Анализ MERGE+INSERTION SORT

Анализ проводился на трех типах тестовых данных: случайных, обратно отсортированных и почти отсортированных.

Для анализа были написаны 2 класс: **ArrayGenerator**, который отвечает за генерацию 3 разных групп массивов, **SortTester** класс для удобного тестирования, который замеряет время работы сортировки и записывает данные в файл для дальнейшего анализа.
Реализации классов в файле main.cpp.

- **ArrayGenerator**:
  - Массивы, заполненные случайными целыми числами в диапазоне [0, 10000]. Для генерации использовался генератор std::mt19937 и распределение std::uniform_int_distribution для обеспечения равномерного распределения.
  - Обратно отсортированные данные: Массивы, элементы которых упорядочены по невозрастанию.
  - Почти отсортированные данные: Массивы, полученные из полностью отсортированных путем выполнения небольшого числа случайных обменов (в данной реализации 1% от размера массива).
- **SortTester**:
  - Сортировка каждого массива запускалась 5 раз и бралось среднее время выполнения, для того что бы минимизировать влияние внешних факторов.
  - Метод измерения времени: Использовалась библиотека std::chrono и high_resolution_clock для максимальной точности (мс).

Результаты замеров были сохранены csv файлы, по этим данным были простоены графики зависимости среднего времени выполнения сортировки в зависимости от размера массива.

Графики

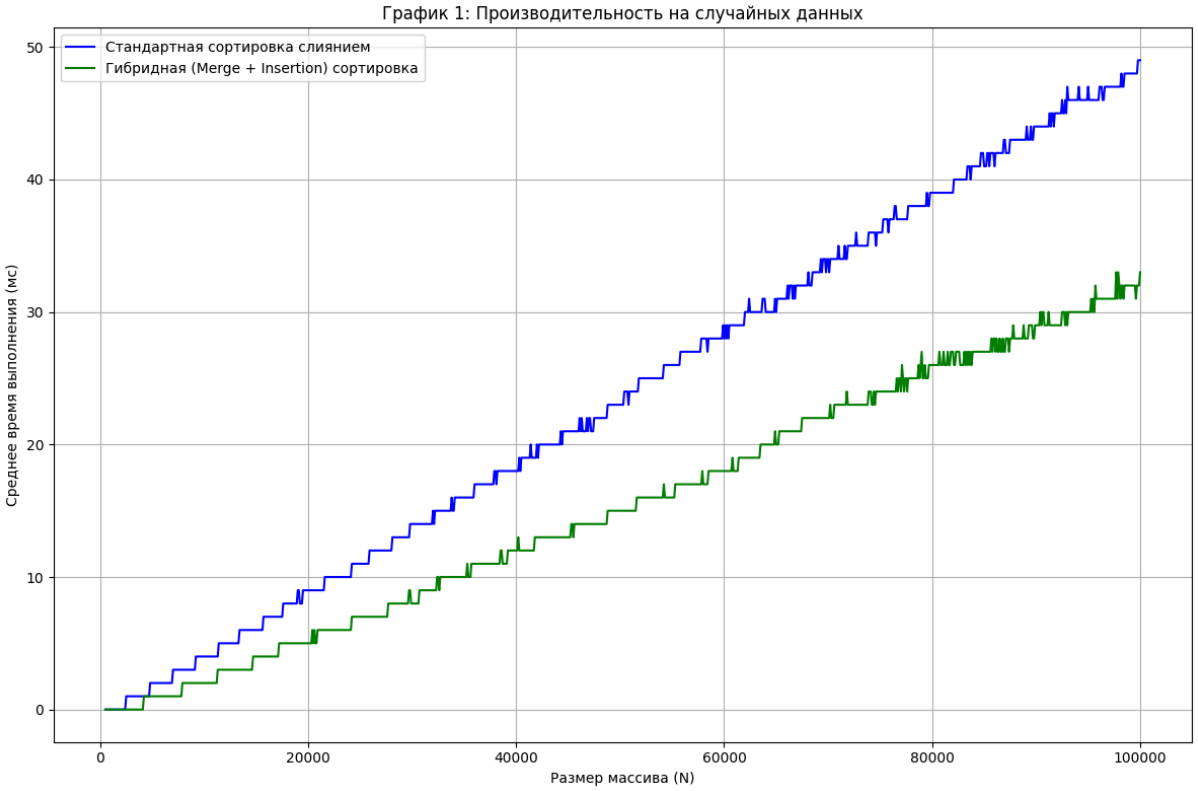

График 1: Производительность на случайных данных.

