

Исходный код mergeSort+InsertionSort:

```
1  #include <iostream>
2  #include <vector>
3
4  void insertionSort(std::vector<int>& array, int left, int right) {
5      for (int i = left + 1; i <= right; ++i) {
6          int key = array[i];
7          int j = i - 1;
8          while (j >= left && array[j] > key) {
9              array[j + 1] = array[j];
10             --j;
11         }
12         array[j + 1] = key;
13     }
14 }
15
16 void merge(std::vector<int>& array, int left, int middle, int right) {
17     int n1 = middle - left + 1;
18     int n2 = right - middle;
19
20     std::vector<int> left_array(n1);
21     std::vector<int> right_array(n2);
22
23     for (size_t i = 0; i < n1; ++i) {
24         left_array[i] = array[left + i];
25     }
26
27     for (size_t i = 0; i < n2; ++i) {
28         right_array[i] = array[middle + 1 + i];
29     }
30
31     int i = 0;
32     int j = 0;
33     int k = left;
34
35     while (i < n1 && j < n2) {
36         while (i < n1 && j < n2) {
37             if (left_array[i] <= right_array[j]) {
38                 array[k++] = left_array[i++];
39             } else {
40                 array[k++] = right_array[j++];
41             }
42         }
43         while (i < n1) {
44             array[k++] = left_array[i++];
45         }
46         while (j < n2) {
47             array[k++] = right_array[j++];
48         }
49     }
50 }
51
52 void hybridSort(std::vector<int>& array, int left, int right) {
53     const int threshold = 15;
54
55     if (right - left + 1 <= threshold) {
56         insertionSort(array, left, right);
57     } else {
58         int middle = left + (right - left) / 2;
59         hybridSort(array, left, middle);
60         hybridSort(array, middle + 1, right);
61         merge(array, left, middle, right);
62     }
63 }
64
65 int main() {
66     int n;
67     std::cin >> n;
```

main

```

63     }
64
65     int main() {
66         int n;
67         std::cin >> n;
68
69         if (n == 0) {
70             return 0;
71         }
72
73         std::vector<int> array(n);
74         for (size_t i = 0; i < n; ++i) {
75             std::cin >> array[i];
76         }
77
78         hybridSort(array, 0, n - 1);
79
80         for (size_t i = 0; i < n; ++i) {
81             std::cout << array[i] << " ";
82         }
83
84         std::cout << "\n";
85         return 0;
86     }
87

```

Этап 1: Реализация класса ArrayGenerator

```

6
7     struct TestResult {
8         int size;
9         std::string type;
10        double time;
11
12        bool operator<(const TestResult& other) const;
13    };
14
15    class ArrayGenerator {
16    public:
17        static std::vector<int> generateRandomArray(int size, int range);
18        static std::vector<int> generateReversedArray(int size);
19        static std::vector<int> generateNearlySortedArray(int size, int swaps);
20    };
21

```

```

8 → bool TestResult::operator<(const TestResult& other) const {
9     if (size == other.size) {
10         return type < other.type;
11     }
12     return size < other.size;
13 }
14
15 → std::vector<int> ArrayGenerator::generateRandomArray(int size, int range) {
16     std::vector<int> array(size);
17     std::mt19937 gen(std::random_device{}());
18     std::uniform_int_distribution<> dist(0, range);
19     std::generate(array.begin(), array.end(), [&]() -> int { return dist(gen); });
20     return array;
21 }
22
23 → std::vector<int> ArrayGenerator::generateReversedArray(int size) {
24     std::vector<int> array(size);
25     std::iota(array.rbegin(), array.rend(), 0);
26     return array;
27 }
28
29 → std::vector<int> ArrayGenerator::generateNearlySortedArray(int size, int swaps) {
30     std::vector<int> array(size);
31     std::iota(array.begin(), array.end(), 0);
32     std::mt19937 gen(std::random_device{}());
33     for (int i = 0; i < swaps; ++i) {
34         int idx1 = gen() % size;
35         int idx2 = gen() % size;
36         std::swap(array[idx1], array[idx2]);
37     }
38     return array;
39 }
40

```

Для удобства была сделана структура **TestResult**, которая хранит размер массива, тип измеряемого массива (рандомный, отсортирован в обратном порядке и почти отсортирован).

