arXiv:2601.00979v1 [cs.DS] 2 Jan 2026

欧几里得算法中的循环置换与余数和的成本

Valentin Blomer Kai-UweBux*

2026年1月2日

摘要

我们讨论对 Gries–Mills 分块交换方案的一种改进，该方案用于原地旋转，平均成本为每个元素移动 1.85 次，最坏情况下的性能仍为每个元素移动 3 次。平均情况的分析依赖于欧几里得算法中余数和的渐近行为。

1 引言

将长度为 n 的数组旋转 k 个位置，会对其中的元素执行循环置换，就像切开一副有 n 张牌的牌，拿起其中的 k 张，再把它们放回底部一样。和任何置换一样，它可以写成若干循环的乘积；而该旋转的循环分解包含 $\gcd(n,k)$ 个循环，每个循环的长度为 $\frac{n}{\gcd(n,k)}$。回想一下，阶为 m 的循环置换可以通过 $m+1$ 次移动来完成（只需额外使用一个内存单元）。因此，用这种方式执行旋转需要 $n+\gcd(n,k)$ 次移动。这种旋转方案称为 Dolphin 算法 [2, 第 4] 节。Shene [5] 已证明，这个移动次数是最优的：任何旋转算法至少都需要这么多次移动。

尽管 Dolphin 算法已达到最优，但在现代机器上运行缓慢，尤其是当 n 较大时。该算法访问内存的顺序非常混乱；对于非常大的数组，它甚至会接近每次内存访问都发生缓存未命中的程度。因此，我们有必要寻找内存

*两位作者感谢德国研究联合会（DFG，德国研究联合会）通过 CRC TRR358 项目（项目编号 491392403）提供的支持。

局部性更好的旋转算法。Gries–Mills [2], 曾讨论过两种此类方案，当时尚未意识到内存局部性的需求：一种是将旋转实现为三次逆序操作的组合，另一种是在每一步交换长度相同的片段，采用递归策略。逆序方案需要 $n - O(1)$ 次交换，而递归块交换策略需要 $n - \gcd(n,\ k)$ 次交换。因此，这两种算法都需要约 $3n$ 次移动；后一种情况至少适用于“典型”的 k。

Van den Hoven [1] 指出，三重反转过程中的某些换位可以重新组合成由三个或两个换位构成的乘积，从而使旋转成为由 4-循环、3-循环和换位构成的乘积。三个换位需要九次移动，而一个 4-循环只需五次移动，因此用一个 4-循环替换三个换位，能显著减少移动次数。由此得到的三元旋转基本只需 $2n$ 次移动，性能显著优于三重反转。

我们介绍一种替代三元旋转的方案：块循环方案。

定理 A. 给定一个长度为 n的数组，块循环旋转算法在最坏情况下需要$3n$ 次移动。对于平均移动次数 $\rho(n)$，其中左段的长度 k 在0到 n之间均匀分布，我们有

$$\rho(n) = Dn + \mathrm{O}(n^{\frac{1}{2}+\varepsilon})$$

对每个 $\varepsilon > 0$均成立。常数 D 约为1.85。

该算法具有良好的内存局部性，采用顺序内存访问模式，并且在实验中表现出色。

值得注意的是，三重反转和三元旋转会暂时（局部地）颠倒数组中各项的顺序。另一方面，块交换和块循环方案会保留块内元素的顺序。因此，这些算法在运行时间对底层数据类型的依赖性方面呈现出明显差异。更多信息请参见第 5 节。特别是，块循环算法似乎尤其适用于小字母表上的长字旋转。

我们在第 2 节介绍块循环方案。在第 3 节中，我们证明 $\rho(n)$ 随着 n 的增大而渐近线性增长；在第 4 节中，我们给出误差项的上界。这需要运用解析数论中的技巧，尤其是对某些指数和进行估计。常数 D 最终被证明是一个函数的积分，该函数具有某些自相似特征，但仍然是黎曼可积的。我们将这个常数计算为一个无穷级数，它可以用多重 zeta 值表示，并可进行数值估计。

致谢 我们感谢 Markus Nebel 和 Jens Stoye 提供有益的讨论和参考资料。我们感谢 Markus Nebel 和 Pascal Schweizer 对本文早期草稿提出意见。

2 块循环算法

块循环算法的出发点是以下块交换方案，它对 Gries 和 Mills 的方案稍作修改：他们将较短的片段（假设长度为 k）与另一端的镜像片段交换，从而将其中所有元素移至最终位置，并留下一个规模为 $n-k$ 的补余旋转问题。在我们的变体中，较短的片段与同样长度的相邻块交换。这样也有 k 个元素到达最终位置，代价是 k 次交换。区别在于，在原始的 Gries–Mills 方案中，较短片段中的元素 被移至最终位置；而在我们的变体中，较短片段中的位置 则由应在此处的元素填入。无论哪种情况，剩余问题都是在长度为 $n-k$ 的数组中进行旋转。可以通过尾递归解决，终止条件为 $k=0$ 或 $k=n$。

这种变体的交换次数与原始块交换方案完全相同。不过，它更便于将交换合并到更长的循环中。具体来说，我们对这些片段进行旋转

$$a_1,\ldots,a_k \mid a_{k+1},\ldots,a_{2k} \mid a_{2k+1},\ldots,a_{3k} \mid \ldots \mid a_{(q-1)k+1},\ldots,a_{qk}$$

进入新的顺序

$$a_{k+1},\ldots,a_{2k} \mid a_{2k+1},\ldots,a_{3k} \mid \ldots \mid a_{(q-1)k+1},\ldots,a_{qk} \mid a_1,\ldots,a_k$$

其中 $q=\lfloor \frac{n}{k} \rfloor$。注意，我们已通过 $(q+1)k$ 次移动，将 $(q-1)k$ 个元素放到了它们的最终位置；因此，问题缩减为在片段 $(a_{(q-1)k+1},\ldots,a_n)$ 中进行另一次旋转，即交换 $(a_{(q-1)k+1},\ldots,a_{qk})$ 和 $(a_{qk+1},\ldots,a_n)$。我们可以使用尾递归来解决剩余的较小问题。

对于较大的 q，块循环与 Dolphin 算法相似。为改善内存访问模式，此块旋转会以批次方式执行，每批处理最多 b 个相邻元素。图 1 展示了 $q=4$ 时这一过程的示意图。

可容纳最多 b 个元素的固定大小缓冲区空间，也可用于提前结束递归。当较短的片段（长度为 k）可以放入缓冲区时，我们便结束递归：此时，可以将整个较短片段移入缓冲区；接下来，将所有剩余的 $n-k$ 个元素移至最终位置；

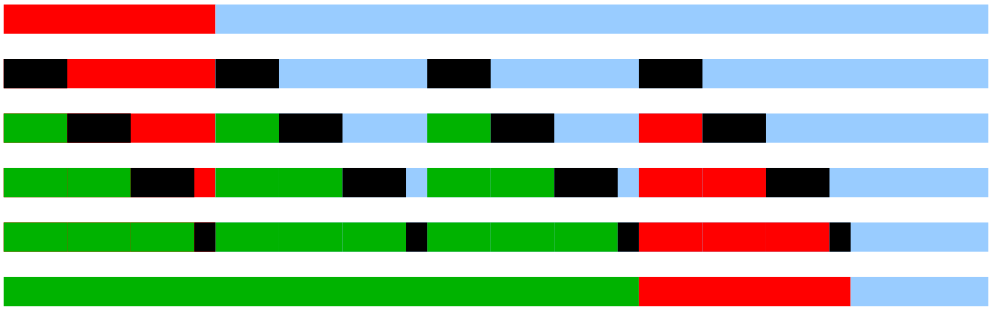

图 1：块循环算法的递归步骤：红线和蓝线分别表示要向右和向左移动的项。黑色片段当前正在进行循环块置换。此类片段的长度不得超过 b。绿线表示已到达最终位置的元素。

最后，将缓冲区中的 k 个元素移至目标位置。这样，我们只需 $n + k$ 次移动便完成了最后一次旋转。

我们不提供伪代码，而是在第 5 节中提供一个以 C++ 为泛型的参考实现，并将其与其他旋转算法进行基准测试比较。

注 1. 将 *Gries–Mills* 方案中的交换组合成长循环，并批量移动元素，这些也是 *Hashem–Li–Salah* 提出的混合旋转方案的主要要点 *[3,* 算法 *3]*。不过，他们并未进行递归，而是只执行第一步，然后使用其他任意旋转方法解决剩余问题（他们建议采用三重反转）。实际上，他们提议在一种后备旋转方法之前加上图 *1* 所示的步骤。

此外，他们没有利用对称性。因此，当左段比右段长时，他们的算法会立即转而采用后备方法，从而错失了相当可观的性能提升。伪代码似乎并未准确反映他们实际进行基准测试的内容。另一个佐证是，混合旋转的伪代码无法正确处理缓冲区大小不能被左段大小整除的情况。

3 最坏情况和平均情况的初步分析

由于块循环方案只是将块交换方案中的交换组合成更长的循环，因此显然不会比 Gries–Mills 方案执行更多次移动。因此，我们已经得到了最坏情况下的上界。

O观察 2。 块循环算法最多需要 $3n-3$ 最大公约数(n,k) 次移动 .
□

观察 3. 例如，当左段长度为8 、右段长度为13时，就会出现最坏情况。此时，递归过程如下：8:13、8:5、3:5、3:2、1:2，因此需要 $3\cdot(8+5+3+2)+4=58$ 步。更一般地，当斐波那契数为 $k=F_\kappa$, $n=F_{\kappa+2}$且满足最大公约数$(n,k)=1$ 时，渐近意义上也会出现这种情况，此时算法需要 $3\sum_{j=1}^{\kappa}F_j-2=3n-3$ 最大公约数$(n,k)-2$步。

令 $m(n,k,b)$ 表示块循环法在使用大小为 b 的缓冲区时，将长度为 n 的数组向左旋转 k 步所需的移动次数。对于 $k\le\frac{n}{2}$，我们得到

$$m(n,k,b)=\begin{cases}0 & \text{if } k=0,\\ n+k & \text{if } k\le b,\\ (\lfloor\frac{n}{k}\rfloor+1)\,k+m(n-(\lfloor\frac{n}{k}\rfloor-1)\,k,n-\lfloor\frac{n}{k}\rfloor\,k,b) & \text{otherwise.}\end{cases}\tag{1}$$

根据左旋转和右旋转的对称性，对于 $k\ge\frac{n}{2}$，有 $m(n,k,b)=m(n,n-k,b)$。

考虑一个略微简化的版本会更加方便。该版本的定义可扩展到实值变量，我们用希腊字母表示它们，以此作为视觉提示。对于 ν，$\beta>0$ 和 $\kappa\in[0,\frac{\nu}{2}]$，我们定义

$$\mu(\nu,\kappa,\beta):=\begin{cases}\nu+\kappa & \text{if } \kappa\le\beta,\\ (\lfloor\frac{\nu}{\kappa}\rfloor+1)\kappa+\mu(\nu-(\lfloor\frac{\nu}{\kappa}\rfloor-1)\kappa,\nu-\lfloor\frac{\nu}{\kappa}\rfloor\,\kappa,\beta) & \text{otherwise.}\end{cases}\tag{2}$$