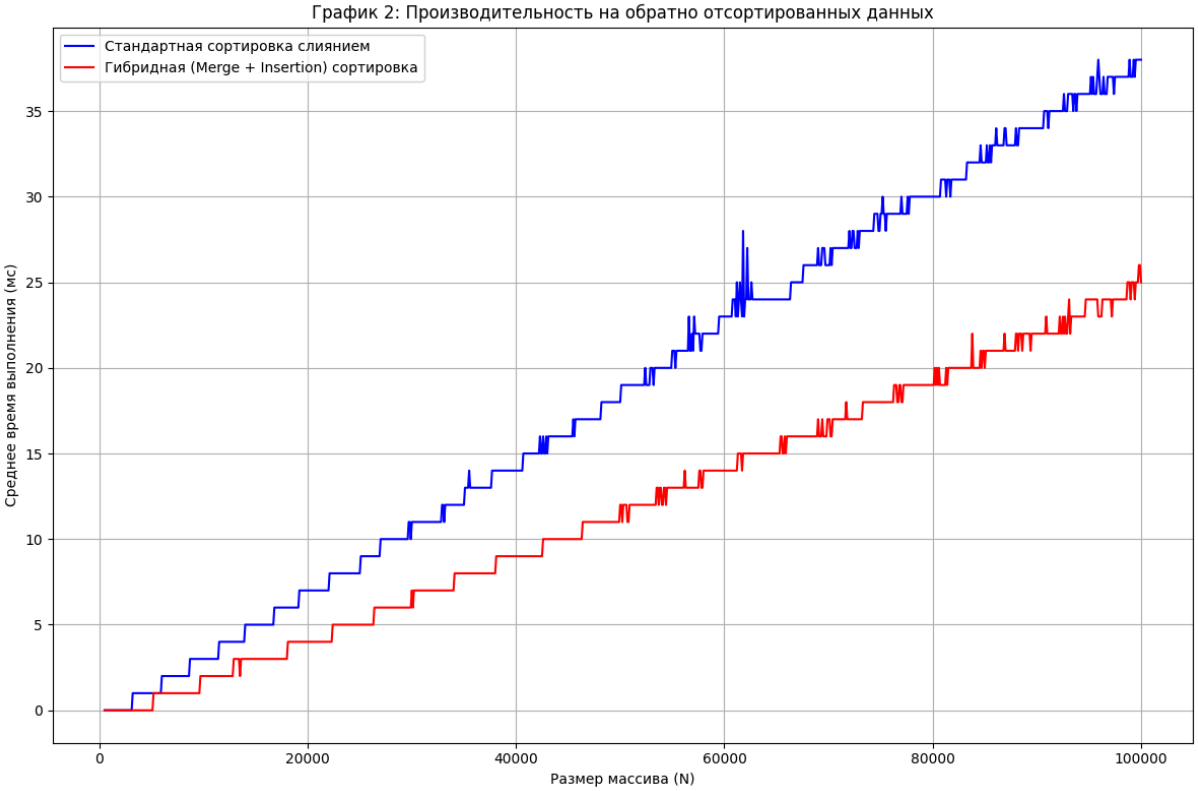

График 2: Производительность на обратно отсортированных данных.

