A2. Анализ MERGE+INSERTION SORT

Головина Арина БПИ244
ID посылки: [349264731]
Ссылка на репозиторий: https://github.com/AMGolovina/Algorithm/tree/main/SET3/A2

Этап 1. Подготовка тестовых данных:

Реализация класса ArrayGenerator:

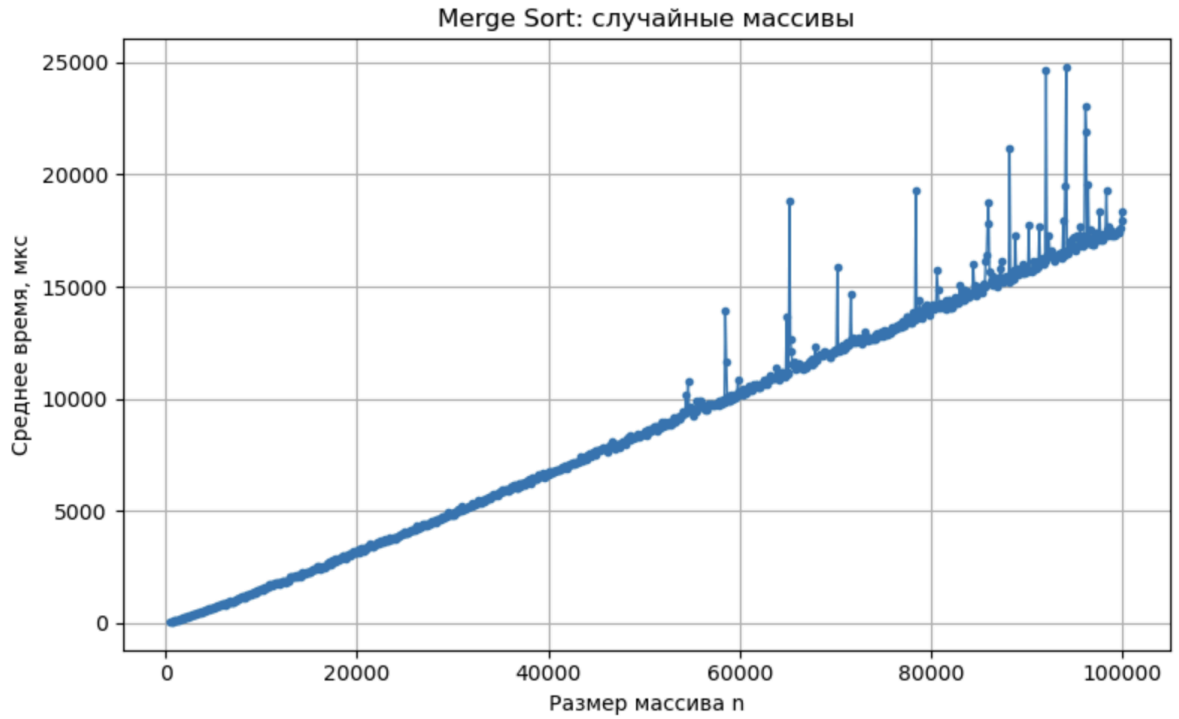
```
class ArrayGenerator {
    std::vector<int> randomBaseArray;
    std::vector<int> reversedBaseArray;
    std::vector<int> nearlyBaseArray;

    int maxLength;
    int minRange;
    int maxRange;

    void generateRandomBaseArray() {
        std::mt19937 gen(static_cast<unsigned>(std::time(nullptr)));
        std::uniform_int_distribution<int> dist(minRange, maxRange);

        randomBaseArray.clear();
        randomBaseArray.reserve(maxLength);
        for (int i = 0; i < maxLength; ++i) {
            randomBaseArray.push_back(dist(gen));
        }
    }

    void generateReversedBaseArray() {
        std::mt19937 gen(static_cast<unsigned>(std::time(nullptr)) +
1);
        std::uniform_int_distribution<int> dist(minRange, maxRange);

        reversedBaseArray.clear();
        reversedBaseArray.reserve(maxLength);
        for (int i = 0; i < maxLength; ++i) {
            reversedBaseArray.push_back(dist(gen));
        }

        std::sort(reversedBaseArray.rbegin(),
reversedBaseArray.rend());
    }

    void generateAlmostBaseArray() {
        std::mt19937 gen(static_cast<unsigned>(std::time(nullptr)) +
2);
        std::uniform_int_distribution<int> dist(minRange, maxRange);
```

```
        nearlyBaseArray.clear();
        nearlyBaseArray.reserve(maxLength);
        for (int i = 0; i < maxLength; ++i) {
            nearlyBaseArray.push_back(dist(gen));
        }
    }

public:
    ArrayGenerator()
        : maxLength(100000),
          minRange(0),
          maxRange(6000)
    {
        generateRandomBaseArray();
        generateReversedBaseArray();
        generateAlmostBaseArray();
    }

    std::vector<int> getRandomArray(int size) {
        if (size > maxLength) {
            size = maxLength;
        }
        return std::vector<int>(randomBaseArray.begin(),
                                randomBaseArray.begin() + size);
    }

    std::vector<int> getReversedArray(int size) {
        if (size > maxLength) {
            size = maxLength;
        }
        return std::vector<int>(reversedBaseArray.begin(),
                                reversedBaseArray.begin() + size);
    }

    std::vector<int> getAlmostSortedArray(int size, int swaps) {
        if (size > maxLength) {
            size = maxLength;
        }

        std::vector<int> almostSortedArray(almostBaseArray.begin(),
                                           almostBaseArray.begin() +
size);

        std::sort(almostSortedArray.begin(),
almostSortedArray.end());

        std::mt19937 gen(static_cast<unsigned>(std::time(nullptr)));
        std::uniform_int_distribution<int> dist(0, size - 1);
```