Этап 2: Эмпирический анализ стандартного алгоритма MERGE SORT

Реализация анализа алгоритма:

```

22 class SortTester {
23 public:
24     static double measureMergeSort(std::vector<int> array);
25     static void mergeSort(std::vector<int>& array, int left, int right);
26 private:
27     static void merge(std::vector<int>& array, int left, int middle, int right) {
28         int n1 = middle - left + 1;
29         int n2 = right - middle;
30
31         std::vector<int> left_array(n1);
32         std::vector<int> right_array(n2);
33
34         for (size_t i = 0; i < n1; ++i) {
35             left_array[i] = array[left + i];
36         }
37
38         for (size_t i = 0; i < n2; ++i) {
39             right_array[i] = array[middle + 1 + i];
40         }
41
42         int i = 0;
43         int j = 0;
44         int k = left;
45
46         while (i < n1 && j < n2) {
47             if (left_array[i] <= right_array[j]) {
48                 array[k++] = left_array[i++];
49             } else {
50                 array[k++] = right_array[j++];
51             }
52         }
53
54         while (i < n1) {
55             array[k++] = left_array[i++];
56         }
57
58         while (j < n2) {
59             array[k++] = right_array[j++];
60         }
61     }
62 };

```

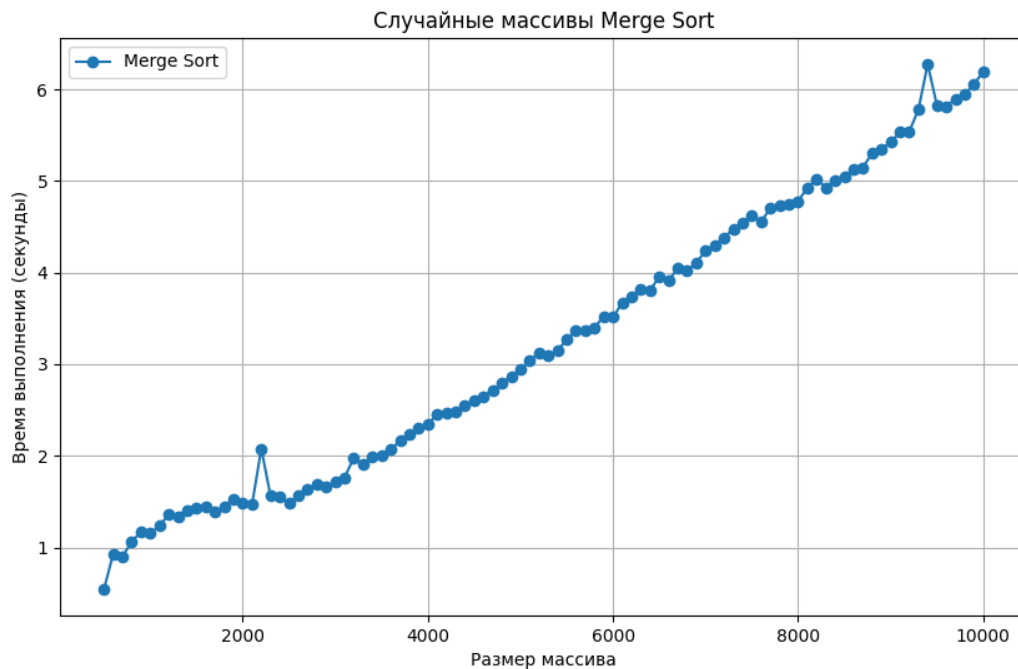
```

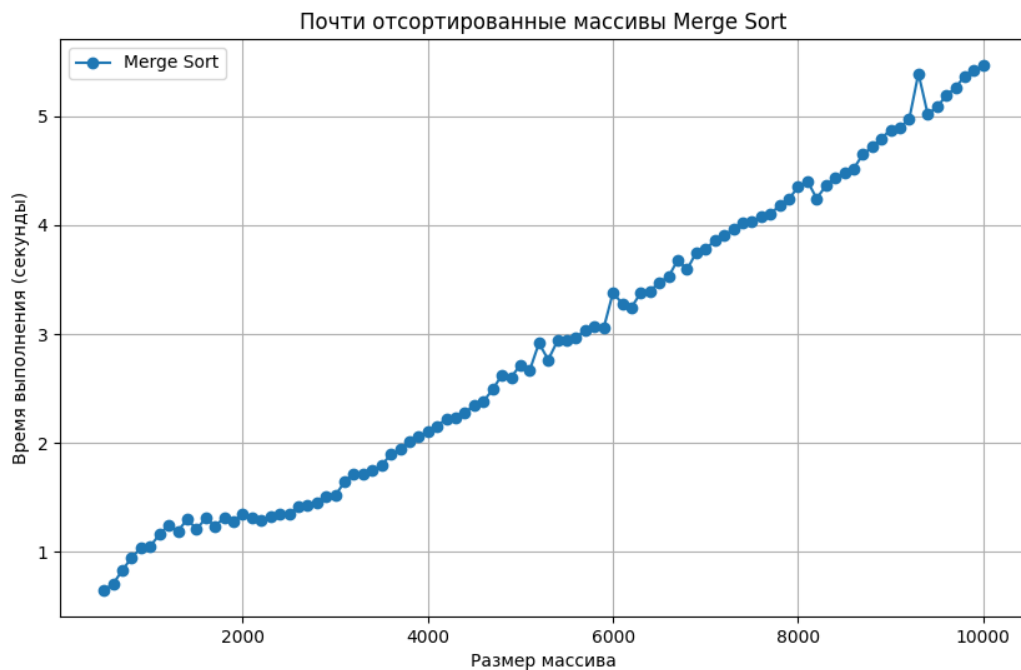
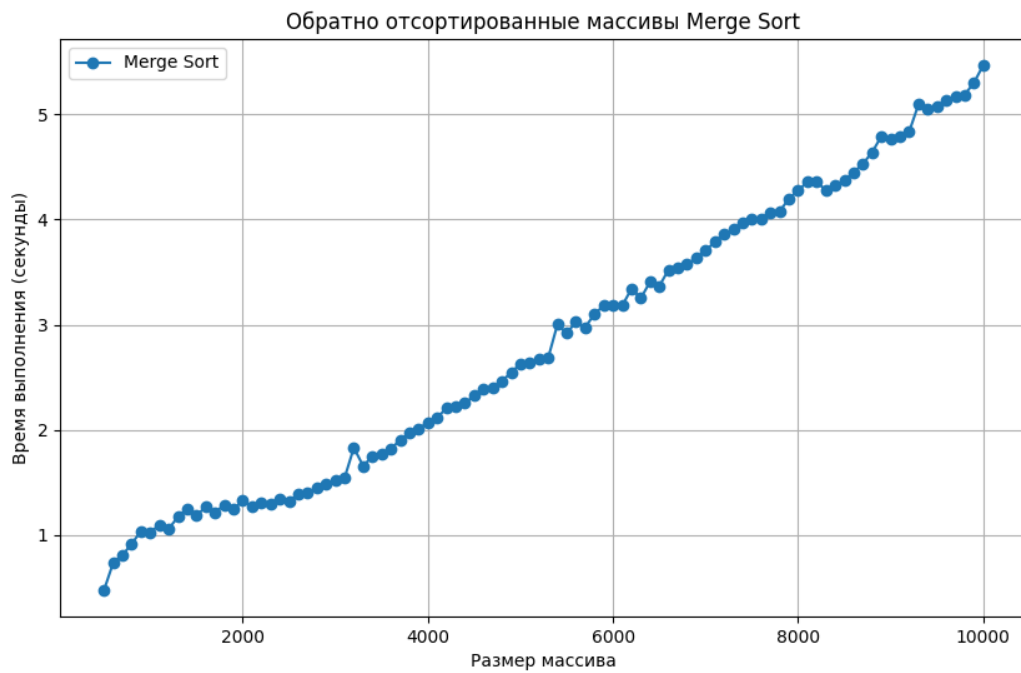
42 → void SortTester::mergeSort(std::vector<int>& array, int left, int right) {
43     if (left >= right) {
44         return;
45     }
46
47     int mid = left + (right - left) / 2;
48
49     mergeSort(array, left, mid);
50     mergeSort(array, mid + 1, right);
51
52     merge(array, left, mid, right);
53 }
54
55 → double SortTester::measureMergeSort(std::vector<int> array) {
56     auto start : time_point<...> = std::chrono::high_resolution_clock::now();
57     mergeSort(array, 0, array.size() - 1);
58     auto elapsed : duration<...> = std::chrono::high_resolution_clock::now() - start;
59     return std::chrono::duration<double, std::milli>(elapsed).count();
60 }
61

```

График результат тестирований:

Размеры массивов варьировались от 500 до 10000 элементов.