根据对称性，我们将其扩展到 $\kappa\in[0,\nu]$：$\mu(\nu,\kappa,\beta)=\mu(\nu,\nu-\kappa,\beta)$。
对于 $\nu>0$ 和 $\kappa\in[0,\frac{\nu}{2}]$，令

$$\kappa':=\nu-\left\lfloor\frac{\nu}{\kappa}\right\rfloor\kappa=\kappa\left(\frac{\nu}{\kappa}-\left\lfloor\frac{\nu}{\kappa}\right\rfloor\right)=\kappa\left\{\frac{\nu}{\kappa}\right\}\in[0,\kappa]$$

and

$$\nu':=\nu-\left(\left\lfloor\frac{\nu}{\kappa}\right\rfloor-1\right)\kappa=\kappa+\kappa'=\kappa\left(1+\left\{\frac{\nu}{\kappa}\right\}\right)$$

其中我们用到了高斯映射 $\{x\}=x-\lfloor x\rfloor$。然后，我们得到

$$\mu(\nu,\kappa,\beta)-\nu=\begin{cases}\kappa & \text{if } \kappa\le\beta,\\ 2\kappa+\mu(\nu',\kappa',\beta)-\nu' & \text{otherwise.}\end{cases}\tag{3}$$

观察 4. 对于 $\nu > 0$ 和 $0 < \kappa \leq \frac{\nu}{2}$，我们有

$$\frac{\nu'}{\nu} = \frac{\kappa}{\nu}\left(1 + \left\{\frac{\nu}{\kappa}\right\}\right) \leq \frac{2}{3},$$

这对 $\kappa \leq \frac{\nu}{3}$ 显然；对于 $\frac{\nu}{3} < \kappa \leq \frac{\nu}{2}$，将 $\frac{\kappa}{\nu} = \frac{1}{2+x}$ 写成$0 \leq x \leq 1$即可看出，这会得到 $\frac{\kappa}{\nu}(1 + \{\frac{\nu}{\kappa}\}) = x(1 + \{\frac{1}{x}\}) = \frac{1+x}{2+x} \leq \frac{2}{3}$。 □

观察 4 使我们能够通过对满足 $\nu \leq \frac{3^r}{2^r}\beta$ 的最小 $r \in \mathbb{N}$ 进行归纳，证明关于 μ 和 ψ 的命题。因此，基础情形是 $\nu \leq \beta$，归纳步骤则从 $\nu \leq B$ 推进到 $\nu \leq \frac{3}{2}B$。

作为该方法的示例，我们证明最坏情况的估计在连续情形下同样成立：

观察 5。我们有 $\mu(\nu,\ \kappa,\ \beta) \leq 3\nu$。

证明。当 $\nu \leq \beta$ 时，结论显然，因为此时我们有 $\mu(\nu,\ \kappa,\ \beta) = \nu + \kappa \leq 3\nu$。当 $\nu > \beta$ 时，我们有 $\mu(\nu,\ \kappa,\ \beta) = \nu + \kappa \leq 3\nu$，如果 $\kappa \leq \beta$，并且

$$\mu(\nu, \kappa, \beta) = \left\lfloor \frac{\nu}{\kappa} + 1 \right\rfloor \kappa + \mu(\nu', \kappa', \beta) = \frac{\left\lfloor \frac{\nu}{\kappa} + 1 \right\rfloor}{\left\lfloor \frac{\nu}{\kappa} - 1 \right\rfloor} \left\lfloor \frac{\nu}{\kappa} - 1 \right\rfloor \kappa + \mu(\nu', \kappa', \beta)$$

否则。对 $n \text{in } \nu \leq \frac{3^n}{2^n}\beta$ 进行归纳法，可假设 $\mu(\nu', \kappa', \beta) \leq 3\nu'$。由于 $\frac{\lfloor \frac{\nu}{\kappa}+1 \rfloor}{\lfloor \frac{\nu}{\kappa}-1 \rfloor} \leq 3$，我们得到

$$\mu(\nu, \kappa, \beta) \leq 3 \left\lfloor \frac{\nu}{\kappa} - 1 \right\rfloor \kappa + 3\nu' = 3\nu$$

为 $\nu' + \lfloor \frac{\nu}{\kappa} - 1 \rfloor \kappa = \nu$。 □

同样的论证也能证明以下结论。

推论 6。 *1.* 函数 $\mu(\nu, \kappa, \beta)$ 关于 β 单调递减。

2. 对于 $\lambda > 0$，有 $\mu(\lambda\nu, \lambda\kappa, \lambda\beta) = \lambda\mu(\nu, \kappa, \beta)$。

3. 函数 μ 为移动次数提供上界：

$$m(n, k, b) \leq \mu(n, k, b)\,.$$

特别是，当 β 趋于 0_+ 时，$\mu(\nu,\ \kappa,\ \beta)$ 的值单调递增，但有上界 3ν。因此，我们可以取极限

$$\mu(\nu, \kappa) = \lim_{\beta \to 0_+} \mu(\nu, \kappa, \beta)\,.$$

显然，我们有 $\mu(\nu,\ \kappa,\ \beta) \le \mu(\nu,\ \kappa)$，且 $\mu(\lambda\nu,\ \lambda\kappa) = \lambda\mu(\nu,\ \kappa)$。因此，所有信息都包含在函数中

$$f(\kappa) := \mu(1, \kappa)\,,$$

它具有对称性 $f(\kappa) = f(1-\kappa)$。
我们在图2中绘制了其图像，这表明 f 具有某种自相似性。为推导自相似关系，注意对于 $0 < \kappa < \frac{1}{2}$，例如我们有 $\frac{1}{5} < \frac{1-\kappa}{4-3\kappa} < \frac{1}{4}$，由此可得

$$f\Big(\frac{1-\kappa}{4-3\kappa}\Big) = 5\frac{1-\kappa}{4-3\kappa} + \mu\Big(\frac{1}{4-3\kappa}, \frac{\kappa}{4-3\kappa}\Big) = \frac{5-5\kappa+f(\kappa)}{4-3\kappa}.$$

解出 $f(\kappa)$，得到自相似关系 $f(\kappa) = 5\kappa - 5 + (4-3\kappa) f\big(\frac{1-\kappa}{4-3\kappa}\big)$。
对范围 $\frac{1}{m+2} < \frac{1-\kappa}{m+1-m\kappa} < \frac{1}{m+1}$ 进行相同的计算，可得到自相似关系

$$f(y) = (m+2)y - (m+2) + (m+1-my) f\Big(\frac{1-y}{m+1-my}\Big) \tag{4}$$

当 $0 < \kappa < \frac{1}{2}$ 时。

定理7。函数 f 在所有无理数处连续。特别地，它的间断点至多可数，并且黎曼可积。

证明。定义（3）表明，可以定义 $\psi(\nu,\ \kappa,\ \beta) = \mu(\nu,\ \kappa,\ \beta) - \nu$，并相应地

$$\psi(\nu,\kappa) := \lim_{\beta\to 0_+} \psi(\nu,\kappa,\beta) = \mu(\nu,\kappa) - \nu \quad \text{and} \quad \psi(\kappa) := \psi(1,\kappa)\,, \tag{5}$$

然后证明 $\kappa \mapsto \psi(\kappa)$ 在任意无理数 κ 处连续。
注意递推关系

$$\psi(\nu,\kappa,\beta) = \begin{cases} \kappa & \text{if } \kappa \le \beta, \\ 2\kappa + \psi(\nu',\kappa',\beta) & \text{otherwise} \end{cases} \tag{6}$$

描述了一种用于计算 $\psi(\nu,\kappa,\beta)$ 的递归算法，并在$\kappa \le \beta$时终止。随着 β 趋近于 0_+，递归深度会增加。在极限 $\beta = 0$下，我们发现，只有当 κ 和 ν 可公度时，递归才会终止。因此，对于有理数 $\frac{\kappa}{\nu}$，我们可以将 $\psi(\nu,\kappa)$ 展开为有限和。如果 $\frac{\kappa}{\nu}$ 初始为无理数，那么在递归过程中它将始终是无理数，终止分支也永远不会执行。在这种情况下，无限递归将 $\psi(\nu,\kappa)$ 表示为一个收敛级数。

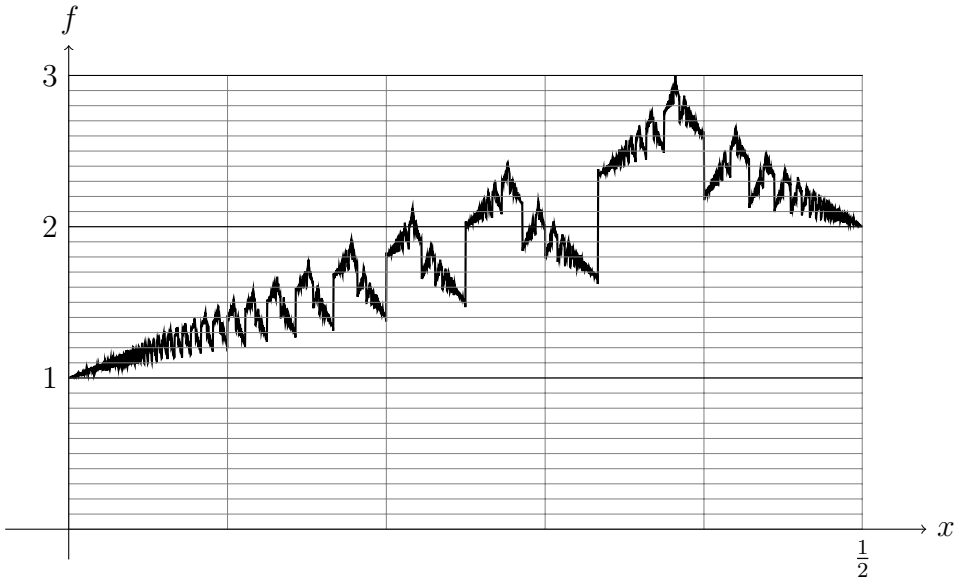

图2：函数 f 在范围 $0 \le x \le \frac{1}{2}$ 上的图像。最大值3在 $\frac{3-\sqrt{5}}{2} = 1-\phi$ 处取得，其中 ϕ 为黄金比例。令 (F_m) 表示斐波那契数列。当 $1-\phi = \lim_{m\to\infty} \frac{F_{m-1}}{F_m+F_{m-1}}$ 时，这个峰值对应于算法的最坏情况行为，如观察 3 中所述。

我们利用齐次性推导出

$$\psi(\kappa) = \begin{cases} 0 & \kappa = 0, \\ 2\kappa + \kappa\,(1 + \{\frac{1}{\kappa}\})\,\psi(\kappa\,\{\frac{1}{\kappa}\}) & \text{otherwise.} \end{cases}$$

根据定义

$$T(x) := x\left(1 + \left\{\frac{1}{x}\right\}\right), \quad S(x) := \frac{\{\frac{1}{x}\}}{1 + \{\frac{1}{x}\}}$$

其中我们形式上令 $S(0) = T(0) = 0$，因此可以去掉分类讨论，写成

$$\psi(x) = 2x + T(x)\,\psi(S(x))\,.$$

展开隐式递归可得

$$\psi(x) = 2x + 2\sum_{j=1}^{\infty} T(x)\,T(S(x))\cdots T(S^{j-1}(x))\,S^j(x)\,.$$

