Id посылки: 349020717

Графики внизу

ссылка на гит: https://github.com/jefrryss/Algorithms.git - все реализации классов SortTester, ArrayGenerator и сортировок на гите

1 Сравнение гибридного MERGE+INSERTION при порогах 15 и 50

Время работы алгоритмов измерялось в **микросекундах**. Для каждого размера массива алгоритм прогонялся несколько раз, а затем считалось **среднее время**.

Я проверил два варианта гибрида:

```
THRESHOLD = 15
THRESHOLD = 50
```

- **`results.txt`** — гибрид с порогом `THRESHOLD = 50`;
- **`results1.txt`** — гибрид с порогом `THRESHOLD = 15`.
-

Также проверял три типа массивов:

- случайные (`random`)
- полностью обратные (`reversed`)
- почти отсортированные (`almost`)

2. Сравнение по типам данных

Случайные массивы (`random`)

- TH=15 → быстрее Merge Sort.
- TH=50 → быстрее Merge Sort.

Обратные массивы (`reversed`)

- TH=15 → работает быстрее Merge.
- TH=50 → при данных > 70 000 работает медлнее Merge.

Почти отсортированные (`almost`)

- TH=15 → работает лучше Merge, но медленее Th=50.

- TH=50 → лучший результат, гибрид работает быстро.

3. Итог

- **Порог 15 — лучший выбор.**
  Даёт ускорение и почти никогда не замедляется.
- **Порог 50 — слишком большой.**
  Из-за этого гибрид иногда работает **даже хуже**, чем обычный Merge Sort.

Финальный вывод:

> **Гибридный Merge+Insertion Sort работает лучше всех при THRESHOLD ≈ 10–20.**
> **Значение 15 — оптимальное.**
> **Значение 50 — замедляет работу на больших массивах.**

Класс `ArrayGenerator`

Поля

- `int maxSize_`
  Максимальный размер базовых массивов (обычно 100000).
- `vector<long long> randomBase_`
  Базовый массив со случайными значениями (0...6000).
- `vector<long long> reversedBase_`
  Тот же набор значений, но отсортированный по убыванию.
- `vector<long long> almostSortedBase_`
  Почти отсортированный массив (отсортированный по возрастанию + несколько случайных перестановок).

Методы

- `ArrayGenerator(int maxSize = 100000)`
  Конструктор. Генерирует все три базовых массива сразу.
- `int maxSize() const`
  Возвращает максимальный размер базовых массивов.
- `vector<long long> getRandomArray(int n) const`
  Возвращает первые `n` элементов из `randomBase_`.
- `vector<long long> getReversedArray(int n) const`
  Возвращает первые `n` элементов из `reversedBase_`.

- `vector<long long> getAlmostSortedArray(int n) const`
  Возвращает первые `n` элементов из `almostSortedBase_`.

Класс `SortTester`

Поля

- `ArrayGenerator& gen_`
  Ссылка на генератор массивов. Через него берутся данные для тестов.
- `int repeats_`
  Количество повторов для каждого размера массива (чтобы усреднить время).

