

?? ??:

```
□□□□□□□□
□□20□
□□ □□□: "□□□□□"
C:\Program Files\Java
```

4. ????

- □□□ □□□ **□□□□ (`"`)□ □□□□ (`\`)**□ □□□□ □ □
- □□ □□□ □□□□□ □□□ □ □
- □□□□ □□ □□ □□□□□ □□□□ □□□ □□
- □□□□□ □□□□ **□□□(`"..."`)□□ □□(`'...'`)** □□□□ □□ □□

```
System.out.println("Hello\nWorld");  // □□□
char quote = '\'';                   // □□□□□ □□
```

Java ?? ?? ?? (????? ???, printf, format)

1. ????? ?????

> **□□□□□ □□□ (Escape Sequence)**□ □□□ □□□ □□□ □□ (□□□ , □ , □□□ □)□ □□□□ □□ □□ □□□□ .
> Java□□□ `\` □□□□□ □□□□ .

1.1 ?? ????? ???

□□□□□	□□	□□ □□
`\n`	□□□	□□ □□
`\t`	□	□□□□ (□□ 4~8□)
`\"`	□□□□ □□	`"□□ □□□: \"□□\""`
`\'`	□□□□□ □□	`'I\'m fine'`
`\\`	□□□□ □□	`"C:\\Users\\Dain"`
`\r`	□□□ □□	□□□ □ □□□ □□ (□□ □ □)
`\b`	□□□□□	□□ □□ □□ (□□ □ □)
`\f`	□ □□	□□□ □□ (□□ □ □)

1.2 ?? ??

```
System.out.println("□□\t□□");
System.out.println("□□\t25");
System.out.println("□□ □□□: \"□□□□□\"");
System.out.println("C:\\Program Files\\Java");
```

□□ □□ :

```
□□      □□
□□     25
□□ □□□: "□□□□□"
C:\Program Files\Java
```

---

2. ??? ?? ?? ??

Java□□ □□□□ □□□ □□□□ □□□□ □□ □ `System.out.printf()` □□ `String.format()`□ □□□□ .

2.1 ?? ??

```
System.out.printf("□□", □1, □2...);
String result = String.format("□□", □1, □2...);
```

2.2 ?? ?? ??

□□ □□	□□	□□ □□
`%s`	□□□	`"□□: %s"` → □□ □□
`%d`	□□ (10□□)	`"□□: %d"`
`%f`	□□ (□□□ □□)	`"□□: %.2f"`
`%c`	□□	`"□□: %c"`
`%n`	□□□ (OS□ □□)	`"□□ □%n"`
`%%`	□□□ (%) □□	`"100%% □□"`

2.3 ???/?? ??

□□	□□
`%5d`	□□ 5□□ , □□□ □□
`%-5d`	□□ 5□□ , □□ □□
`%05d`	□□□□ 0□□ □□ (□ 5□□)
`%.2f`	□□□ □□□□□□ □□

`%6.2f`	□□ 6□ □ □□□ 2□□ □□

2.4 ?? ??

```
String name = "□□";
int age = 25;
double score = 93.756;

System.out.printf("□□: %s, □□: %d%n", name, age);
System.out.printf("□□: %.2f%n", score);
System.out.printf("□□□: %d%%%n", 100);
```

□□ □□ :

```
□□: □□, □□: 25
□□: 93.76
□□□: 100%
```

3. ?? ???

□□	□□	□□
□□□□□ □□	`\n`, `\t`, `\\`	□□□ □□ □□
□□ □□ (printf)	`%d`, `%.2f`, `%s`	□□□ □□□□ □□

4. ?? ?? ?? (ANSI ??)

Java □□□ □□ □□ □□□ □□□□ □□□ , □□□□□ **ANSI** □□ □ □□□□ □□ □□ □□□ □ □□ .

```
public class ColorExample {
    public static final String RED = "\u001B[31m";
    public static final String RESET = "\u001B[0m";

    public static void main(String[] args) {
        System.out.println(RED + "□ □□□ □□□□□." + RESET);
    }
}
```

□□	ANSI □□
□□	`\u001B[31m`
□□	`\u001B[32m`
□□	`\u001B[33m`
□□	`\u001B[34m`
□□	`\u001B[0m`

---

5. ?? ? ?? ?? ??

Java□□□ `java.time.LocalDateTime`□ `DateTimeFormatter`□ □□□ □□ /□□□ □□□ □□□□ □□□ □ □□ .

5.1 DateTimeFormatter ?? ??

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

LocalDateTime now = LocalDateTime.now();
DateTimeFormatter fmt = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
System.out.println("□□ □□: " + now.format(fmt));
```

5.2 ?? ?? ??

□□	□□	□□
`yyyy`	□□	`2025`
`MM`	□	`05`
`dd`	□	`22`
`HH`	□□ (24□□□)	`14`
`hh`	□□ (12□□□)	`02`
`mm`	□	`07`

`ss`	□	`59`
`a`	□□ /□□	`AM` / `PM`
`E`	□□	`Wed`

5.3 ??? ?? ?? ??

```
DateTimeFormatter customFmt = DateTimeFormatter.ofPattern("yyyy□ MM□ dd□ (E) a hh:mm");
String formatted = now.format(customFmt);
System.out.println("□□□□ □□: " + formatted);
```

---

6. ??? ?? ??

Java□□ □□ □□ □□□□ `java.util.logging.Logger`□ □□ .

6.1 Logger ?? ??