注意， $T(S^j(x)) \leq \frac{2}{3}$且 $S^{j+1}(x) \leq \frac{1}{2}$。由此可知，对于每个 $\varepsilon > 0$，都存在$N \in \mathbb{N}$ 使得

$$2\sum_{j>N} T(x)\,T(S(x))\cdots T(S^{j-1}(x))\,S^j(x) \leq \sum_{j>N}\left(\frac{2}{3}\right)^j \leq \varepsilon$$

在 x 上一致收敛。也就是说，定义 $\psi(x)$ 的级数一致收敛到 ψ。

现在，设 $x \in [0, \frac{1}{2}]$ 为无理数。函数 S 和 T 在每个开区间（$\frac{1}{M+1}$，$\frac{1}{M}$）上都是连续的。通过考察 N 阶迭代原像，我们发现存在一个包含 x 的开区间 I，在该区间上，部分和

$$2x + 2\sum_{j=1}^{N} T(x)\,T(S(x))\cdots T(S^{j-1}(x))\,S^j(x)$$

是连续的。

He因此， ψ 是在 x 处连续的函数列的一致极限。结论得证。 □

推论 8。 假设 X 是一个在$[0, \frac{1}{2}]$上均匀分布的实随机变量。那么， $f(X) = \psi(X) + 1$ 的分布的 m 阶矩为

$$\mathbb{E}[f(X)^m] = \frac{\int_0^{\frac{1}{2}} f(x)^m \,\mathrm{d}\,x}{\frac{1}{2}}.$$

特别地，任意阶的矩都存在。 □

现在我们可以证明，块循环算法的平均移动次数具有明确的渐近行为。我们不考虑使用较大缓冲区所带来的节省：渐近而言，任何固定大小的缓冲区都无关紧要。因此，令

$$m(n,k) = m(n,k,1)$$

表示使用大小为 1 的缓冲区，将长度为 n 的数组旋转 k 步时，块循环算法所需的移动次数。长度为 n 的数组旋转的平均移动次数为

$$\rho(n) := \frac{1}{n}\sum_{0\leq k<n} m(n,k),$$

其中假设 k 服从均匀分布。

Th定理 9。 平均移动次数 $\rho(n)$ 随 n 线性增长。更准确地说 ，

$$\lim_{n\to\infty}\frac{\rho(n)}{n} = \mathbb{E}[f(X)] = 2\int_0^{\frac{1}{2}} f(x)\,\mathrm{d}\,x$$

其中， X 是一个在 $[0, \frac{1}{2}]$上均匀分布的实随机变量。 -

在给出证明之前，我们先总结一些初步观察。下一节将对该常数进行求值。

我们区分两种移动：移入和移出缓冲区区域的移动（A 类），以及在数组内部的移动（B 类）。

观察10。 对于 $k \le \frac{n}{2}$，设(k_j, n_j) 为块循环算法第 j 次迭代中（子）问题的参数。*A* 类移动的次数显然为$2(k_1 + k_2 + k_3 + \cdots)$。我们有如下递归关系：

$$\begin{aligned} k_1 &= k, & n_1 &= n, \\ k_{i+1} &= (n_i \bmod k_i), & n_{i+1} &= k_i + k_{i+1}. \end{aligned}$$

这实现了函数 ψ 在 $\beta = 0$ 条件下的递归关系 *(6)*。因此，*A* 类移动的次数等于 $\psi(n, k) = n\psi(\frac{k}{n})$，这里采用记号(5)。– □

为便于后续参考，我们指出，线段大小序列 (k_j) 正是一次欧几里得算法运行过程中遇到的余数序列。

引理 11。 设 $k \le \frac{n}{2}$。与上文一样，设(k_j, n_j) 为块循环算法第 j 次迭代中（子）问题的参数。考虑由下式定义的序列(r_j)

$$\begin{aligned} r_1 &= k, & b_1 &= n, \\ r_{j+1} &= (b_j \bmod r_j), & b_{j+1} &= r_j \end{aligned}$$

由欧几里得算法递归定义，并在 $r_{j+1} = 0$ 时终止。

然后，对于每个 j，都有 $k_j = r_j$。特别地，

1. *The* 块循环算法在参数为 $(0$,最大公约数$(n, k))$ 的子问题上终止。
2. 由于 $k_1 + k_2 + \cdots = r_1 + r_2 + \cdots$，我们有

$$n\psi\left(\frac{k}{n}\right) = \psi(n, k) = 2\left(r_1 + r_2 + \cdots\right).$$

证明。 沿着 (r_j, b_j) 和 (k_j, n_j) 的递归进行直接归纳可得， $r_j = k_j$ 以及 $b_j \cong n_j$ 模 r_j。 □

观察 12. 每次 *B* 型移动，恰有一个元素到达其最终位置。因此，这种移动最多有 n 次。事实上，除递归结束时遇到将大小为 最大公约数(k, n) 的数组旋转 0 位的子问题外，每个元素都必须恰好经历一次 *B* 型移动。因此，我们恰好进行了 $n -$ 最大公约数(k, n) 次 *B* 型移动。 □

定理 *9* 的证明. 移动次数 $m(n,k)$ 是 A 类计数与 B 类计数之和。因此，对于 $k \le \frac{n}{2}$，我们有

$$m(n,k) = n - \gcd(k,n) + n\psi\Big(\frac{k}{n}\Big) = nf\Big(\frac{k}{n}\Big) - \gcd(k,n). \tag{7}$$

由于 $\gcd(k,n) = \gcd(n-k,n)$ 且 $m(k,n) = m(n-k,n)$，该公式对所有 $k < n$ 均成立；并且我们有

$$2\int_0^{\frac{1}{2}} f(x)\,\mathrm{d}\,x = \int_0^1 f(x)\,\mathrm{d}\,x = \lim_{n\to\infty}\frac{1}{n}\sum_{0\le k<n} f\Big(\frac{k}{n}\Big).$$

因此，我们可以计算

$$\begin{aligned}\lim_{n\to\infty}\frac{\rho(n)}{n} &= \lim_{n\to\infty}\frac{1}{n^2}\sum_{0\le k<n} m(k,n)\\ &= \lim_{n\to\infty}\frac{1}{n^2}\sum_{0\le k<n} nf\Big(\frac{k}{n}\Big) - \lim_{n\to\infty}\frac{1}{n^2}\sum_{0\le k<n}\gcd(k,n)\\ &= 2\int_0^{\frac{1}{2}} f(x)\,\mathrm{d}\,x.\end{aligned}$$

证明完毕。 □

注 13。 类似地，函数 $f_\beta(\kappa) := \mu(1,\ \kappa,\ \beta)$ 估计了算法在使用大小为 βn 的缓冲区时每个元素的移动次数。如图 *3* 所示，即使底部段和顶部段都无法装入缓冲区，算法也能利用缓冲区。一旦递归导向一个能够受益于使用缓冲区的子问题，算法就会开始使用缓冲区。

4 块循环方案的渐近成本

引理 11 将移动次数与欧几里得算法中的余数之和联系起来。因此，我们可以利用一些解析数论来分析算法的平均成本（以移动次数衡量）。具体而言，我们将定理 9 细化如下：

定理 14。 平均成本随 n 线性增长。更确切地说，对于每个$\varepsilon > 0$，我们有

$$\rho(n) = Dn + \mathrm{O}(n^{\frac{1}{2}+\varepsilon})$$

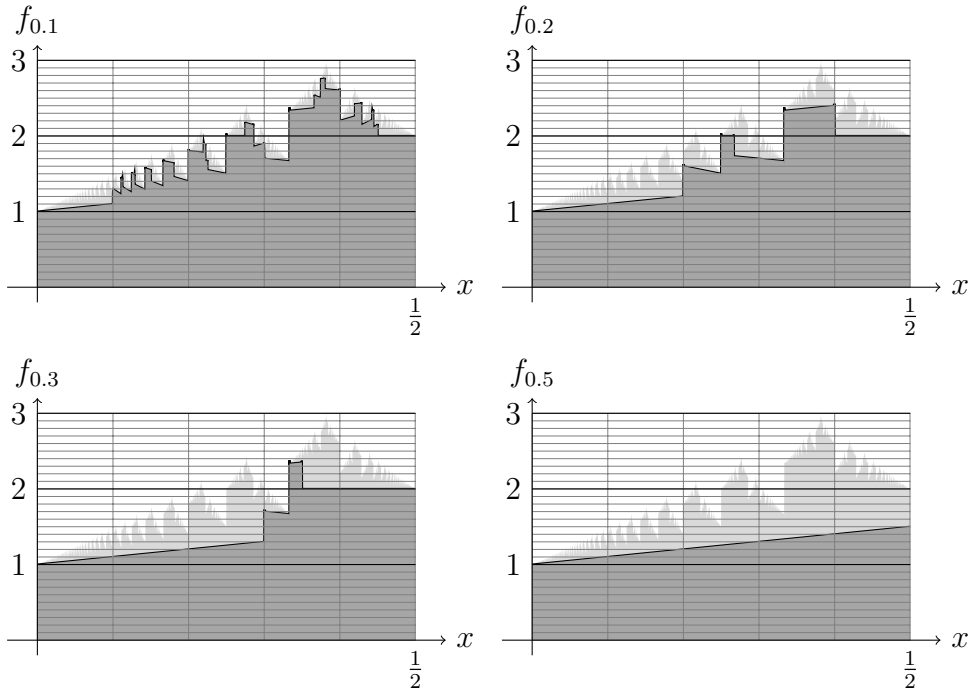

图 3：不同缓冲区大小下的预期成本。缓冲区大小为 10% 时，预期成本约为每个元素移动 1.73 次；为 20% 时，约为每个元素移动 1.6 次；为 30% 时，约为每个元素移动 1.49 次。缓冲区大小为 50% 且底部段较短时（由对称性可假设如此），该片段可放入缓冲区，因此我们得到每个元素预期移动 1.25 次。

其中

$$D \approx 1.85$$

是显式已知的。具体而言，我们将 D表示为 $D = 1+4C$，其中 C通过以下两种方式计算：作为下文 (13)和 (15)中的快速收敛双重无穷级数。

注15。 特别是，每个元素的预期移动次数小于2。因此，在平均情况下，该算法所需的移动次数少于三元旋转方案。正如我们所见，渐近分布的高阶矩也存在。然而，由于最坏情况下每个元素需要移动3 次，高阶矩在实际应用中的意义不大。数值近似得出的标准差约为每个元素移动0.50 次。

本节余下部分将全部用于证明定理 14，并确定常数 D。

按照解析数论中的惯例，我们将采用 $\varepsilon > 0$ 约定，即所有涉及 ε 的陈述都意味着对每个足够小的 $\varepsilon > 0$ 均成立。此类陈述的典型形式是 $\sum_{d|n} 1 = O(n^{\varepsilon})$。

理解 $m(n,\ k)$ 的主要难点在于求和

$$\overline{m}(n,k) = r_1 + r_2 + \cdots \tag{8}$$

欧几里得算法中出现的余数。首先，考虑互素数 $n =:\ r_0$ 和 $k =:\ r_1$的欧几里得算法运行过程：

$$\begin{aligned} r_0 &= q_0 r_1 + r_2 \\ r_1 &= q_1 r_2 + r_3 \\ &\vdots \\ r_{t-1} &= q_{t-1} r_t + 1 \\ r_t &= q_t \\ r_{t+1} &= 1 \\ r_{t+2} &= 0. \end{aligned}$$

