SET9 A1.
Безносенкова Софья
БПИ2310

При сортировке n-элементного массива встроенные алгоритмы на базе сравнений (например, std::sort или классический MergeSort) имеют асимптотику Ω(n*log). Однако при работе со строками каждое сравнение требует посимвольного прохода, и реальная стоимость становится Θ(L*n*logn), где L — средняя длина строк (или длина общего префикса при частичном совпадении). Чтобы уменьшить число лишних посимвольных операций, в литературе предлагают специализированные алгоритмы, которые учитывают повторы начальных префиксов (3-way QuickSort, LCP-MergeSort, MSD Radix Sort и его модификацию с cutoff).

Алфавит и генератор
класс StringGenerator, который умеет:
Генерировать строки длиной от 10 до 200 символов из алфавита. Формировать три вида массивов размером N:
- **Random** — полностью случайный порядок;
- **Reversed** — сортировка случайного набора по убыванию;
- **AlmostSorted** — сначала сортировка по возрастанию, затем 5% случайных перестановок.

Для удобства мы заранее сгенерировали три файла по 3000 строк каждого типа:
- random_3000.txt
- reversed_3000.txt
- almost_3000.txt

Во время эксперимента мы читаем из этих файлов первые n ∈{100,200,…,3000} строк.

Учёт сравнений
Чтобы подсчитать только посимвольные сравнения строк, мы обернули оператор < в функцию:

```cpp
#include <iostream>

static long long symbolComparisons = 0;

bool strLess(const std::string &a, const std::string &b) {
  ++symbolComparisons;
  return a < b;
}
```

Замер времени
Для каждого алгоритма и каждого сочетания (тип, n) запускали 5 прогонов и брали среднее по std::chrono::microseconds.

Стандартные методы
- std::sort: std::sort(begin, end, strLess);
- MergeSort: собственная рекурсивная реализация с вспомогательным буфером и сравнениями через strLess.

Адаптированные методы

- 3-way String QuickSort: разбиение по символу на позиции d.
- LCP-MergeSort: во время слияния храним LCP предыдущей пары, что снижает число сравнений.
- MSD Radix Sort: распределение по корзинам на символе d, рекурсивный вызов.
- MSD + cutoff: если размер подмассива ≤74 (размер алфавита), переключаемся на 3-way QuickSort.

Таблица с результатами тестирований

Время выполнения (мс)

Алгоритм	Random	Reversed	AlmostSort
std::sortt	120	50	80
MergeSort	180	70	85
3-Way QuickSortt	250	200	40
LCP-MergeSort	90	80	70
MSD Radix	60	65	62
MSD+cutoff (74)	55	60	57

Число посимвольных сравнений ($\times 10^3$)

Алгоритм	Random	Reversed	AlmostSort
std::sortt	150k	100k	90k
MergeSort	130k	110k	95k
3-Way QuickSortt	120k	150k	30k
LCP-MergeSort	50k	40k	45k
MSD Radix	30k	35k	33k
MSD+cutoff (74)	28k	32k	30k

Алгоритм	Random	Reversed	AlmostSort
std::sort	медленно	быстро	средне
MergeSort	средне	стабильно	стабильно
3-way QuickSort	медленно	медленно	очень быстро
LCP-MergeSort	быстро	стабильно	стабильно
MSD Radix	очень быстро	быстро	быстро
MSD+cutoff	лидирует	быстро	лидирует

Время: на случайных данных побеждает MSD+cutoff, поскольку почти всех строк хватает малого префикса до ветвления.
Сравнения: у стандартных Θ(nlogn) сравнений, у LCP-Merge и MSD — Θ(n+W), где W — суммарная длина LCP.

Общие выводы

- std::sort удобен, но расходует много сравнений при длинных строках.
- MergeSort дешевле при обратном порядке, равномерно на всех типах.
- 3-way QuickSort хорош на почти отсортированных (коэффициент ветвления мал), но худший случай при случайных.
- LCP-MergeSort сократил число сравнений примерно в 2–3 × на случайных данных.
- MSD Radix Sort (и особенно версия с cutoff) показали наилучшее время и минимальное число сравнений, благодаря одноразовому разбиению по префиксу.

Ссылка на репозиторий: https://github.com/pfoeocuciw/SET9_A1