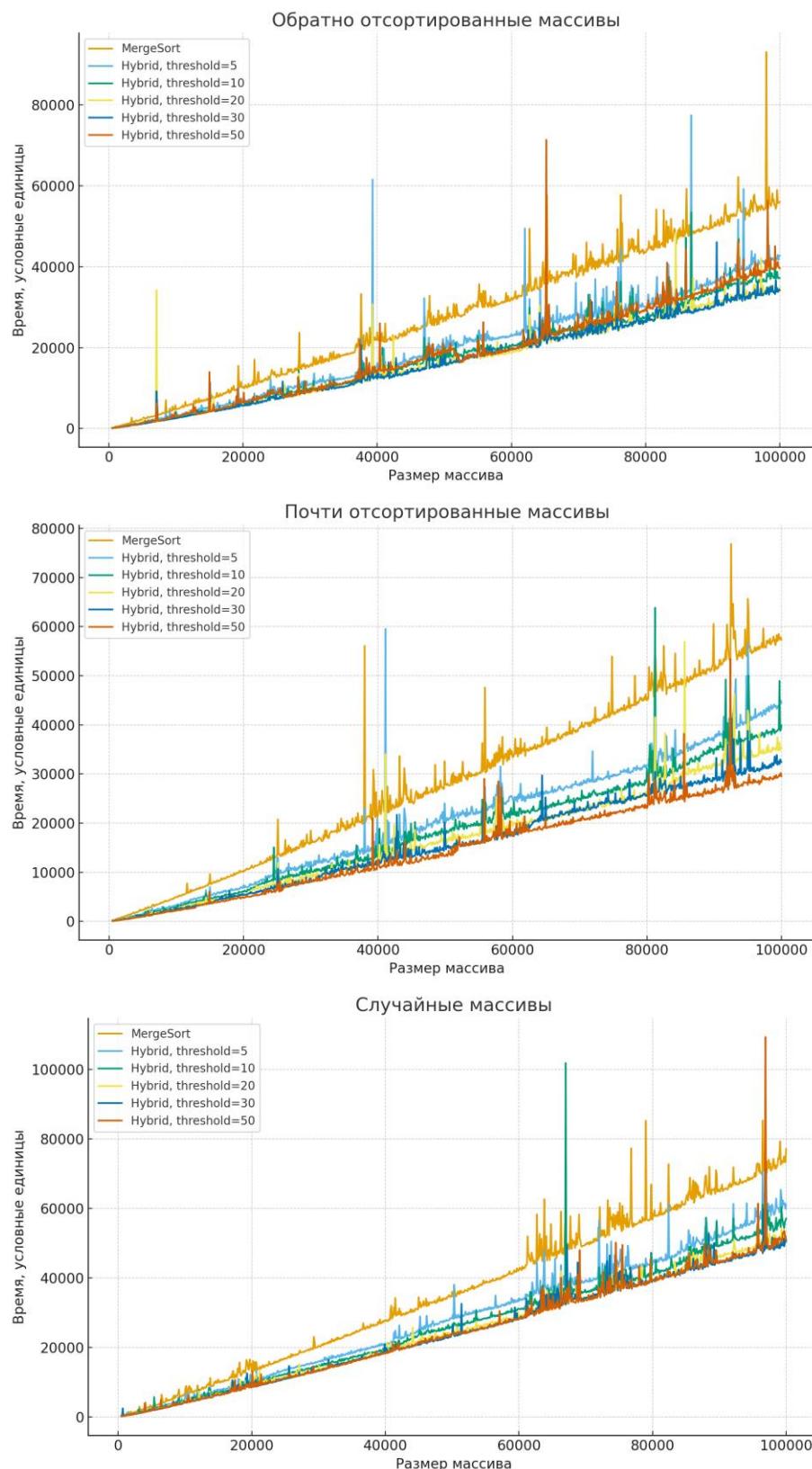


Графики:



Выводы

1. Общие наблюдения

- На всех типах данных чистый MergeSort самый медленный.
- Любая гибридная версия Merge+Insertion даёт ускорение примерно в 1.3–2 раза.
- Рост времени близок к $O(n \log n)$, без странных скачков.

2. Случайные массивы

- При $N \geq 50\,000$ гибрид быстрее MergeSort примерно:
 - $t=5: \sim 1.25\times, t=10: \sim 1.35\times, t=20: \sim 1.45\times, t=30-50: \sim 1.47\times$.
- Оптимальный диапазон порога: 20–30 элементов.

3. Обратно отсортированные массивы

- Для больших N :
 - $t=5: \sim 1.4\times, t=10: \sim 1.5\times, t=20-30: \sim 1.65-1.66\times, t=50: \sim 1.5\times$.
- Слишком большой порог (50) ухудшает результат: InsertionSort на длинных «плохих» кусках слишком дорог
- Разумный порог: 20–30 элементов.

4. Почти отсортированные массивы

- Здесь InsertionSort максимально выгоден: при $N \geq 50\,000$ гибрид быстрее MergeSort примерно:
 - $t=5: \sim 1.4\times, t=10: \sim 1.5\times, t=20: \sim 1.7\times, t=30: \sim 1.8\times, t=50: \sim 1.9\times$.
- Чем больше threshold, тем лучше: порог 50 даёт лучший результат.

5. Итог

- Гибридный Merge+Insertion практически во всех экспериментах заметно лучше чистого MergeSort.
- Если настраивать порог под конкретный тип:
 - случайный и обратно отсортированный: $t \approx 20\text{--}30$,
 - почти отсортированный: t как можно больше (в нашем эксперименте — 50).
- Если нужен один универсальный порог для всех случаев, разумный компромисс — $t \approx 30$:
 - почти оптимален на случайных,
 - оптимален на обратно отсортированных – на почти отсортированных всего немного хуже $t=50$, но всё равно почти в 1.8 раза быстрее стандартного MergeSort.

Ссылка на реализацию и данные

- ID посылки задачи A1i на Codeforces: [349015954](#)
- Публичный репозиторий с исходными данными и скриптами построения графиков: <https://github.com/LeskaProgrammer/algo>