注意 $q_t = r_t \geq 2$。我们定义如下简写：

$$[] = 1 \qquad [c_0] = c_0 \qquad [c_0, c_1] = c_0 c_1 + 1 \qquad [c_0, \ldots, c_t] = c_0[c_1, \ldots, c_t] + [c_2, \ldots, c_t]$$

并通过归纳法得到 $r_j = [q_j, \ldots, q_t]$。特别地，我们得到 $n = [q_0, \ldots, q_t]$ 和

$$\overline{m}(n,k) = [q_1, \ldots, q_t] + [q_2, \ldots, q_t] + \cdots + [q_{t-1}, q_t] + [q_t] + 1. \tag{9}$$

归纳法还表明

$$\begin{pmatrix} [c_0, \ldots, c_t] & [c_0, \ldots, c_{t-1}] \\ [c_1, \ldots, c_t] & [c_1, \ldots, c_{t-1}] \end{pmatrix} = \begin{pmatrix} c_0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} c_1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} c_t & 1 \\ 1 & 0 \end{pmatrix}.$$

因此

$$\begin{pmatrix} [c_0, \ldots, c_t] & [c_0, \ldots, c_{t-1}] \\ [c_1, \ldots, c_t] & [c_1, \ldots, c_{t-1}] \end{pmatrix} = \begin{pmatrix} [c_0, \ldots, c_s] & [c_0, \ldots, c_{s-1}] \\ [c_1, \ldots, c_s] & [c_1, \ldots, c_{s-1}] \end{pmatrix} \begin{pmatrix} [c_{s+1}, \ldots, c_t] & [c_{s+1}, \ldots, c_{t-1}] \\ [c_{s+2}, \ldots, c_t] & [c_{s+2}, \ldots, c_{t-1}] \end{pmatrix}$$

尤其是

$$[c_0, \ldots, c_t] = [c_0, \ldots, c_s][c_{s+1}, \ldots, c_t] + [c_0, \ldots, c_{s-1}][c_{s+2}, \ldots, c_t].$$

Heilbronn [4, 第 93] 页证明了

$$x = [c_0, \ldots, c_s],$$

$$x' = [c_{s+1}, \dots, c_t],$$
$$y = [c_0, \dots, c_{s-1}],$$
$$y' = [c_{s+2}, \dots, c_t]$$

在这些集合之间建立了一一对应

$$\{(s, (c_0, \dots, c_t)) \mid 0 \le s < t\,,\, n = [c_0, \dots, c_t],\, 2 \le c_0,\, 2 \le c_t\}$$

and

$$\{(x, x', y, y') \mid x > y \ge 1,\, x' > y' \ge 1,\, 1 = \gcd(x, y) = \gcd(x', y'),\, n = xx' + yy'\}.$$

根据（9），我们首先推得

$$\sum_{\substack{1 \le k \le \frac{n}{2} \\ 1 = \gcd(n,k)}} \overline{m}(k, n) = \sum_{\substack{x > y \ge 1 \\ x' > y' \ge 1 \\ 1 = \gcd(x,y) \\ 1 = \gcd(x',y') \\ n = xx' + yy'}} x' + \sum_{\substack{1 \le k \le \frac{n}{2} \\ 1 = \gcd(n,k)}} \gcd(n, k)$$

然后

$$\sum_{1 \le k \le \frac{n}{2}} \overline{m}(k, n) = \sum_{\substack{x > y \ge 1 \\ x' > y' \ge 1 \\ 1 = \gcd(x,y) \\ n = xx' + yy'}} x' + \sum_{1 \le k \le \frac{n}{2}} \gcd(n, k). \tag{10}$$

我们将右侧的第一项称为 $R(n)$。第二项为 $O(n^{1+\varepsilon})$，因此

$$\sum_{1 \le k \le \frac{n}{2}} \overline{m}(k, n) = R(n) + \mathrm{O}(n^{1+\varepsilon}). \tag{11}$$

我们希望去除 $R(n)$上的最大公因数条件，为此我们定义

$$G(n) := \sum_{\substack{x > y \ge 1 \\ x' > y' \ge 1 \\ n = xx' + yy'}} x' = \sum_{\substack{x > y \ge 1 \\ x' > y' \ge 1 \\ n = xx' + yy'}} x = \frac{1}{2} \sum_{\substack{x > y \ge 1 \\ x' > y' \ge 1 \\ n = xx' + yy'}} (x' + x)\,.$$

现在，我们可以根据这些二元组的最大公约数对（x, y）进行排序（该最大公约数自动整除 n），从而得出 $G(n) = \sum_{d|n} R(d)$。于是得到

$$R(n) = \sum_{d|n} \mu(d) G\left(\frac{n}{d}\right) \tag{12}$$

通过 M¨obius 反演。

我们接下来分析 G。注意到

$$G(n)=\frac{1}{2}\sum_{\substack{x>y\geq 1\\ x'>y'\geq 1\\ n=xx'+yy'}}(x'+x)=\sum_{\substack{x>y\geq 1\\ x'>y'\geq 1\\ n=xx'+yy'\\ x'>x}}(x'+x)+\mathrm{O}(n^{1+\varepsilon})$$

$$=\sum_{d|n}\sum_{\substack{x>y\geq 1\\ x'>y'\geq 1\\ n=xx'+yy'\\ x'>x\\ d=\gcd(x,y)}}(x'+x)+\mathrm{O}(n^{1+\varepsilon})=\sum_{d|n}\sum_{\substack{x>y\geq 1\\ x'>y'\geq 1\\ \frac{n}{d}=xx'+yy'\\ x'>dx\\ 1=\gcd(x,y)}}(x'+dx)+\mathrm{O}(n^{1+\varepsilon}).$$

重写求和条件后得到 $x'=\frac{n}{dx}-\frac{yy'}{x}$，其中假设$x$整除$\frac{n}{d}-yy'$。那么，$x'>y'$等价于$\frac{n}{d(y+x)}>y'$。类似地，$x'>dx$ 等价于$\frac{n}{d}-yy'>dx^2$。因此：

$$G(n)=\sum_{d|n}\sum_{\substack{x>y\geq 1\\ \frac{n}{d(x+y)}>y'\geq 1\\ \frac{n}{d}\equiv yy' \bmod x\\ \frac{n}{d}-yy'>dx^2\\ 1=\gcd(x,y)}}\left(\frac{n}{dx}-\frac{yy'}{x}+dx\right)+\mathrm{O}(n^{1+\varepsilon}).$$

暂且固定 d、x和 y，并考虑以下关于

$$G(n,d,x,y):=\sum_{\substack{\frac{n}{d(x+y)}>y'\geq 1\\ \frac{n}{d}\equiv yy' \bmod x\\ \frac{n}{d}-yy'>dx^2}}\left(\frac{n}{dx}-\frac{yy'}{x}+dx\right)$$

y'的求和。由于 x 和 y 互素，同余式 $\frac{n}{d}\equiv yy'$ 模 x 确定了 y' 模 x的余数。我们用 a表示一个代表元，使得 $\frac{n}{d}\equiv yy'$模 x 等价于 $y'\equiv a$ 模 x。

此外，我们发现 $\frac{n}{d}-yy'>dx^2$ 转化为 $y'<\frac{n}{dy}-\frac{dx^2}{y}=\frac{n-d^2x^2}{dy}$。因此，$y'$的上界为 $Y=Y(n,\ d,\ x,\ y):=\min\left(\frac{n}{d(x+y)},\ \frac{n-d^2x^2}{dy}\right)$。

因此

$$G(n,d,x,y)=\sum_{\substack{y'\equiv a \bmod x\\ 1\leq y'<Y}}\left(\frac{n}{dx}-\frac{yy'}{x}+dx\right)=\sum_{\substack{y'\equiv a \bmod x\\ 1\leq y'<Y}}(A+By')$$

其中

$$A=\frac{n}{dx}+dx\qquad\text{and}\qquad B=-\frac{y}{x}.$$

我们希望去掉繁琐的求和条件 $y' \equiv a \bmod x$。利用特征

$$\begin{aligned} \mathrm{e}_x : \mathbb{Z}/x\mathbb{Z} &\longrightarrow \mathbb{C} \\ b &\mapsto \mathrm{e}^{\frac{2\pi \mathrm{i}}{x}b} \end{aligned}$$

我们改写为

$$\begin{aligned} G(n,d,x,y) &= \frac{1}{x}\sum_{b\in\mathbb{Z}/x\mathbb{Z}} \mathrm{e}_x(ba) \sum_{1\le y'<Y} (A+By')\mathrm{e}_x(-by') \\ &= \frac{1}{x}\sum_{1\le y'<Y} (A+By') + \frac{1}{x}\sum_{b\neq 0} \mathrm{e}_x(ba) \sum_{1\le y'<Y} (A+By')\mathrm{e}_x(-by'). \end{aligned}$$

主项将来自右侧的第一项。注意到

$$\left| \frac{1}{x}(YA+\frac{Y^2}{2}B) \ - \ \frac{1}{x}\sum_{1\le y'<Y}(A+By') \right| \le |A|+|BY|\,.$$

因此我们有 $G(n,\ d,\ x,\ y) = G_1(n,\ d,\ x,\ y) + G_2(n,\ d,\ x,\ y) + G_3(n,\ d,\ x,\ y)$，其中

$$\begin{aligned} G_1(n,d,x,y) &= \frac{1}{x}(YA+\frac{Y^2}{2}B), \qquad |G_2(n,d,x,y)| \le |A|+|BY|\,, \\ G_3(n,d,x,y) &= \frac{1}{x}\sum_{b\neq 0} \mathrm{e}_x(ba) \sum_{1\le y'<Y} (A+By')\mathrm{e}_x(-by'). \end{aligned}$$

Thus

$$G(n) = G_1(n) + G_2(n) + G_3(n)$$

其中

$$G_k(n) = \sum_{d|n} \sum_{\substack{x>y\ge 1 \\ 1=\gcd(x,y) \\ 1<Y(n,d,x,y)}} G_k(n,d,x,y)\,.$$

引理 16。我们有 $G_2(n) = \mathrm{O}(n^{\frac{3}{2}+\varepsilon})$。

证明。注意到 $Y \ge 1$ 意味着 $n > d^2x^2$。也就是说，$x < \sqrt{\frac{n}{d}}$ 是 $Y \ge 1$的必要条件。此外，我们有 $Y \le \frac{n}{dx}$。因此

$$\begin{aligned} |G_2(n)| &\le \sum_{d|n} \sum_{1\le y<x<\frac{\sqrt{n}}{d}} |A|+|BY| \le \sum_{d|n} \sum_{1\le y<x<\frac{\sqrt{n}}{d}} \frac{n}{dx} + dx + \frac{yn}{dx^2} \\ &\le \sum_{d|n} \sum_{1\le x<\frac{\sqrt{n}}{d}} \frac{n}{d} + dx^2 + \frac{n}{d} \le \sum_{d|n} \frac{\sqrt{n}}{d}\left(\frac{n}{d}+\frac{n}{d}+\frac{n}{d}\right) = \mathrm{O}(n^{\frac{3}{2}+\varepsilon}). \end{aligned}$$