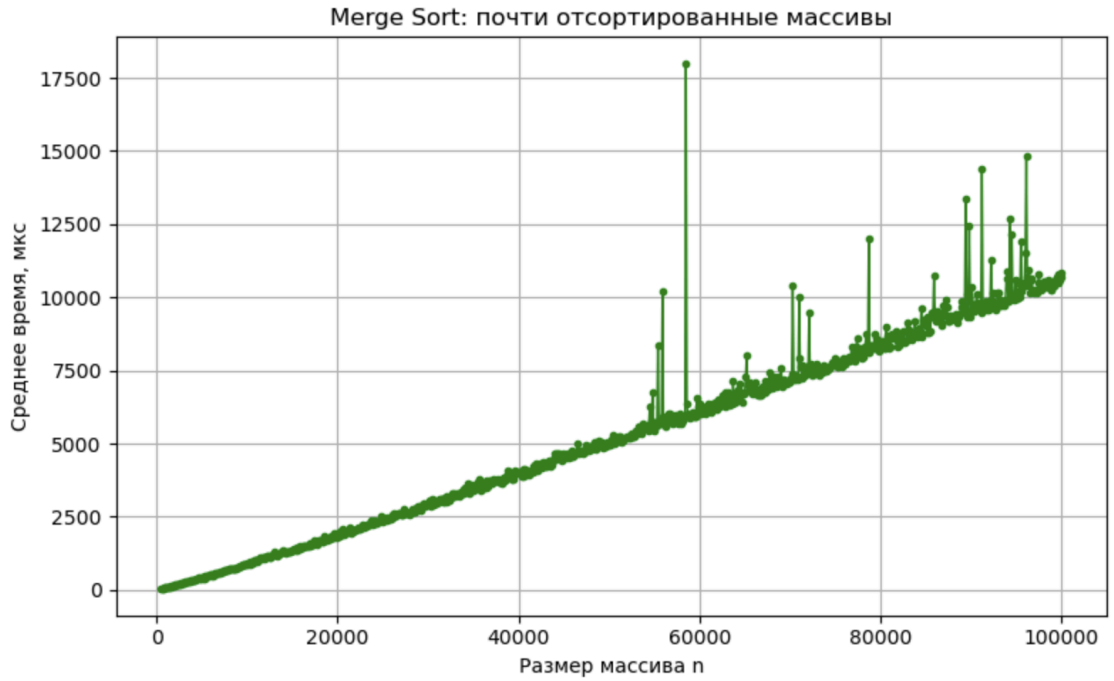
```
        for (int i = 0; i < swaps; ++i) {
            int i1 = dist(gen);
            int i2 = dist(gen);
            std::swap(almostSortedArray[i1],
almostSortedArray[i2]);
        }

        return almostSortedArray;
    }
};
```

