# A2. Анализ MERGE+INSERTION SORT

ID: 293141530
Алгоритм лежит в A2i/main.cpp

Этап 1. Подготовка тестовых данных
Реализуйте класс `ArrayGenerator` для генерации тестовых массивов, заполненных целыми числами, со следующими характеристиками:

1. Массивы, которые заполнены случайными значениями в некотором диапазоне.
2. Массивы, которые отсортированы в обратном порядке по невозрастанию.
3. Массивы, которые <<почти>> отсортированы. Их можно получить, обменяв местами небольшое количество пар элементов в полностью отсортированном массиве.

В рамках задачи зафиксируем следующие параметры для подготовки тестовых данных:

- размеры массивов --- от 500 до 10000 с шагом 100 и
- диапазон случайных значений --- от 0 до 6000.

Для удобства подготовки тестовых данных сгенерируйте для каждого случая массив максимальной длины (10000), из которого выбирайте подмассив необходимого размера (500, 600, 700, ... элементов).

Класс `ArrayGenerator` лежит в `A2_additional/array_generator.h`
Этап 2. Эмпирический анализ стандартного алгоритма MERGE SORT
Проведите замеры времени работы стандартной реализации алгоритма MERGE SORT и представьте результаты в виде трех (групп) графиков для каждой категории тестовых данных. Замеры времени работы удобнее всего производить с помощью встроенной библиотеки `std::chrono`. Например:

```
auto start = std::chrono::high_resolution_clock::now();
mergeSort(A, l, r);
auto elapsed = std::chrono::high_resolution_clock::now() - start;
long long msec =
std::chrono::duration_cast<std::chrono::milliseconds>(elapsed).cou
nt();
```

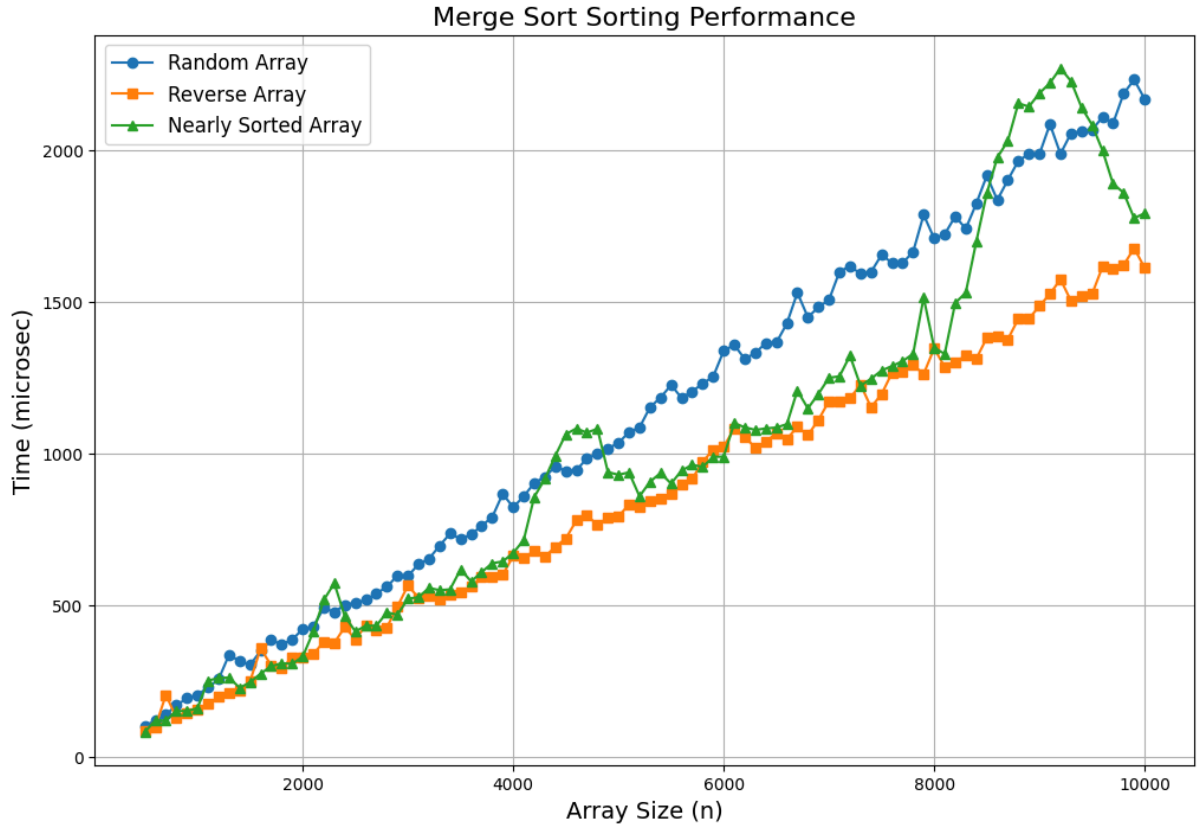
При выполнении эмпирического анализа обратите внимание на:

