Задача A2. Анализ MERGE+INSERTION SORT

Известно, что алгоритм `MERGE SORT` является одним из оптимальных алгоритмов сортировки на основе сравнений элементов и имеет сложность $O(n \cdot \log n)$, а алгоритм `INSERTION SORT` имеет сложность $O(n^2)$. Однако на массивах сравнительно небольшого размера `INSERTION SORT` сортирует быстрее `MERGE SORT` ввиду лучшей оценки скрытой константы, а также снижения накладных расходов, которые связаны с рекурсией.

В рамках этой задачи требуется провести экспериментальное исследование двух реализаций алгоритма `MERGE SORT`:

- стандартной рекурсивной (с выделением дополнительной памяти) и
- гибридной, которая на массивах малого размера переключается на `INSERTION SORT`.

Язык программирования, который должен использоваться при реализации алгоритмов и проведении замеров времени их работы, — `C++`. Ограничений на используемые средства обработки и визуализации эмпирических данных нет.

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).

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).count();
```

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.

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`.

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`.

Помимо графиков и пояснений, приложите:

1. Реализацию класса `ArrayGenerator` и `SortTester`.
2. ID своей посылки по задаче `A2i` в системе `CodeForces` с реализацией `MERGE+INSERTION SORT`.
3. Ссылку на публичный репозиторий с исходными данными, полученными в результате эмпирических замеров времени работы алгоритмов.

Система оценки

1. 5 баллов Реализация гибридного алгоритма сортировки `MERGE+INSERTION SORT`.
2. 6 баллов Реализация внутренней инфраструктуры для экспериментального анализа — классы `ArrayGenerator` и `SortTester`.
3. 7 баллов Представление эмпирических замеров времени работы рассматриваемых алгоритмов.
4. 7 баллов Сравнительный анализ полученных эмпирических данных.

Обратите внимание, что загрузка реализации алгоритма `MERGE+INSERTION SORT` в задачу `A2i` является *необходимым* условием для получения оценки по другим критериям.