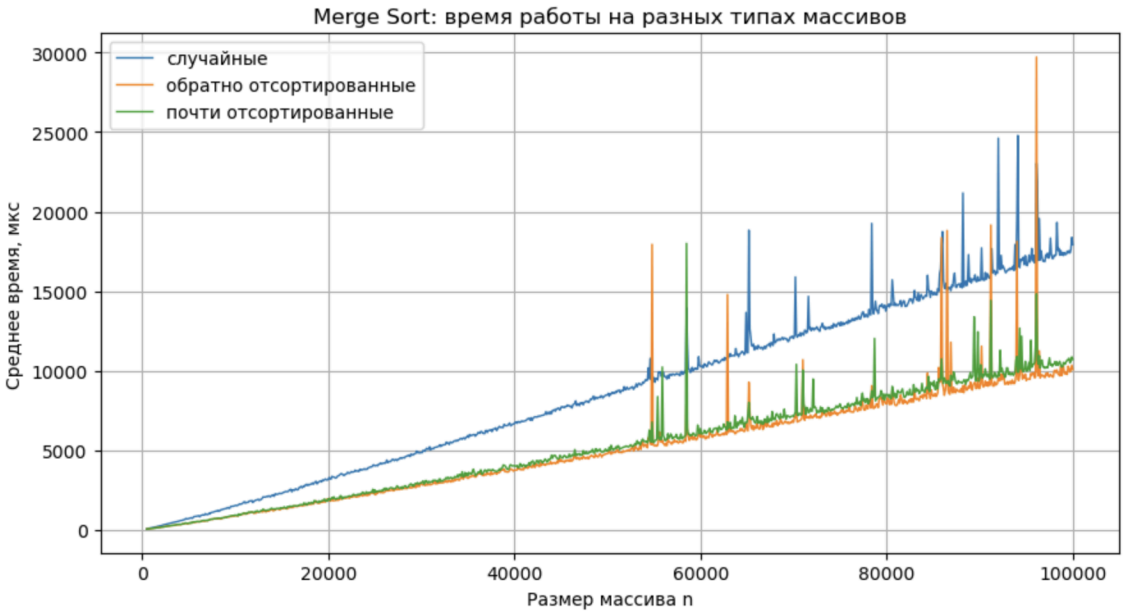
Этап 2. Эмпирический анализ стандартного алгоритма Merge Sort:

MergeSortDATA.txt - файл с результатами работы сортировки MergeSort.



