Отчет 9-A1

1. Сравнение времени выполнения

В результате сравнения **времени выполнения** различных алгоритмов сортировки на разных типах данных (случайных, перевёрнутых и почти отсортированных) получены следующие результаты:

- `std::sort` показал хорошую производительность для всех типов данных, что соответствует его теоретической сложности O(n log n). Время выполнения растёт пропорционально логарифму от размера данных, что видно на графике для случайных данных и других типов.
- `mergesort_LCP` (сортировка с использованием наибольшего общего префикса) также имеет O(n log n) сложность, но из-за дополнительных операций на каждом уровне рекурсии его время выполнения немного выше, чем у `std::sort`, что подтверждается графиками.
- `quick3way` (тернарная быстрая сортировка) показал худшие результаты для некоторых типов данных, особенно для случайных, где он работает медленнее, чем другие алгоритмы. Это объясняется его худшим временем выполнения в худшем случае $O(n^2)$, несмотря на то, что в среднем он может работать быстрее, чем стандартная быстрая сортировка.
- `msd_radix` показал отличные результаты, особенно на больших данных, так как его сложность O(n·k) (где k — длина строк), и его эффективность сильно зависит от алфавита.
- `msd_hybrid` показал особенно хорошие результаты на больших данных, поскольку использует переключение на быструю сортировку при малых подмассивах, что позволяет избежать неэффективности для малых объёмов данных.

2. Сравнение количества посимвольных сравнений

Графики, показывающие **количество посимвольных сравнений**, показывают следующие результаты:

- `std::sort` использует гораздо меньше сравнений, чем другие алгоритмы для всех типов данных.
- `quick3way` требует больше сравнений в худших случаях, что подтверждается теоретической оценкой сложности для стандартной быстрой сортировки с плохим выбором опорного элемента.
- `mergesort_LCP` использует больше сравнений, поскольку для слияния необходимо вычислять наибольшие общие префиксы, что добавляет дополнительное количество сравнений.
- `msd_radix` и `msd_hybrid` требуют меньше сравнений, поскольку эти алгоритмы основаны на частичном сравнении строк по символам и используют радиусные разделения для сортировки.

3. Эмпирическая оценка сложности

Эмпирическая оценка сложности с использованием логарифмической регрессии показала следующие результаты:

- Для `std::sort` наблюдается довольно стабильный рост времени с коэффициентом $\alpha \approx 0.75$ для случайных данных, что близко к теоретической оценке $O(n \log n)$.
- Для `mergesort_LCP` коэффициент $\alpha \approx 1.05$, что немного больше, чем у стандартной сортировки, из-за дополнительных операций при обработке наибольшего общего префикса.
- `quick3way` показал заметные отклонения в зависимости от данных, с коэффициентом $\alpha \approx 1.00$ для перевёрнутых данных, что объясняется худшими случаями работы алгоритма на случайных данных.
- Для `msd_radix` и `msd_hybrid` коэффициенты α находятся в пределах 0.2 — 0.3 для случайных и перевёрнутых данных, что указывает на линейную или почти линейную сложность. Однако для почти отсортированных данных коэффициент α значительно увеличивается из-за дополнительной обработки на малых подмассивах.

4. Заключение

- **Стандартные алгоритмы сортировки**, такие как `std::sort` и `mergesort_LCP`, показали высокую эффективность с теоретической сложностью O(n log n), особенно при обработке случайных и перевёрнутых данных.
- **Специализированные алгоритмы сортировки**, такие как `msd_radix` и `msd_hybrid`, показали значительное улучшение производительности на больших данных, что делает их подходящими для задач, где размер данных значительно велик и важна производительность как по времени, так и по количеству сравнений.
- `quick3way` показал худшие результаты на случайных данных, что подтверждает теоретическую оценку сложности $O(n^2)$ в худшем случае, однако на отсортированных или почти отсортированных данных он работает достаточно эффективно.
- Для **случайных данных** лучше всего использовать `std::sort` или `mergesort_LCP`.
- Для **очень больших данных**, где важна производительность, следует использовать `msd_hybrid` или `msd_radix`, особенно если данные можно эффективно разделить на подмассивы.

Ссылки:

Ссылка на гит: sirMatras/set9a1

a1m: 321172335

a1q: 321172377

a1r: 321299623

a1rq: 321172449