□

观察 17。 复平面中的单位圆上有两种显然的度量：使用弧长的内蕴度量，以及由嵌入诱导的欧几里得度量。这两种度量是双 *Lipschitz* 等价的。更确切地说，对于 $\phi \in [-\pi, \pi]$，我们有

$$\frac{2}{\pi}\left|\phi\right| \leq \left|\mathrm{e}^{\mathrm{i}\phi} - 1\right| \leq \left|\phi\right|,$$

so

$$\left|\sum_{j=k}^{l-1} \mathrm{e}^{\mathrm{i}\phi j}\right| = \left|\frac{\mathrm{e}^{\mathrm{i}\phi k} - \mathrm{e}^{\mathrm{i}\phi l}}{1 - \mathrm{e}^{\mathrm{i}\phi}}\right| \leq \frac{2}{\left|\mathrm{e}^{\mathrm{i}\phi} - 1\right|} \leq \left|\frac{\pi}{\phi}\right|$$

对于 $\phi \neq 0$。同样，再次对于 $\phi \neq 0$，由

$$\sum_{j=1}^{l-1} j\mathrm{e}^{\mathrm{i}\phi j} = \sum_{k=1}^{l-1}\sum_{j=k}^{l-1} \mathrm{e}^{\mathrm{i}\phi j} = \frac{\mathrm{e}^{\mathrm{i}\phi} - \mathrm{e}^{\mathrm{i}\phi l}}{(1 - \mathrm{e}^{\mathrm{i}\phi})^2} - \frac{(l-1)\,\mathrm{e}^{\mathrm{i}\phi l}}{1 - \mathrm{e}^{\mathrm{i}\phi}}$$

我们可以推得

$$\left|\sum_{j=1}^{l-1} j\mathrm{e}^{\mathrm{i}\phi j}\right| \leq \frac{\pi^2}{2\left|\phi\right|^2} + \frac{(l-1)\,\pi}{2\left|\phi\right|}.$$

引理 18. 我们有 $G_3(n) = \mathrm{O}(n^{\frac{3}{2}+\varepsilon})$。

证明。 回顾一下，

$$G_3(n, d, x, y) = \frac{1}{x}\sum_{b \neq 0} \mathrm{e}_x(ba) \sum_{1 \leq y' < Y} (A + By')\mathrm{e}_x(-by').$$

在 $-\frac{x}{2}$ 和 $\frac{x}{2}$ 之间选取 $0 \not\equiv b \in \mathbb{Z}/x\mathbb{Z}$ 的代表元，由观察 17 可得

$$\begin{aligned}\left|\frac{1}{x}\sum_{1 \leq y' < Y} (A + By')\,\mathrm{e}^{-2\pi\mathrm{i}\frac{b}{x}y'}\right| &\leq \frac{1}{|x|}\Big(A\pi\frac{|x|}{2\pi\left|b\right|} + |B|\,\frac{\pi^2}{2}\left(\frac{x}{2\pi\left|b\right|}\right)^2 + |B|\,\frac{Y\pi}{2}\frac{|x|}{2\pi\left|b\right|}\Big) \\ &\leq \Big(\frac{n}{d|x|} + d|x|\Big)\,\frac{1}{2\left|b\right|} + \frac{y}{8\left|b\right|^2} + \frac{n}{d\,(|x| + y)}\frac{y}{4\left|b\right|\left|x\right|}.\end{aligned}$$

因此

$$\begin{aligned}|G_3(n, d, x, y)| &\leq 2\sum_{b=1}^{x/2}\Big(\frac{n}{dx} + dx\Big)\,\frac{1}{2\left|b\right|} + 2\sum_{b=1}^{x/2}\frac{y}{8\left|b\right|^2} + 2\sum_{b=1}^{x/2}\frac{n}{d\,(x+y)}\frac{y}{4\left|b\right|x} \\ &\leq \Big(\frac{n}{dx} + dx\Big)\ln(x) + \frac{\pi^2}{24}y + \frac{n}{d\,(x+y)}\frac{y}{2x}\ln(x).\end{aligned}$$

由 $Y(n,d,x,y) \le \frac{\sqrt{x}}{d}$ 可得

$$|G_3(n)| \le \sum_{d|n} \sum_{1\le y<x\le \frac{\sqrt{n}}{d}} |G_3(n,d,x,y)|.$$

根据 $|G_3(n,\ d,\ x,\ y)|$ 的上界对总和进行拆分，可得以下估计：

$$\sum_{d|n} \sum_{1\le y<x\le \frac{\sqrt{n}}{d}} \left(\frac{n}{dx}+dx\right)\ln(x) \le \sum_{d|n} \sum_{1\le x\le \frac{\sqrt{n}}{d}} \left(\frac{n}{d}+\frac{n}{d}\right)\ln(x) = \mathrm{O}(n^{\frac{3}{2}+\varepsilon}),$$

$$\sum_{d|n} \sum_{1\le y<x\le \frac{\sqrt{n}}{d}} \frac{\pi^2}{24} y \le \sum_{d|n} \frac{n}{d^2}\frac{\pi^2}{24}\frac{\sqrt{n}}{d} = \mathrm{O}(n^{\frac{3}{2}+\varepsilon}),$$

$$\sum_{d|n} \sum_{1\le y<x\le \frac{\sqrt{n}}{d}} \frac{n}{d\,(x+y)}\frac{y}{2x}\ln(x) \le \sum_{d|n} \sum_{1\le x\le \frac{\sqrt{n}}{d}} \frac{nx}{2d\,(x+1)}\ln(x) = \mathrm{O}(n^{\frac{3}{2}+\varepsilon}).$$

结论得证。 □

引理 19. 我们有

$$G_1(n) = Cn^2 \sum_{d|n} \frac{1}{d^2} + \mathrm{O}(n^{\frac{3}{2}+\varepsilon})$$

其中

$$C = \sum_{\substack{x>y\ge 1\\ 1=\gcd(x,y)}} \frac{2x+y}{2x^2\,(x+y)^2}. \tag{13}$$

证明. 回顾

$$G_1(n) = \sum_{d|n} \sum_{\substack{x>y\ge 1\\ 1=\gcd(x,y)\\ 1<Y(n,d,x,y)}} G_1(n,d,x,y) = \sum_{d|n} \sum_{\substack{x>y\ge 1\\ 1=\gcd(x,y)\\ 1<Y(n,d,x,y)}} \frac{1}{x}\left(AY + B\frac{Y^2}{2}\right)$$

并注意到

$$Y(n,d,x,y) = \min\Big(\frac{n}{d\,(x+y)}, \frac{n-d^2x^2}{dy}\Big) = \begin{cases} \frac{n}{d(x+y)} & \text{if } y \le \frac{n}{d^2x} - x \\ \frac{n-d^2x^2}{dy} & \text{if } y \ge \frac{n}{d^2x} - x. \end{cases}$$

我们将求和拆分

$$G_1(n) = \sum_{d|n} G_1^{<}(n,d) + \sum_{d|n} G_1^{\ge}(n,d)$$

with

$$G_1^{<}(n,d) := \sum_{\substack{x>y\geq 1\\ 1=\gcd(x,y)\\ y<\frac{n}{d^2x}-x\\ 1<\frac{n}{d(x+y)}}} \frac{1}{x}\left(AY+B\frac{Y^2}{2}\right), \quad G_1^{\geq}(n,d) := \sum_{\substack{x>y\geq 1\\ 1=\gcd(x,y)\\ y\geq\frac{n}{d^2x}-x\\ 1<\frac{n-d^2x^2}{dy}}} \frac{1}{x}\left(AY+B\frac{Y^2}{2}\right).$$

展开 A 和 B 的定义，经过直接计算可得

$$G_1^{\geq}(n,d) = \sum_{\substack{x>y\geq 1\\ 1=\gcd(x,y)\\ y\geq\frac{n}{d^2x}-x\\ 1<\frac{n-d^2x^2}{dy}}} \left(\frac{n^2}{2d^2x^2y}+\frac{n}{y}-\frac{3d^2x^2}{2y}\right).$$

由于 x 和 y 的上界分别为 $\sqrt{n/d}$ 和 $2x^2$ 的下界为 $\frac{n}{d^2}$，我们得到 $G_1^{\geq}(n,d) = \Theta(n^{\frac{3}{2}})$，因此

$$\sum_{d|n} G_1^{\geq}(n,d) = \mathrm{O}(n^{\frac{3}{2}+\varepsilon}).$$

为了分析 $G_1^{<}(n,\ d)$，我们首先简化求和条件。由 $y<\frac{n}{d^2x}-x$，我们先推得 $x+y<\frac{n}{d^2x}$，再推得 $dx<\frac{n}{d(x+y)}$ 特别是，鉴于 $x>y\geq 1$ 且注意到 $x^2\leq n/d^2$，条件 $1<\frac{n}{d(x+y)}$ 是多余的。因此

$$\begin{aligned}
G_1^{<}(n,d) &= \sum_{\substack{x>y\geq 1\\ 1=\gcd(x,y)\\ y<\frac{n}{d^2x}-x}} \frac{1}{x}\left(A\frac{n}{d(x+y)}+\frac{B}{2}\left(\frac{n}{d(x+y)}\right)^2\right)\\
&= \sum_{\substack{x>y\geq 1\\ 1=\gcd(x,y)\\ y<\frac{n}{d^2x}-x}} \left(\frac{n^2}{d^2x^2(x+y)}+\frac{n}{(x+y)}-\frac{n^2y}{2x^2d^2(x+y)^2}\right)\\
&= \sum_{\substack{x>y\geq 1\\ 1=\gcd(x,y)\\ y<\frac{n}{d^2x}-x}} \left(\frac{n^2}{d^2x^2(x+y)}-\frac{n^2y}{2x^2d^2(x+y)^2}\right)+\mathrm{O}(n^{\frac{3}{2}})\\
&= \frac{n^2}{d^2}\sum_{\substack{x>y\geq 1\\ 1=\gcd(x,y)\\ y<\frac{n}{d^2x}-x}} \left(\frac{2x+y}{2x^2(x+y)^2}\right)+\mathrm{O}(n^{\frac{3}{2}})
\end{aligned}$$

and

$$\sum_{d|n} G_1^{<}(n,d) = \sum_{d|n} \frac{n^2}{d^2} \sum_{\substack{x>y\geq 1 \\ 1=\gcd(x,y) \\ y<\frac{n}{d^2x}-x}} \left(\frac{2x+y}{2x^2(x+y)^2} \right) + \mathrm{O}(n^{\frac{3}{2}+\varepsilon}). \tag{14}$$

我们想去掉截断项 $y < \frac{n}{d^2x} - x$，因此对尾项进行估计。注意，$y \geq \frac{n}{d^2x} - x$ 与 $x > y$ 结合可推出 $2d^2x^2 > n$，即 $x > \frac{\sqrt{n}}{\sqrt{2}d}$。因此我们有

$$\sum_{d|n} \frac{n^2}{d^2} \sum_{\substack{x>y\geq 1 \\ 1=\gcd(x,y) \\ y\geq \frac{n}{d^2x}-x}} \frac{2x+y}{2x^2(x+y)^2} \leq \sum_{d|n} \frac{n^2}{d^2} \sum_{\substack{x>\frac{\sqrt{n}}{\sqrt{2}d} \\ 1\leq y<x}} \frac{1}{x^3} \leq \sum_{d|n} n^2 \sum_{x>\frac{\sqrt{n}}{\sqrt{2}d}} \frac{1}{d^2x^2}$$

$$\leq \sum_{d|n} n^2 \sum_{k>\frac{\sqrt{n}}{\sqrt{2}}} \frac{1}{k^2} = \mathrm{O}(n^{\frac{3}{2}+\varepsilon}).$$