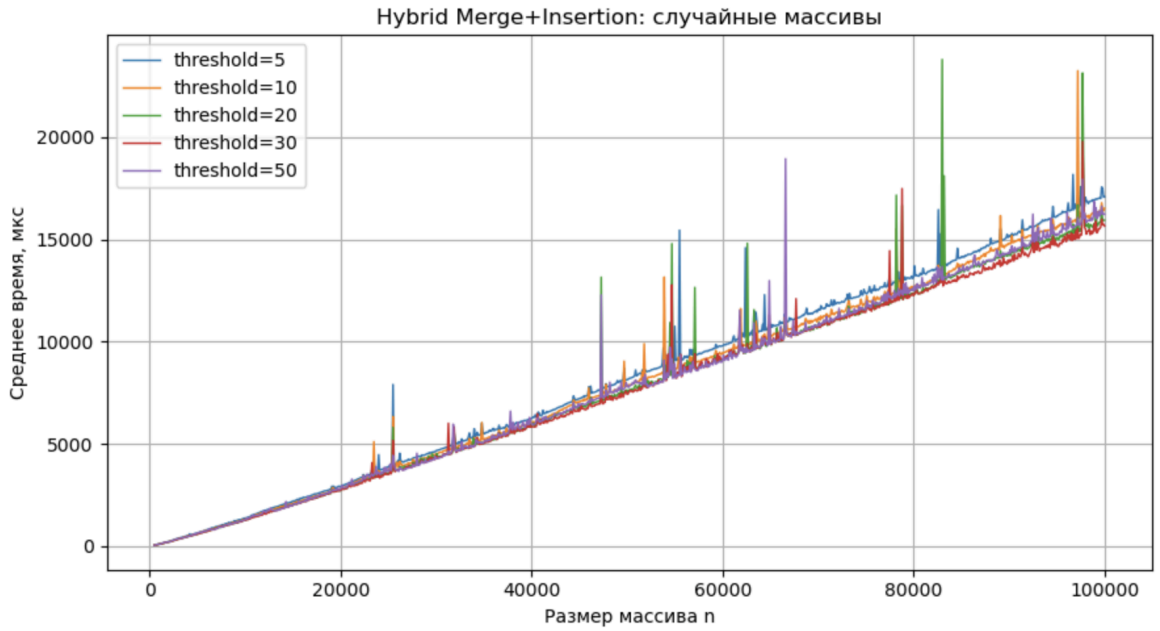

Merge Sort: обратно отсортированные массивы
Среднее время, мкс
Размер массива n
0
5000
10000
15000
20000
25000
30000
0
20000
40000
60000
80000
100000

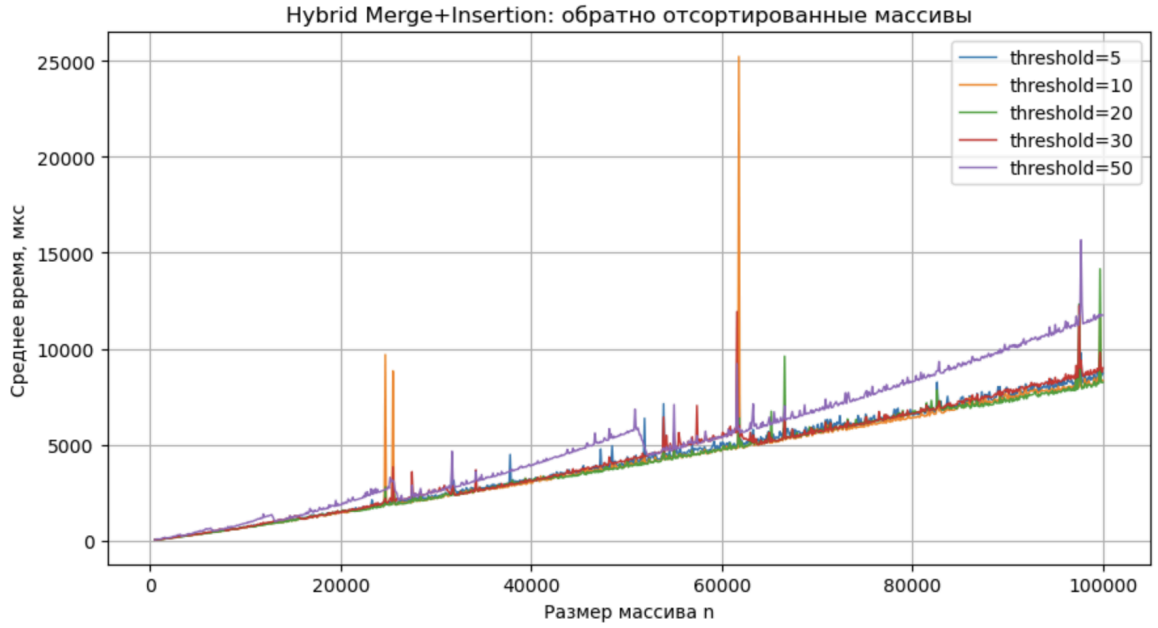

Merge Sort: почти отсортированные массивы
Среднее время, мкс
Размер массива n
0
2500
5000
7500
10000
12500
15000
17500
0
20000
40000
60000
80000
100000