- минимизацию влияния работы других программ и сетевого подключения,

- необходимость многократных замеров времени работы и усреднения результатов --- количество замеров и способ усреднения остаётся на ваше усмотрение,
- выбор единиц измерения времени --- единицы измерения слишком большого масштаба (например, секунды) приведут к получению плохо интерпретируемых результатов.

Замеры я проводил в микросекундах(иначе слишком маленький разброс)
в A2_additional/sorting_alg.h лежат реализации сортировок
в A2_additional/sort_tester.h лежат функции замера времени
в A2_additional/main.cpp лижит запуск

в A2_additional/ms_results.txt лежат результаты замеров
в A2_additional/Set_3_A2.ipynb лежит код построения графиков
Ссылка на google collab с построением графиков: тык
Графики:




Зеленый график сильно улетел, но я случайно менял 100 элементов из 10000, это 1%, видимо они в конце встречаются, я перезапускал несколько раз, скачек остался.

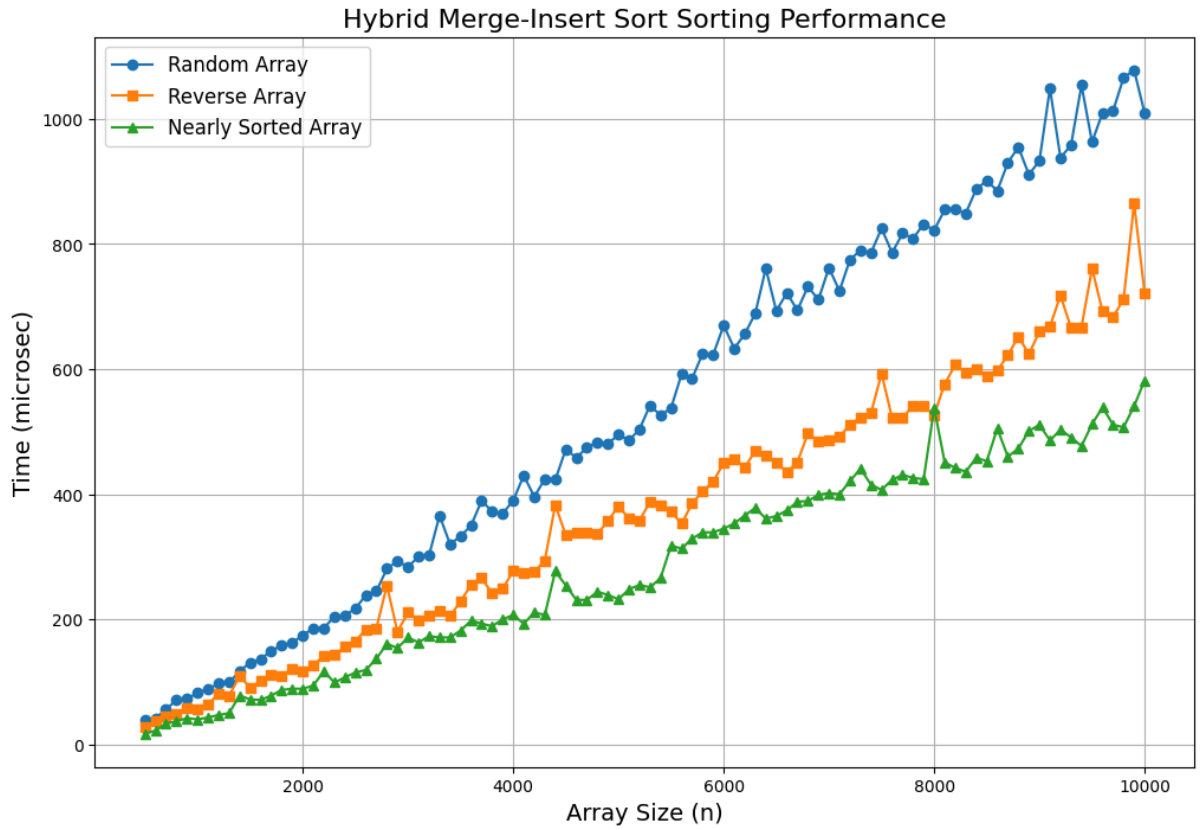
Этап 3. Эмпирический анализ гибридного алгоритма MERGE+INSERTION SORT

Проведите аналогичные предыдущему этапу замеры времени работы гибридной реализации алгоритма MERGE+INSERTION SORT и представьте результаты в виде трех (групп) графиков для каждой категории подготовленных тестовых данных.