结合前面的估计，可得

$$G_1(n) = \sum_{d|n} \frac{n^2}{d^2} \sum_{\substack{x>y\geq 1 \\ 1=\gcd(x,y)}} \frac{2x+y}{2x^2(x+y)^2} + \mathrm{O}(n^{\frac{3}{2}+\varepsilon})$$

如所声称。 □

定理 *14* 的证明。回顾一下，$G(n) = G_1(n) + G_2(n) + G_3(n)$。根据引理 16、引理 18 和引理 19，我们推得

$$G(n) = Cn^2 \sum_{d|n} \frac{1}{d^2} + \mathrm{O}(n^{\frac{3}{2}+\varepsilon}), \quad C = \sum_{\substack{x>y\geq 1 \\ 1=\gcd(x,y)}} \frac{2x+y}{2x^2(x+y)^2}.$$

只需回顾式 (11) 和 (12)，并注意到

$$\sum_{d|n} d^2 = \sum_{f|n} \frac{n^2}{f^2} = n^2 \sum_{f|n} \frac{1}{f^2}$$

由 M¨obius 反演可得

$$n^2 = \sum_{d|n} \mu(d) \frac{n^2}{d^2} \sum_{f|\frac{n}{d}} \frac{1}{f^2},$$

因此

$$\sum_{1\leq k\leq \frac{n}{2}} \overline{m}(k,n) = \sum_{d|n} \mu(d) G\left(\frac{n}{d}\right) + \mathrm{O}(n^{1+\varepsilon}) = Cn^2 + \mathrm{O}(n^{\frac{3}{2}+\varepsilon}).$$

利用（8）、引理 11 和（7），我们得出

$$\sum_{k\leq\frac{n}{2}} m(n,k) = \frac{n^2}{2} + 2\sum_{k\leq\frac{n}{2}} \overline{m}(n,k) + \mathrm{O}(n^{1+\varepsilon}) = \left(\frac{1}{2} + 2C\right) n^2 + \mathrm{O}(n^{\frac{3}{2}+\varepsilon}).$$

由于 $m(n,\ k) = m(n,\ n-k)$，我们最终得到

$$\rho(n) = \frac{2}{n}\sum_{k\leq\frac{n}{2}} m(n,k) = (1+4C)\, n + \mathrm{O}(n^{\frac{1}{2}+\varepsilon}),$$

这就以常数 $D = 1 + 4C$证明了该结论。 □

注 20. 对数对 (n,k) 执行欧几里得算法，若 $n > k \geq \frac{n}{2}$，则会得到第一个余数 $-k$，随后产生的余数与对$(n,n-k)$执行算法时相同。因此，在这种情况下：

$$\overline{m}(n,k) = k + \overline{m}(n,n-k)$$

由此观察可知，在范围 $1 \leq k \leq n$ 上取平均的欧几里得算法余数和为

$$\left(\frac{3}{8} + 2C\right) n + \mathrm{O}(n^{\frac{1}{2}+\varepsilon}).$$

注 21. 我们可以用一种更容易理解的方式改写常数 C。我们有

$$\begin{aligned}\zeta(3)\, C &= \left(\sum_{d} \frac{1}{d^3}\right) \sum_{\substack{x>y\geq 1\\ 1=\gcd(x,y)}} \frac{1}{2y}\left(\frac{1}{x^2} - \frac{1}{(x+y)^2}\right)\\ &= \sum_{x>y\geq 1} \frac{1}{2y}\left(\frac{1}{x^2} - \frac{1}{(x+y)^2}\right) = \frac{1}{2}\zeta(3) - \frac{1}{2}\sum_{x>y\geq 1} \frac{1}{y\,(x+y)^2},\end{aligned}$$

其中在倒数第二步中，我们使用了多重 *zeta* 值的欧拉公式，参见例如 *[6,* 第 *509* 页，末尾的公式，其中 $k = 3$*]*。因此，我们得到

$$C = \frac{1}{2} - \frac{1}{2\zeta(3)} \sum_{x>y\geq 1} \frac{1}{y\,(x+y)^2}. \tag{15}$$

我们注意到，无限和的任何截断都能给出 C 的上界，因此也能给出 D的上界。

5 实现与基准测试

为了比较块循环方案与块交换算法、三重反转和三元旋转的运行时间，我们给出了每种算法的实现 C++，它们都是作用于随机访问迭代器的泛型算法。

块循环算法的实现

在我们的实现中，我们使用一个较大的缓冲区（由常量 `BUFFER_SIZE` 控制）来结束递归，并使用一个较小的缓冲区（由常量 `BATCH_SIZE` 控制）来批处理循环置换。在基准测试中，较大的缓冲区可容纳 256 个数组元素，而较小的缓冲区为 32 字节。

此外，当左段和右段大小相同时，我们还会提前退出，此时只需调用 `std::swap_ranges()`。

1 常量表达式 static 结构体 block_cycle_t { 2 template <typename 迭代器 >3 迭代器 函数调用运算符 (迭代器 from, 迭代器 middle, 迭代器 to) const { 4 std::size_t l_size =middle - from; 5 std::size_t r_size =to - middle; 6 using value_type =std::decay_t<decltype(*from) >; 7 常量表达式 std::size_t const buffer_size =BUFFER_SIZE; 8 常量表达式 std::size_t const batch_size =9 std::max(std::size_t(1), BATCH_SIZE / sizeof(value_type)); 10 static_ assert(batch_size <=buffer_size); 11 thread_local std::数组<value_type, buffer_size > buffer; 12 while (l_size > 0&& r_size > 0) { 13 if (l_size <r_size) { 14 if (l_size <=buffer_ size) { 15 std::move(from, middle, buffer.begin()); 16 std::move(middle, to, from); 17 std::move(buffer.begin(), buffer.begin() + l_size, to - l_size); 18 break; 19 } 20 std::size_t i = 0; 21 for (; i <l_size - batch_size; i +=batch_size) { 22 std::move(from + i, from +i + batch_size, buffer.begin()); 23 std::size_t shift = 0; 24 for (; shift <=r_size - l_size; shift += l_size) { 25 std::move(from +i +shift +l_size, 26 from +i +shift +l_size +batch_size, 27 from +i +shift); 28 } 29 std::move(buffer.begin(), 30 buffer.begin() +batch_size, 31 fro m +i +shift); 32 } 33 std::size_t n_remaining =l_size - i; 34 std::size_t shift = 0; 35 std::move(from +i, from +i +n_remaining, buffer.begin()); 36 for (; shift +l_size <=r_ size; shift +=l_size) { 37 std::move(from + i + shift + l_size, 38 from +i +shift +l_size + n_remaining, 39 from +i +shift); 40 } 41 std::move(buffer.begin(), 42 buffer.begin() +n_ remaining,

```
43 from  +i  +shift ); 44 from +=shift; 45 middle  +=shift; 46 r_size -=shift; 47 继续; 48 }
49 if ( r_size  <l_size ) { 50 if ( r_size  <=buffer_size ) { 51 std::move( middle, to,
buffer.begin() ); 52 std::move_backward( from, middle, to ); 53 std::move(
buffer.begin(),buffer.begin()  +  r_size, from ); 54 跳出; 55 } 56 std::size_t i  =  0; 57 for
( ; i  <r_size - batch_size; i  +=batch_size ) { 58 std::move( middle  +i, middle  +i  +batch_
size, buffer.begin() ); 59 std::size_t shift  =  0; 60 for ( ; shift  <=l_size - r_size; shift  +=
r_size ) { 61 std::move( middle  +i - shift - r_size, 62 middle  +i -shift - r_size  +batch_
size, 63 middle  +i - shift ); 64 } 65 std::move( buffer.begin(), 66 buffer.begin()  +batch_
size, 67 middle  +i - shift ); 68 } 69 std::size_t n_remaining  =r_size - i; 70 std::move(
middle  +i, middle  +i  +n_remaining, buffer.begin() ); 71 std::size_t shift  =  0; 72 for ( ;
shift  <=l_size - r_size; shift  +=r_size ) { 73 std::move( middle  +i - shift - r_size, 74
middle  +i -shift - r_size  +n_remaining, 75 middle  +i - shift ); 76 } 77 std::move(
buffer.begin(), 78 buffer.begin()  +n_remaining, 79 middle  +i - shift ); 80 to -=shift; 81
middle -=shift; 82 l_size -=shift; 83 继续; 84 } 85 std::swap_ranges( from, middle,
middle ); 86 跳出; 87 } 88 return ( from  +r_size ); 89 } 90 } block_cycle;
```

块交换算法的实现

1 constexprstaticstructblock_swap_t{ 2 template <typenameIterator >3 Iteratoroperator()(Iteratorfrom,Iteratormiddle,Iteratorto)const{ 4 std::size_tr_size = to-middle; 5 std::size_tl_size =middle-from; 6 while (true){ 7 if(r_size == 0) { 8 return(from); 9 } 10 if(l_size == 0) { 11 return(to); 12 } 13 if(r_size >=l_size){ 14 std::swap_ranges(from,middle,middle); 15 from =middle; 16 middle +=l_size; 17 r_size-=l_size; 18 continue; 19 } 20 { // 0 <r_size <l_size 21 std::swap_ranges(middle,to,middle-r_size); 22 to -=r_size; 23 middle-=r_size; 24 l_size-=r_size; 25 continue; 26 } 27 } 28 } 29 } 块交换;

三重反转与三元旋转的实现

三重反转的基本方案很简单。

1 常量表达式 static 结构体 basic_triple_reverse_t { 2 public: 3 template <typename 迭代器 >4 迭代器 函数调用运算符 () (迭代器 from, 迭代器 middle, 迭代器 to) const { 5 std::size_t const r_size =to - middle; 6 std::reverse(from, middle); 7 std::reverse(middle, to); 8 std::reverse(from, to); 9 return (from +r_size); 10 } 11 } basic_triple_reverse; 三位旋转更为复杂。 1 常量表达式 static 结构体 unchecked_trinity_t { 2 template < typename A, typename B, typename C >3 void swap3 (A & a, B & b, C & c) const { 4 auto temp =std::move(a);

```
5 a = std::move( b ); 6 b = std::move( c ); 7 c = std::move( temp ); 8 } 9 template < typename A, typename B, typename C, typename D >10 void swap4 ( A & a, B & b, C & c, D & d ) const { 11 auto temp =std::move( a ); 12 a = std::move( b ); 13 b = std::move( c ); 14 c = std::move( d ); 15 d = std::move( temp ); 16 } 17 template <typename 迭代器 >18 迭代器 函数调用运算符 () ( 迭代器 from, 迭代器 middle, 迭代器 to ) const { 19 std::size_t const l_size =middle - from; 20 std::size_t const r_size =to - middle; 21 迭代器 iter_l_beg =from; 22 迭代器 iter_l_end = middle; 23 迭代器 iter_r_beg =middle; 24 迭代器 iter_r_end =to; 25 if ( l_size >=r_size ) { 26 for ( std::size_t counter =r_size / 2; counter -- > 0; ) { 27 -- iter_l_end; -- iter_r_end; 28 swap4( *iter_l_end, *iter_l_beg, *iter_r_beg, *iter_r_end );29 ++ iter_l_beg; ++iter_r_beg; 30 } 31 for ( std::size_t counter = ( iter_l_end - iter_l_beg ) / 2; counter -- > 0;) { 32 -- iter_l_end; -- iter_r_end; 33 swap3( *iter_l_end, *iter_l_beg, *iter_r_end );34 ++iter_l_beg; 35 } 36 } else { 37 for ( std::size_t counter =l_size / 2; counter -- > 0; ) { 38 -- iter_l_end; -- iter_r_end; 39 swap4( *iter_l_end, *iter_l_beg, *iter_r_beg, *iter_r_end );40 ++ iter_l_beg; ++iter_r_beg; 41 } 42 for ( std::size_t counter = ( iter_r_end - iter_r_beg ) / 2; counter -- > 0;) { 43 -- iter_r_end; 44 swap3( *iter_l_beg, *iter_r_beg, *iter_r_end );45 ++ iter_l_beg; ++iter_r_beg; 46 } 47 } 48 std::reverse( iter_l_beg, iter_r_end ); 49 return ( from +r_size ); 50 } 51 } unchecked_trinity;
```