```
import java.util.logging.*;

Logger logger = Logger.getLogger("MyLogger");

logger.info("□□ □□□");
logger.warning("□□ □□□");
logger.severe("□□ □□□");
```

6.2 ?? ???? ?? ??

```
String name = "□□";
int age = 25;
logger.info(String.format("□□□ □□: %s, □□: %d", name, age));
```

6.3 ?? ?? ??

□□	□□
`SEVERE`	□□□ □□
`WARNING`	□□
`INFO`	□□ □□
`CONFIG`	□□ □□ □□
`FINE`	□□□ □□□ □□

6.4 ?? ?? ??????

- □□ □□ □□ (□□/□□)
- □□ □□□ □□ (□□, □□ □□, □□□ □□ □)
- `logging.properties` □□□ □□ □□ □□

Java ???(Regular Expression)

1. ??

□□□□□ (Regular Expression)□ □□ □□□□ □□□ □□ □□ , □□ , □□□□ □ □□□□ □□□ □□□□ . □□□□□ `java.util.regex` □□□□ □□ □□□□□□ □□□ □ □□ .

1.1. ????? ?? ??

□□□□□□ □□□□ □□□ □□ □□□ □□□□ □□□□□ , □□ □□□□ □ □□ □□ □□ □□□ □□ .

- □□ □□□ : `[ ]`
  - □ : `[a-z]`□ □□□ □□□□ □□□□ .
- □□□ : `{n,m}`, `+`, `*`, `?`
  - □ : `a{2,4}`□ 'a'□ 2□ □□ 4□ □□ □□□□ □□□ □□□□ .
- □□ □□ : `^`, `$`, `.`, `|`, `\`
  - □ : `^abc`□ 'abc'□ □□□□ □□□□ □□□□ .

1.2. ?? ???? ??

- `.`: □□□ □ □□ (□□ □□□ □□)
- `\d`: □□ (0-9)
- `\w`: □□□ □ □□ , □□ (a-zA-Z0-9_)
- `\s`: □□ □□ (□ , □□□□ □)
- `\b`: □□ □□

2. ???? ????? ????

2.1. `Pattern` ???

`Pattern` □□□□ □□□□□□ □□□ □□□□□ □ □□□□ . □ □□□□ □□□□□ □□□ □□□□ □□□□ □□□ □□□□ .

2.1.1. Pattern ???

```
Pattern pattern = Pattern.compile("a*b");
```

□□ □□□□ `"a*b"`□ 'a'□ 0□ □□ □□□□ □□ 'b'□ □□ □□□ □□□□ .

2.1.2. Matcher ??? ??

`Matcher` □□□□ `Pattern` □□□ □□□□ □□□□□ □□□ □□□□ .

```
Pattern pattern = Pattern.compile("a*b");
Matcher matcher = pattern.matcher("aaab");
boolean matches = matcher.matches(); // true
```

2.2. `Pattern`? `Matcher`? ?? ???

- `matches()`: □□ □□□□ □□□□□□ □□□ □□
- `find()`: □□□ □□□ □□□ □□□□□ □□□□□ □□
- `replaceAll()`: □□□□ □□□ □□ □□□□ □□

3. ????? ?? ??

3.1. ??? ?? ??

□□□ □□□ □□□□ □□□□ □□□□□ □□

```
Pattern pattern = Pattern.compile("^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-
]+\\.)+[a-zA-Z]{2,7}$");
Matcher matcher = pattern.matcher("example@domain.com");
boolean isValid = matcher.matches(); // true
```

3.2. ???? ?? ??

□□□□ □□□ □□□ □□□□ □□□□□ □□

```
Pattern pattern = Pattern.compile("^\\d{3}-\\d{3,4}-\\d{4}$");
Matcher matcher = pattern.matcher("010-1234-5678");
boolean isValid = matcher.matches(); // true
```

Java Arrays.sort() ??

1. ??

- `Arrays.sort()`□ Java□□ □□□ □□□ □ □□□□ □□ □□□□□ .
- □ □□□□ □□□ **□□□(in-place) □□ □□□□ , □□□□ □□ = (`void`□ □□□□.)**

2. ?? ??

```
Arrays.sort(□□);
```

□□□□	□□
□□	□□ □□ (□ : `int[]`, `char[]`, `String[]`)

3. ???? `void`? ??

□□	□□
□□□ □□	`Arrays.sort()`□ □□ □□ □□□ □□□
□□□ □□ □□ □ □	□□□ □□□ □□ □□□ □□□□ □□
□□□ □□	□□□□ □□ □□ □□ □□ □□ □□ □□
□□ X	`int[] sorted = Arrays.sort(arr);`□ □□ □□

4. ??

```
int[] numbers = {3, 1, 4};
Arrays.sort(numbers);  // numbers □□□ □□□


System.out.println(Arrays.toString(numbers)); // [1, 3, 4]
```

5. ?? ?? ?? ?

```
int[] sorted = Arrays.sort(numbers); // □ □□□ □□: void□ int[]□ □□□ □ □□
```

6. ??: ??? ??? ?? ??? ????

```
int[] original = {3, 1, 2};
int[] copy = Arrays.copyOf(original, original.length);
Arrays.sort(copy);
```

7. ?? ???

□□□	□□
`Arrays.sort(arr)`	□□ □□□□ □□
`Arrays.sort(arr, Comparator)`	□□□ □□ □□ □□ (□□ □□)
`Collections.sort(list)`	□□□ □□ (□□□ □□)