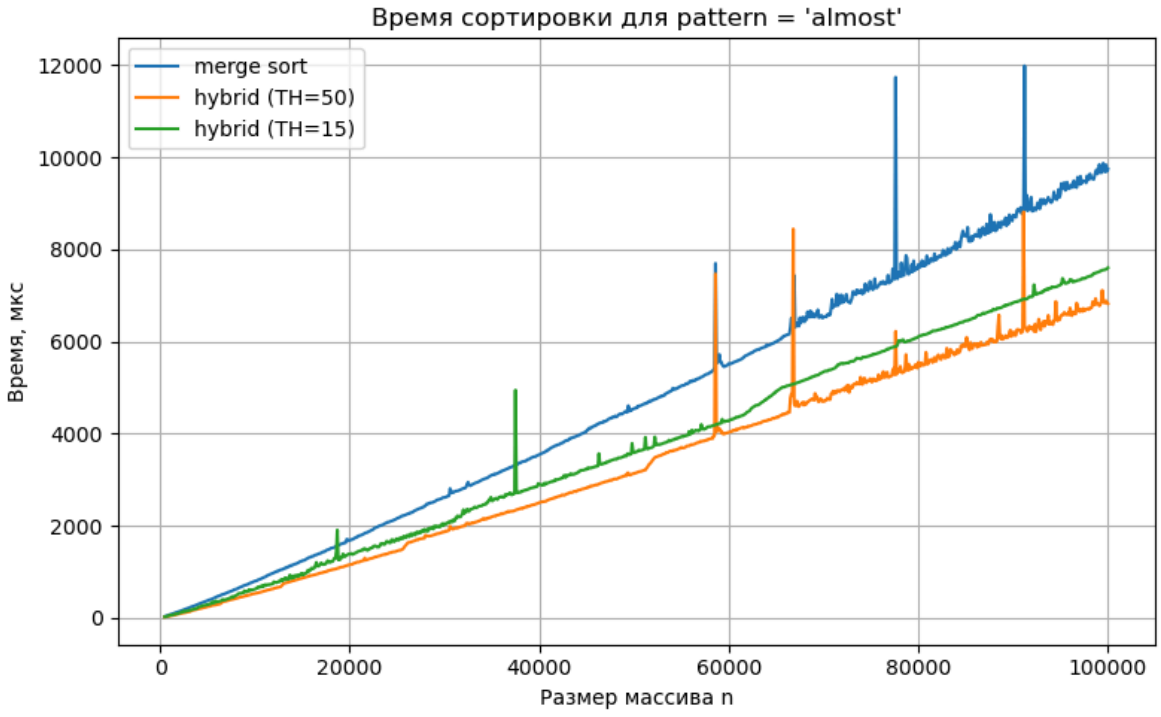
Методы

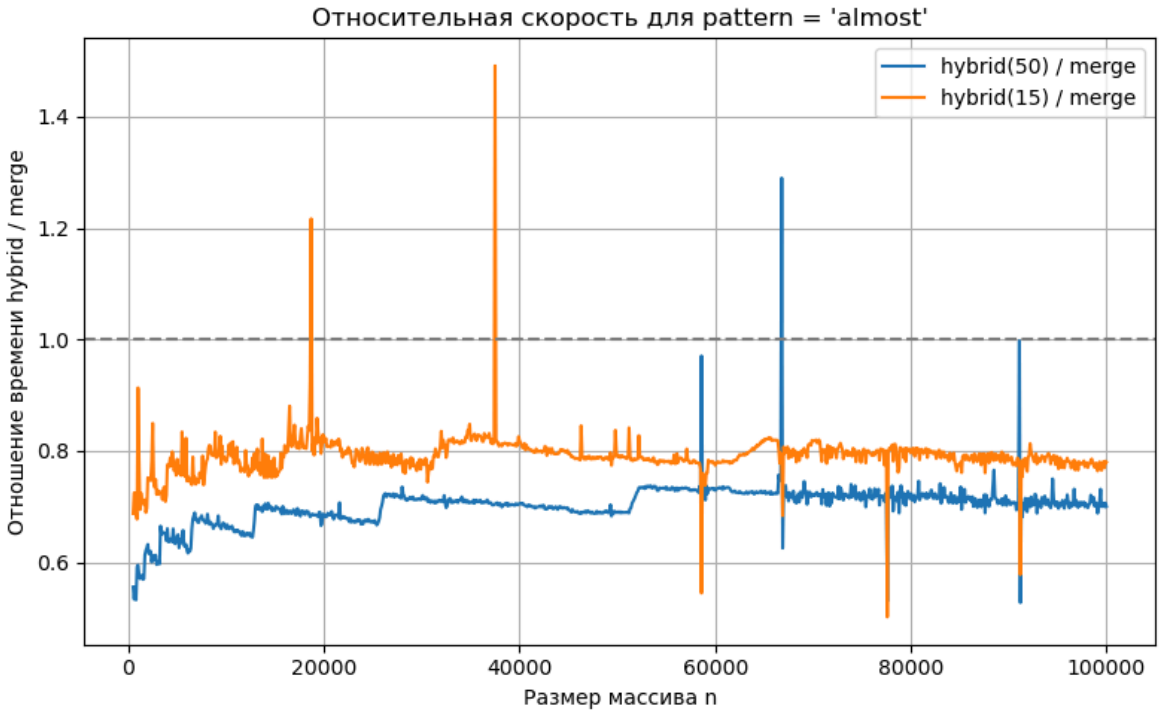
- `SortTester(ArrayGenerator& gen, int repeats)`
  Конструктор. Сохраняет ссылку на генератор и число повторов.
- `void run()`
  Основной метод.
  Циклы по размерам `n` и типам массивов (`random`, `reversed`, `almost`), запуск тестов и запись результатов в `results.txt`.
- `void test_one_size(ofstream& out, int n, const string& pattern)` *(приватный)*
  Для фиксированных `n` и типа массива:
  - получает массив нужного вида;
  - измеряет время `merge_sort`;
  - измеряет время `merge_sort_hybrid`;
  - пишет две строки в файл: для `merge` и для `hybrid`.