必须防范 $k = 0$ 和 $k = n$ 这两种情况。我们采用泛型实现。

```
1 常量表达式静态 结构体检查中_t{2 私有:
```

3 模板 <类型名旋转 >4 结构体 工作器{ 5 旋转 旋转; 6 模板 <类型名迭代器 >7 迭代器运算符 ()(迭代器from,迭代器middle,迭代器to)常量{ 8 如果(middle ==from ||middle ==to){ 9 返回 (from +(to-middle)); 10 } 11 返回(旋转(from,middle, to)); 12 } 13 }; 14 公有: 15 模板 <类型名旋转 >16 常量表达式工作器<旋转 >运算符 []()(旋转rotate)常量{ 17 返回 {rotate}; 18 } 19 } 检查中;

对于三重反转，廉价检查可以避免一次代价高昂的操作。接下来，我们定义：

1 常量表达式静态自动三位一体 =检查[未检查_三位一体];2 常量表达式静态自动三重_反转 =检查[基本_三重_反转];

缓冲

小幅旋转可以像块循环算法中提到的提前退出策略一样，借助缓冲区来实现。我们也可以用通用的方式实现这一点：

1 constexprstatic结构体缓冲_t{ 2 private: 3 template <typenameRotate >4 结构体 工作器 { 5 Rotate 旋转; 6 template <typename迭代器 >7 迭代器operator()(迭代器from,迭代器 middle,迭代器to)const{ 8 std::size_tl_size =middle-from; 9 std::size_tr_size =to-middle; 10 迭代器result =from +r_size; 11 if(middle ==from ||middle ==to){ 12 return(result); 13 } 14 constexprstaticconstunsigned缓冲区_size =BUFFER_SIZE; 15 usingvalue_type = std::decay_t<decltype(*from) >; 16 thread_localstd::数组<value_type,缓冲区_size >缓冲区; 17 if(l_size <=缓冲区_size){ 18 std::move(from,middle,buffer.begin()); 19 std::move(middle,to,from);

20 std::move(buffer.begin(),buffer.begin() +l_size,to-l_size); 21 return(result); 22 } elseif(r_size <=缓冲区_size){ 23 std::move(middle,to,buffer.begin()); 24 std::move_backward(from,middle,to); 25 std::move(buffer.begin(),buffer.begin() +r_size,from); 26 return(result); 27 } 28 return(旋转(from,middle, to)); 29 } 30 }; 31 public: 32 template <typenameRotate >33 constexprworker<Rotate >operator [](Rotate旋转)const{ 34 return {旋转}; 35 } 36 } 缓冲;

接下来，我们定义：

1 常量表达式静态自动缓冲_三重_反转 =缓冲[基础_三重_反转];2 常量表达式静态自动缓冲_三位一体 =缓冲[未检查_三位一体];

测量时，我们使用的缓冲区大小为 256 个元素。

直接基准测试

我们对七种算法进行基准测试（分块循环、块交换、带缓冲和不带缓冲的三元旋转、带缓冲和不带缓冲的三重反转，以及 `std::rotate`），使用五种不同元素类型的数组（`int64_t`、 `int32_t`、 `short`、 `char`和`long double`），这些元素类型都可以平凡移动。

测量是在一台搭载 Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 的设备上进行的。缓存配置如下：

名称	单大小	全大小	方式	类型	级别	集合	物理行	一致性大小
L1d	32K	128K	8	数据	1	64	1	64
L1i	32K	128K	8	指令	1	64	1	64
L2	256K	1M	4	统一	2	1024	1	64
L3	8M	8M	16	路统一	3	8192	1	64

代码使用 Clang 和 GCC 在优化级别3下编译。
图4和图5中的图表展示了每字节的运行时间（纳秒），并以数组中的字节数为函数自变量；也就是说，横轴表示元素个数乘以元素类型的大小。这些图表比较了不同的算法。图6和图7中的图表展示了相同的数据，但这一次

我们比较固定算法的运行时间如何取决于序列的元素类型。

对于可平凡移动类型，在中等、大型和超大型数组中，分块循环优于其他算法。对于小型数组，结果不一，取决于数据类型的大小：旋转由 `char`组成的小型数组时，分块循环比三元旋转更快；而旋转由 `int64_t` 组成的小型数组时，三重反转似乎是最佳选择。

值得注意的是，对于可平凡移动类型，块循环算法的性能几乎不受元素类型影响。这种效果在 Clang 中最为明显，见图6；但在 GCC 中也能观察到。这种现象在块交换方案中也有一定程度的体现，但程度稍弱；其他旋转算法则没有这种表现。合理推测，这种效果可归因于两个因素：(a) 批量处理相邻元素；(b) 在所有移动操作中保留元素的相对顺序。这样一来，即使旋转由 `char`组成的数组，编译器也能利用针对长字的机器指令。如果编译器在某种逆序算法中使用了硬件寄存器的全部位宽，就需要额外代码来重新排列寄存器中的字节顺序。我们还注意到， `long double` 在若干算法中表现得有些反常。

参考文献

[1] Igor Van den Hoven。scandum/rotate。https://github.com/scandum/rotate，2021年。[2] David Gries 和 Harlan Mills。交换区段，1981年1月。[3] Mervat Hashem、Kenli Li 和 Ahmad Salah。Pi-sqrt：多核系统上原地序列旋转的新型并行实现。《集群计算》，26(1)：539–557，2023年。[4] H. Heilbronn。关于一类有限连分数平均长度的研究。载于 Paul Tur´an（编），《数论与分析（纪念 *Edmund Landau* 的论文集）》，第87–96页。Plenum，纽约，1969年。[5] Ching-Kuang Shene。两种原地数组旋转算法的分析。《计算机杂志》，40(9)：541–546，1997年。[6] Don Zagier。ζ函数的值及其应用。载于《第一届欧洲数学大会，第二卷（巴黎，*1992*年）》，《数学进展》第120卷，第497–512页。Birkh¨auser，巴塞尔，1994年。

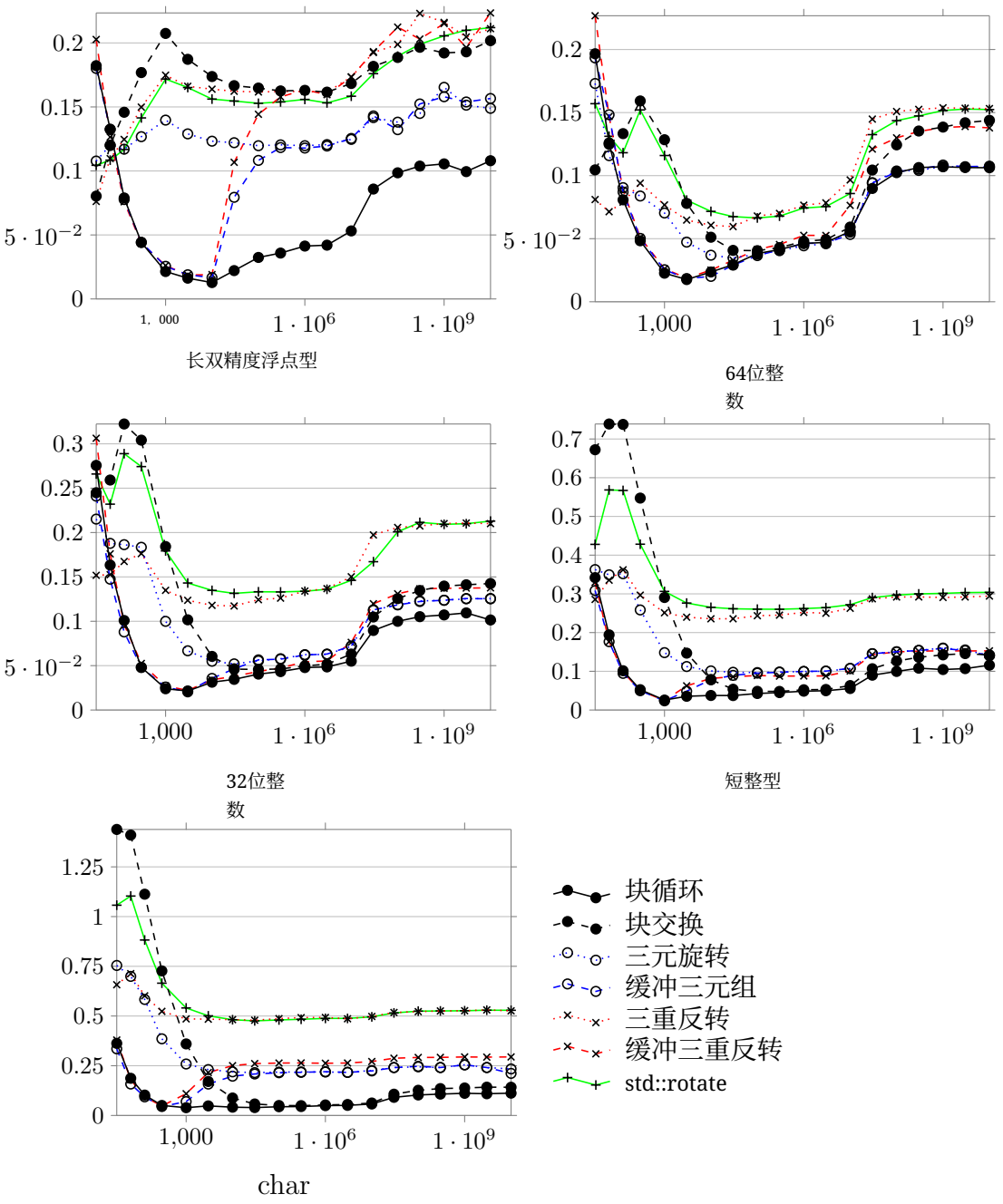

图4：针对长双精度浮点型、64位整数、32位整数、短整型和字符型数组，不同旋转算法的性能。 x轴表示数组大小，单位为字节； y轴表示运行时间，单位为每字节纳秒数。（代码使用 Clang++ 在优化级别3下编译）