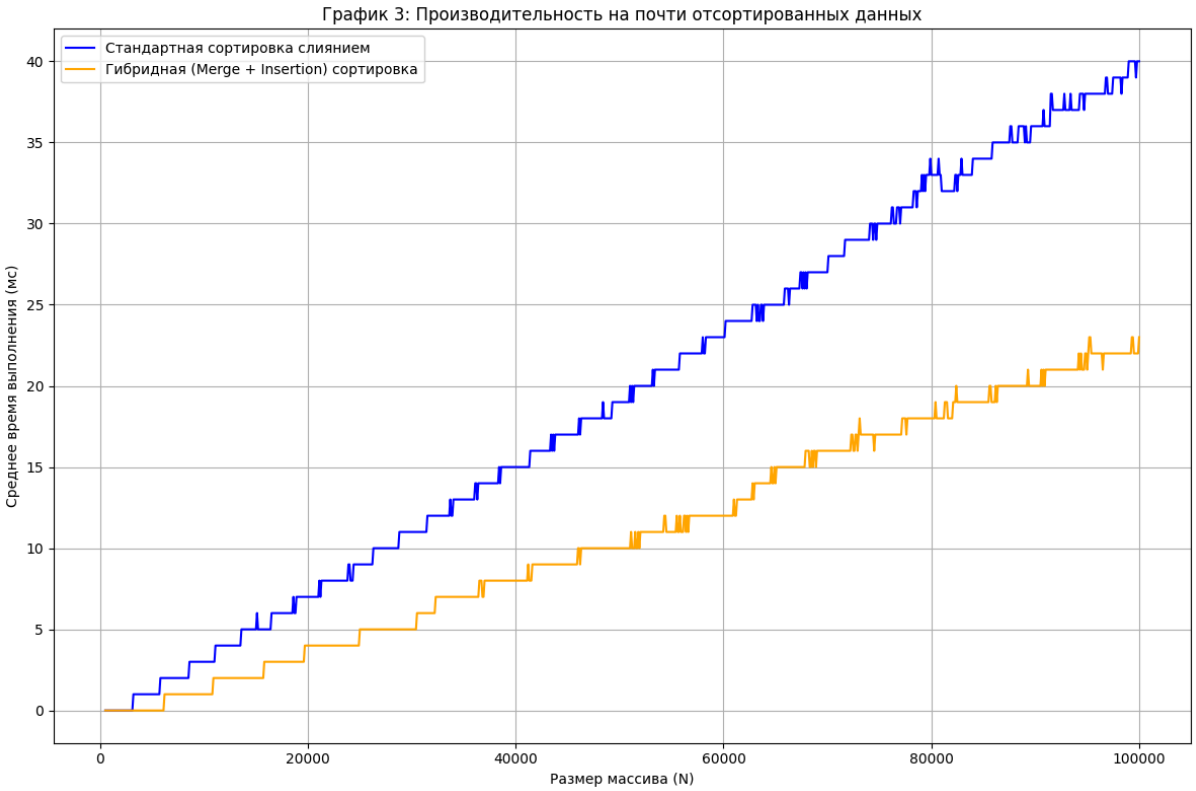

График 3: Производительность на почти отсортированных данных.

Анализ графиков

- Все прямые слегка изгибаются вверх, что соответвтует асимптотике O(NlogN), заметно это становится на графике стандартной сортировки слиянием при рандомных данных. Сам изгиб плохо виден, так как logN растет крайне медленно, для того что бы увидеть изгиб более явно можно взять N = 5′000′000.
- Можно заметить разницу в скорости работы двух алгоритмов, гибридная реализация очевидно гораздо быстрее. Однако ключевое наблюдение заключается в том, что разница во времени работы будет расти с увеличением N. Так например при N = 20′000 разница составляет около 7-8мс, при N = 100′000 разница достигает около 17-18мс. Это объясняется тем, что время работы алгоритма можно описать как $T(n) = c * N * \log(N)$, где c - константа зависящая от накладных расходов. При оптимизации с помощью сортировки вставками мы как раз оптимизируем накладные расходы при размере подмассива <= 15, тем самым можно описать время работы обычного мердж сорта как $T_1(n) = c_1 * N * \log(N)$, гибридного как $T_2(n) = c_2 *$

$N * \log(N)$. Рассмотрим их разницу $\Delta T = (c_1 - c_2) * N * \log(N)$, отсюда видно, что при увеличении N разница так же будет расти.

- Рассмотрим особый случай: На графике 3 видно, что константа c_2 для гибридного алгоритма ещё меньше, чем для остальных случаев. Это связано с тем, что на почти отсортированных массивах Insertion Sort работает за почти линейное время.

Вывод

Оптимизация алгоритма Merge Sort путем замены его на более простой алгоритм, с меньшей константой, при небольших значениях N, не меняет его асимптотическую сложность, но сушественно уменьшает константу накладных расходов. Это ведет к масштабируемому преимуществу - чем больше размер массива, тем более эффективен гибридный алгоритм Merge Sort + Insertion Sort. Этот алгоиритм так же более предпочитителен по сравнению со стандратным, так как в особых случаях (например, почти отсортированные массивы) он может показать более значительный прирост в производительности.

Ссылки на код:

- Посылка на Codeforces:

  https://dsahse25.contest.codeforces.com/group/SLdI1pWUpC/contest/647790/submission/348220090

- Ссылка на репозиторий:

  https://github.com/DarkRecklessness/Merge_Sort-vs-Merge-Sort-and-Insertion-Sort