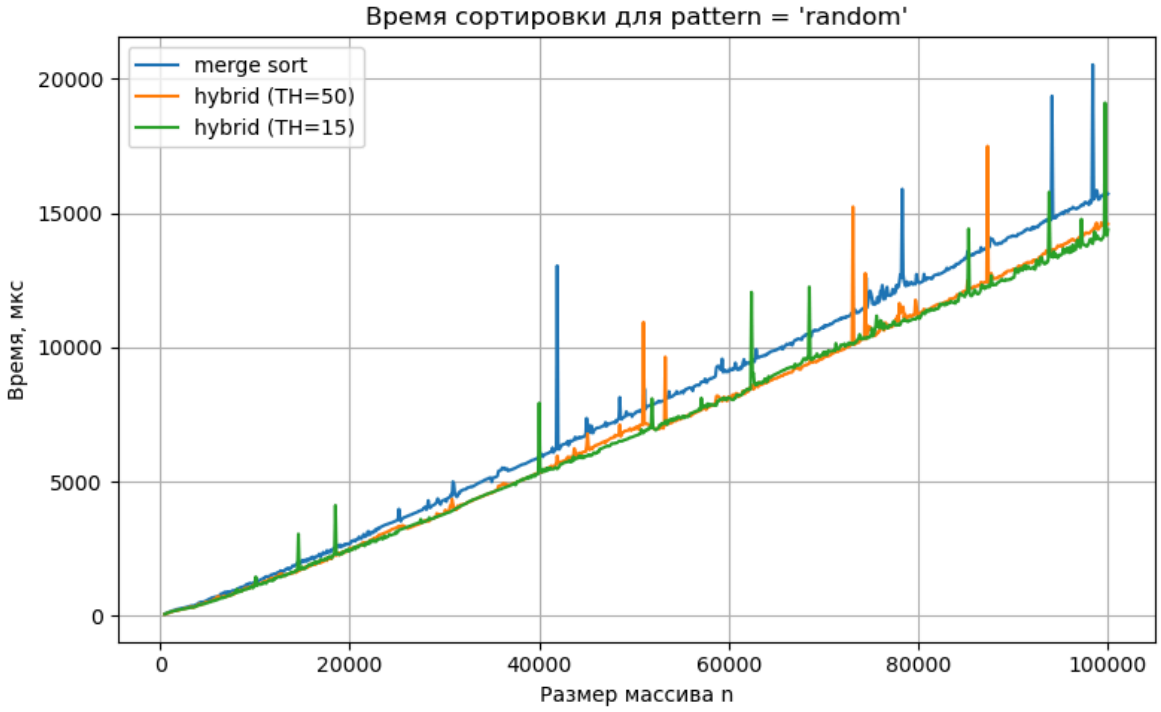
In [12]:
```
%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt

df50 = pd.read_csv("results.txt", sep=r"\s+")
df15 = pd.read_csv("results1.txt", sep=r"\s+")



patterns = sorted(set(df50["pattern"].unique()) & set(df15["pattern"].uni
print("Найденные типы массивов:", patterns)

for pattern in patterns:
    sub50 = df50[df50["pattern"] == pattern]
    sub15 = df15[df15["pattern"] == pattern]
```

```
    piv50 = sub50.pivot(index="n", columns="algo", values="avg_time_micro
    piv15 = sub15.pivot(index="n", columns="algo", values="avg_time_micro

    for name, piv in [("TH=50", piv50), ("TH=15", piv15)]:
        if "merge" not in piv.columns or "hybrid" not in piv.columns:
            print(f"⚠ Для pattern={pattern}, {name} нет merge или hybrid:

    merge_time = piv50["merge"]
    hybrid50 = piv50["hybrid"]
    hybrid15 = piv15["hybrid"]

    plt.figure(figsize=(8, 5))
    plt.plot(merge_time.index, merge_time, label="merge sort")