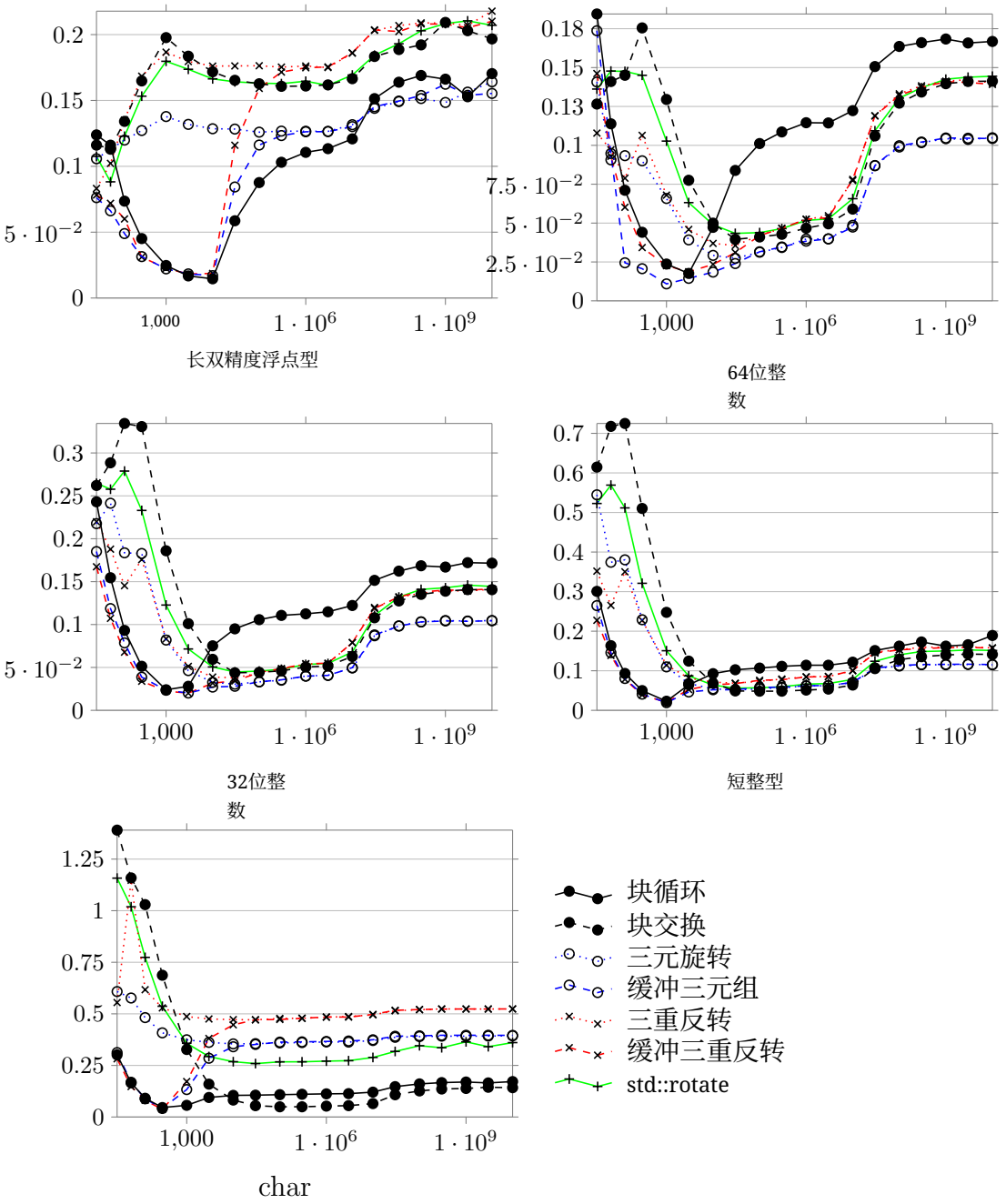

图 5：长双精度浮点型、int64、32位整数、短整型和字符型数组采用不同旋转算法时的性能。 x轴表示以字节为单位的数组大小； y轴表示以每字节纳秒数计的运行时间。（代码使用 GCC 优化级别3编译）

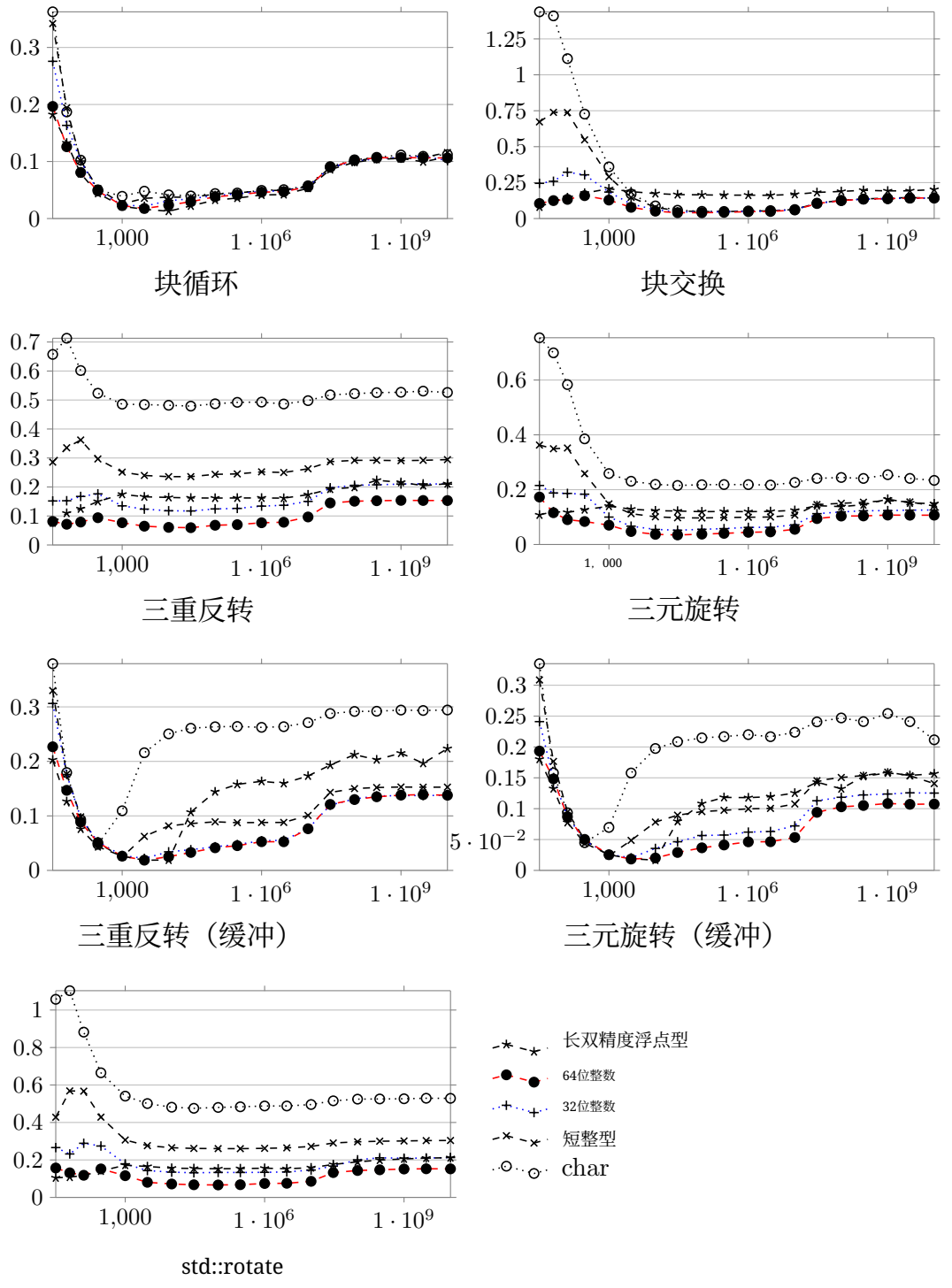

图6：不同数组旋转算法的性能。 x轴表示数组大小（以字节为单位）； y轴表示每字节的运行时间，单位为纳秒。（代码使用 Clang 的优化级别3编译）

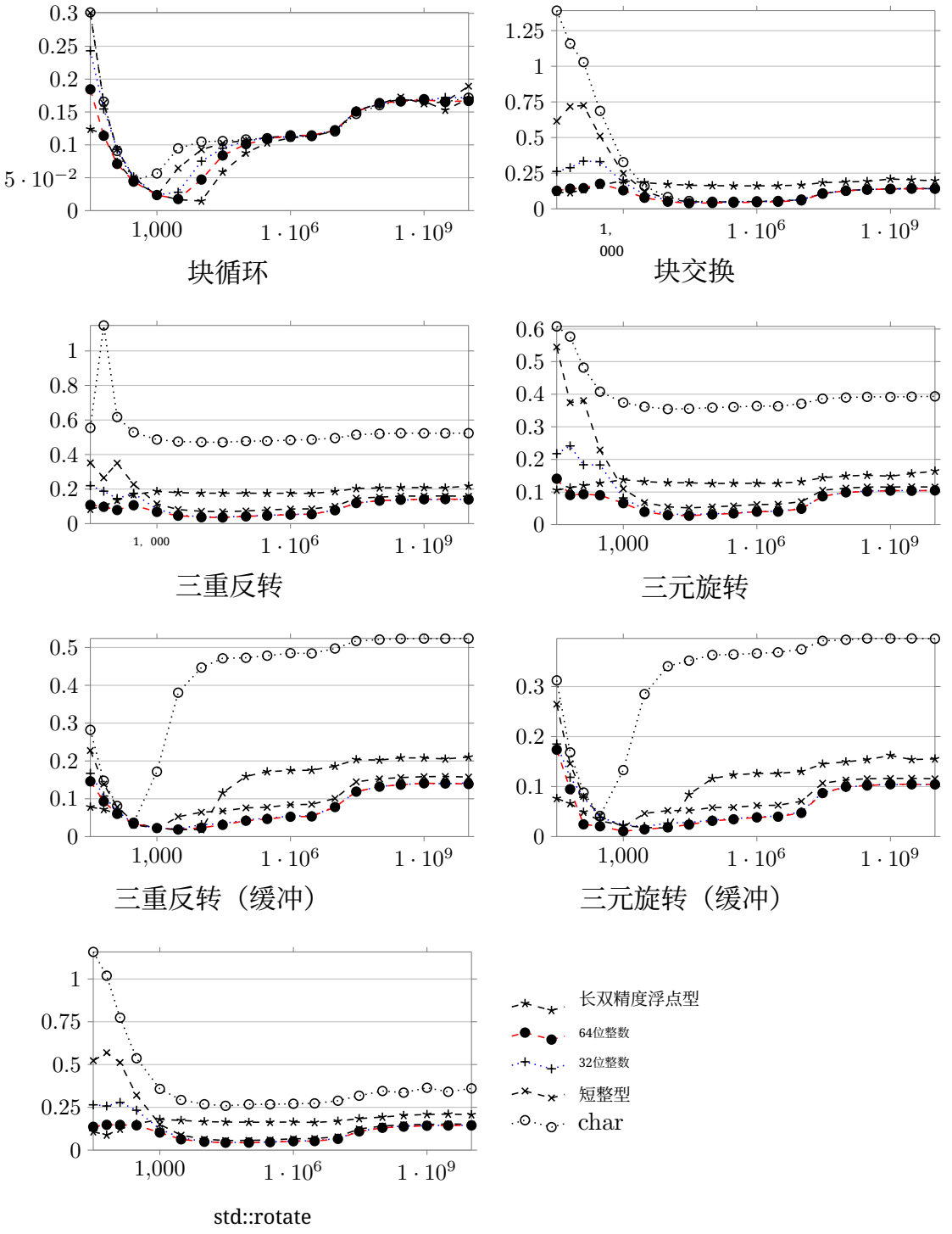

图7：不同数组旋转算法的性能。 x轴表示以字节为单位的数组大小； y轴表示每字节的运行时间，单位为纳秒。（使用 GCC 并开启优化级别3编译的代码）