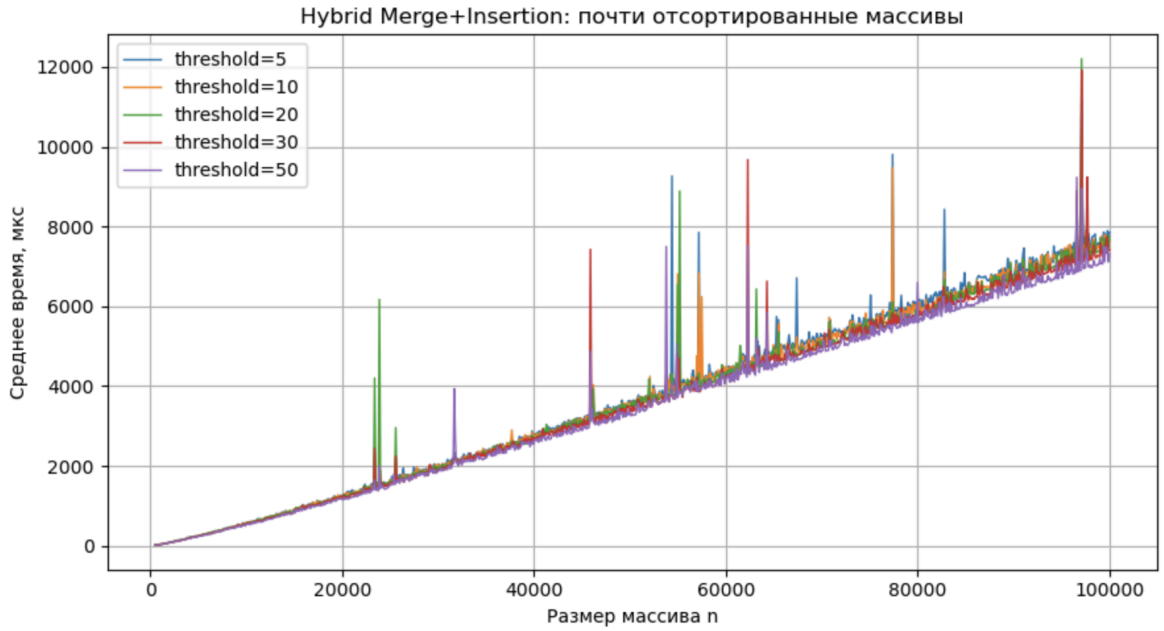


Этап 3. Эмпирический анализ гибридного алгоритма MERGE+INSERTION SORT:

HubridSortDATA.txt - файл с результатами работы гибридной сортировки.


Hybrid Merge+Insertion: случайные массивы
threshold=5
threshold=10
threshold=20
threshold=30
threshold=50
Среднее время, мкс
0
5000
10000
15000
20000
0
20000
40000
60000
80000
100000
Размер массива n


Hybrid Merge+Insertion: обратно отсортированные массивы
threshold=5
threshold=10
threshold=20
threshold=30
threshold=50
Среднее время, мкс
0
5000
10000
15000
20000
25000
0
20000
40000
60000
80000
100000
Размер массива n



Этап 4. Сравнительный анализ:

На массивах среднего и большого размера гибридный Merge + Insertion Sort в большинстве экспериментов оказался быстрее стандартного Merge Sort, так как на маленьких подмассивах вставки работают эффективнее. Эффективность гибридного алгоритма зависит от порога threshold: слишком маленький не даёт выигрыша, а слишком большой замедляет работу из-за квадратичного поведения Insertion Sort. По результатам экспериментов наилучший диапазон threshold лежит примерно в пределах 10–20 элементов: здесь гибрид стабильно быстрее классического Merge Sort, тогда как при 30–50 на больших массивах он начинает проигрывать. На почти отсортированных массивах гибрид даёт максимальный выигрыш, на случайных — умеренный, а на обратно отсортированных большие значения threshold особенно вредны. В целом гибридный Merge + Insertion Sort при разумном выборе threshold не хуже Merge Sort, но нужно использовать небольшой фиксированный порог порядка 10–20 элементов.