    plt.plot(hybrid50.index, hybrid50, label="hybrid (TH=50)")
    plt.plot(hybrid15.index, hybrid15, label="hybrid (TH=15)")
    plt.xlabel("Размер массива n")
    plt.ylabel("Время, мкс")
    plt.title(f"Время сортировки для pattern = '{pattern}'")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    ratio50 = hybrid50 / merge_time
    ratio15 = hybrid15 / merge_time

    plt.figure(figsize=(8, 5))
    plt.plot(ratio50.index, ratio50, label="hybrid(50) / merge")
    plt.plot(ratio15.index, ratio15, label="hybrid(15) / merge")
    plt.axhline(1.0, linestyle="--", color="gray")
    plt.xlabel("Размер массива n")
    plt.ylabel("Отношение времени hybrid / merge")
    plt.title(f"Относительная скорость для pattern = '{pattern}'")
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

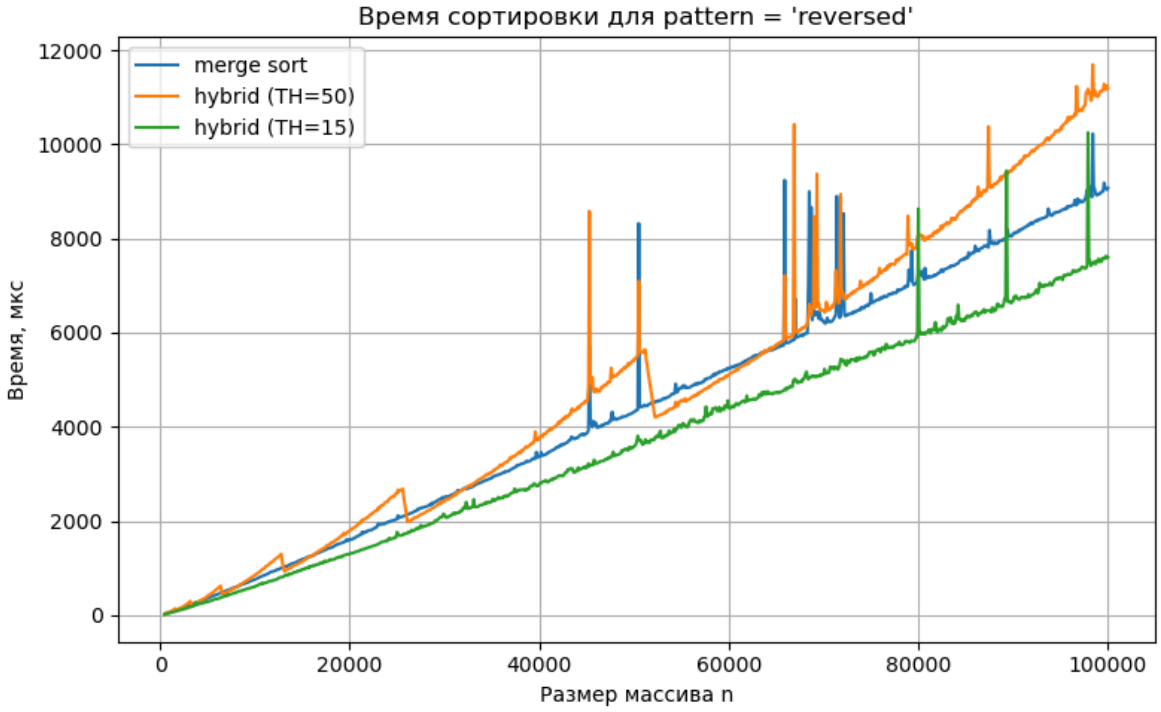
```
Найденные типы массивов: ['almost', 'random', 'reversed']
```

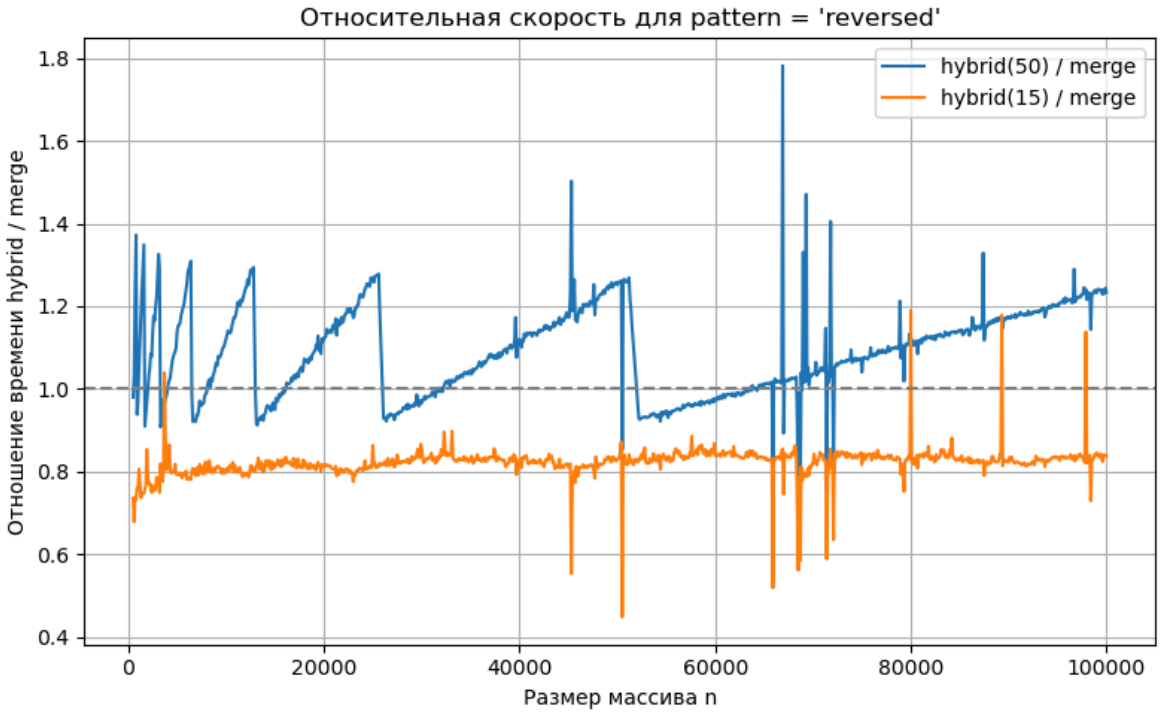



Относительная скорость для pattern = 'almost'
hybrid(50) / merge
hybrid(15) / merge
Отношение времени hybrid / merge
1.4
1.2
1.0
0.8
0.6
0
20000
40000
60000
80000
100000
Размер массива n


Время сортировки для pattern = 'random'
merge sort
hybrid (TH=50)
hybrid (TH=15)
Время, мкс
20000
15000
10000
5000
0
0
20000
40000
60000
80000
100000
Размер массива n


Относительная скорость для pattern = 'random'
hybrid(50) / merge
hybrid(15) / merge
Отношение времени hybrid / merge
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
20000
40000
60000
80000
100000
Размер массива n


Время сортировки для pattern = 'reversed'
merge sort
hybrid (TH=50)
hybrid (TH=15)
Время, мкс
12000
10000
8000
6000
4000
2000
0
0
20000
40000
60000
80000
100000
Размер массива n


Относительная скорость для pattern = 'reversed'
hybrid(50) / merge
hybrid(15) / merge
Отношение времени hybrid / merge
Размер массива n
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
20000
40000
60000
80000
100000