Диапазон параметра переключения (`threshold`) на алгоритм INSERTION SORT остаётся полностью на ваше усмотрение. Например, можно рассмотреть переключение на INSERTION SORT для массивов, состоящих из 5, 10, 20, 30 и 50 элементов.

Функции эмпирического замера времени работы рассматриваемых алгоритмов сортировки реализуйте в отдельном классе `SortTester`.

в A2_additional/hms_results.txt лежат результаты замеров
в A2_additional/Set_3_A2.ipynb лежит код построения графиков(наверное уже увидели)
Графики:




threshold брал 20

Этап 4. Сравнительный анализ

Опишите полученные вами результаты и представьте *содержательные* выводы о сравнении временных затрат двух рассмотренных реализаций алгоритма сортировки слиянием. При выполнении сравнительного анализа, среди прочего, можно обратить внимание на поиск порогового значения параметра переключения в гибридном алгоритме MERGE+INSERTION SORT, начиная с которого он работает медленнее стандартной реализации MERGE SORT.

На основе проведенных замеров и построенных графиков для стандартного алгоритма Merge Sort и гибридного алгоритма Merge+Insertion Sort сделаны следующие наблюдения и выводы:

1. Анализ временных затрат стандартного Merge Sort

• Общая характеристика:
• Стандартный Merge Sort демонстрирует стабильные временные затраты для всех типов массивов: случайных, обратно отсортированных и почти отсортированных.
• С ростом размера массива временные затраты увеличиваются nlogn относительно , что соответствует теоретической сложности алгоритма.
• Различия для типов массивов:
• Случайные и обратно отсортированные массивы обрабатываются примерно за одинаковое время.
• Для почти отсортированных массивов время выполнения немного меньше благодаря меньшему количеству необходимых перестановок (кроме 8000-9000 не понятно из-за чего).

2. Анализ временных затрат гибридного Merge+Insertion Sort

• Общая характеристика:
• Гибридный Merge+Insertion Sort существенно превосходит стандартный Merge Sort на массивах малого и среднего размера, достигая вдвое большей скорости на массивах размером .
• При увеличении размера массива разница между алгоритмами постепенно уменьшается, а на очень больших массивах стандартный Merge Sort начинает работать быстрее из-за лучшей масштабируемости.
• Различия для типов массивов:
• На случайных и обратно отсортированных массивах гибридный подход эффективен вплоть до массивов среднего размера.
• На почти отсортированных массивах гибридный алгоритм показывает лучшие результаты даже на более крупных массивах, благодаря высокой эффективности сортировки вставками.

3. Сравнение алгоритмов и пороговое значение

• Пороговое значение размера массива:
• На массивах размером 10000  гибридный алгоритм работает вдвое быстрее стандартного Merge Sort. Это связано с тем, что сортировка вставками снижает накладные расходы на рекурсию и ускоряет обработку небольших подмассивов.
• Для массивов  преимущество гибридного подхода исчезает, и стандартный Merge Sort начинает показывать лучшие результаты за счёт лучшей производительности на больших объёмах данных.

• Пример порогового значения:
• Для текущей реализации оптимальным пороговым значением, где гибридный алгоритм переключается на сортировку вставками, является область – (5-400. После этого стандартный Merge Sort начинает показывать сопоставимое или лучшее время. Для 400 на random почти одинаковые, на reverse обычный merge_sort быстрее в 2 раза, на near hybrid намного быстрее в ~4 раза.

4. Выводы

1. Эффективность гибридного Merge+Insertion Sort:
• На массивах размером до 10000 гибридный алгоритм стабильно превосходит стандартный Merge Sort, показывая вдвое меньшие временные затраты.
• Особенно эффективен на почти отсортированных массивах, где сортировка вставками работает с минимальными накладными расходами.
2. Эффективность стандартного Merge Sort:
• На больших массивах (> 10000) стандартный Merge Sort становится более предпочтительным из-за меньшего числа рекурсивных вызовов и лучшей масштабируемости.
3. Практическое применение:
• Гибридный подход лучше всего подходит для обработки массивов малого и среднего размера, а также массивов с частично отсортированной структурой.
• Для больших массивов стандартный Merge Sort остаётся более универсальным решением.