内存越界错误修复

为什么我什么都要依赖调试，不调试完全无法提前预料问题

这是一个很正常的现象，尤其是在 **C 语言** 这类底层语言中，因为它不像 Python、Java 这些语言有运行时检查和自动内存管理，C 语言完全由程序员负责 **内存管理** 和 **边界控制**，所以很容易发生 **越界访问、野指针、未初始化变量** 等问题，而这些错误**往往不会立即显现**，只有在某些特定情况下才会触发，例如：

- 访问越界但没有立刻崩溃（因为访问到了合法但错误的内存）。
- 运行时数据不同，导致某次运行出现 bug，而另一次运行正常。
- 某些编译优化可能掩盖了错误，导致看似"时好时坏"。

你为什么容易依赖调试？

1. C 语言本身没有数组越界检查

在 C 语言中，`malloc` 只是分配了一块内存，但**不会检查你访问的索引是否合法**。比如：

```c
int* arr = malloc(5 * sizeof(int)); // 只分配了 5 个整数
arr[5] = 10; // 超过了分配的范围，访问越界！
```

如果你不调试，可能程序不会崩溃（因为访问的是其他合法内存），但 `arr[5]` 存储的值可能是 **随机垃圾数据**，让你很难发现错误。

2. 你的代码逻辑中可能隐含了边界问题

在你的代码里：

```c
for (int i = root, j = 0; i < root + k; i++, j++) {
    Kth->data[j] = nums[i];
}
```

你默认 `root + k` **一定合法**，但如果 `k` 过大，这个 `i` 可能越界访问 `nums[i]`，导致**不可预测的行为**。

如果不调试，你可能完全意识不到 `i` 会超出 `numsSize`，因为代码表面上看起来是正确的。

3. 你的思维方式可能是"顺序推理"，但程序是"动态执行"

- 你可能习惯于在**代码层面**思考，而不是在**运行层面**思考。
- 例如你可能这样想：

> “`for` 循环是从 `root` 开始，`i < root + k`，没问题！”

但你忽略了：

- `root` 可能是 `0`
- `k` 可能比 `numsSize` 还大
- `malloc(k+1)` 可能分配的空间不足

这些变量在运行时可能和你想象的不一样，导致意外的 bug。

如何减少对调试的依赖？

虽然调试很重要，但你可以通过以下方法**提前发现问题**，减少对调试的依赖：

1. 增加边界检查

不要**假设**所有索引都合法，而是**明确检查边界**：

```c
for (int i = root, j = 0; i < root + k && i < numsSize; i++, j++) {
    Kth->data[j] = nums[i]; // 只有在合法范围内才访问 nums[i]
}
```

这样即使 `k` 过大，也不会导致 `nums[i]` 访问越界。

---

2. 使用 `assert` 进行运行时检查

在 `malloc` 之后检查是否成功分配：

```c
Kth->data = (int*)malloc(sizeof(int) * k);
assert(Kth->data != NULL); // 如果分配失败，直接终止程序
```

在访问数组时添加 `assert`：

```c
assert(i < numsSize); // 确保 i 没有越界
```

这样可以**在错误发生时立刻暴露，而不是等到奇怪的 bug 出现再调试**。

---

3. 启用编译器的警告和工具

- **开启 `-Wall -Wextra`**（如果使用 GCC 或 Clang）

```sh
gcc -Wall -Wextra -g your_code.c -o your_program
```

这样会检测很多**潜在问题**，比如未初始化变量、可能的越界等。

- **使用 `AddressSanitizer`**

```sh
gcc -fsanitize=address -g your_code.c -o your_program
./your_program
```

这个工具会在**访问越界时立即报错**，不像普通运行那样"默默地"出错。

---

4. 画图或者手动演算

调试前，试着自己在纸上**模拟变量的变化**：

- `root` 取什么值？
- `root + k` 取什么值？
- `i` 取值范围是什么？
- `numsSize` 可能比 `k` 小吗？ 这样可以提前发现潜在问题，而不是等到出错后才调试。

总结

为什么你会依赖调试？

1. **C 语言不提供数组越界检查**，访问非法内存不会立刻报错，除非调试工具介入。
2. **代码逻辑的动态执行方式可能和你想象的不一样**，比如 `k > numsSize` 的情况你可能没考虑到。
3. **没有显式的边界检查**，导致意外的访问越界。

如何减少对调试的依赖？ ✅ **增加边界检查**，比如 `i < numsSize`。
✅ **使用 `assert` 检测异常情况**。
✅ **启用 `-Wall -Wextra` 和 `AddressSanitizer`** 来自动检测错误。
✅ **手动推演变量的变化**，确保逻辑正确。

如果你开始在写代码时**提前考虑边界情况**，你就可以在**还没运行**代码时就发现 bug，从而减少对调试的依赖！
🚀