Этап 3: Эмпирический анализ гибридного алгоритма MERGE+INSERTION SORT

Реализация анализа алгоритма:

```
22 class SortTester {
23 public:
24     static double measureHybridSort(std::vector<int> array);
25     static void hybridSort(std::vector<int>& array, int left, int right);
26 private:
27     static void insertionSort(std::vector<int>& array, int left, int right) {
28         for (int i = left + 1; i <= right; ++i) {
29             int key = array[i];
30             int j = i - 1;
31             while (j >= left && array[j] > key) {
32                 array[j + 1] = array[j];
33                 --j;
34             }
35             array[j + 1] = key;
36         }
37     }
38     static void merge(std::vector<int>& array, int left, int middle, int right) {
39         int n1 = middle - left + 1;
40         int n2 = right - middle;
41
42         std::vector<int> left_array(n1);
43         std::vector<int> right_array(n2);
44
45         for (size_t i = 0; i < n1; ++i) {
46             left_array[i] = array[left + i];
47         }
48         for (size_t i = 0; i < n2; ++i) {
49             right_array[i] = array[middle + 1 + i];
50         }
51
52         int i = 0, j = 0, k = left;
53         while (i < n1 && j < n2) {
54             if (left_array[i] <= right_array[j]) {
55                 array[k++] = left_array[i++];
56             } else {
57                 array[k++] = right_array[j++];
58             }
59         }
60
61         while (i < n1) {
62             array[k++] = left_array[i++];
63         }
64         while (j < n2) {
65             array[k++] = right_array[j++];
66         }
67     }
68 };
69
```

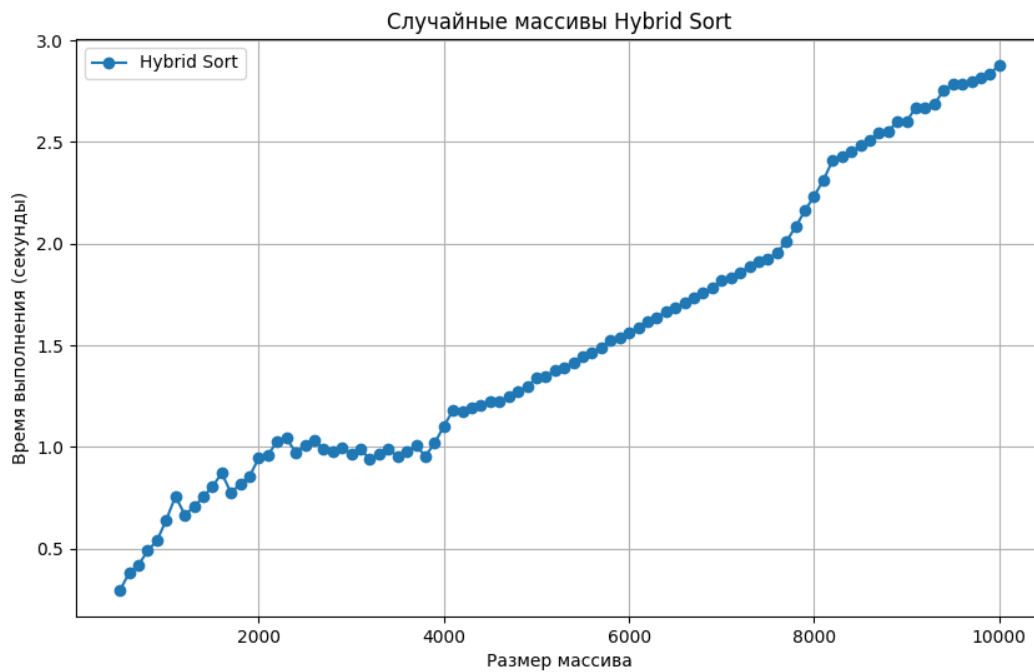
```

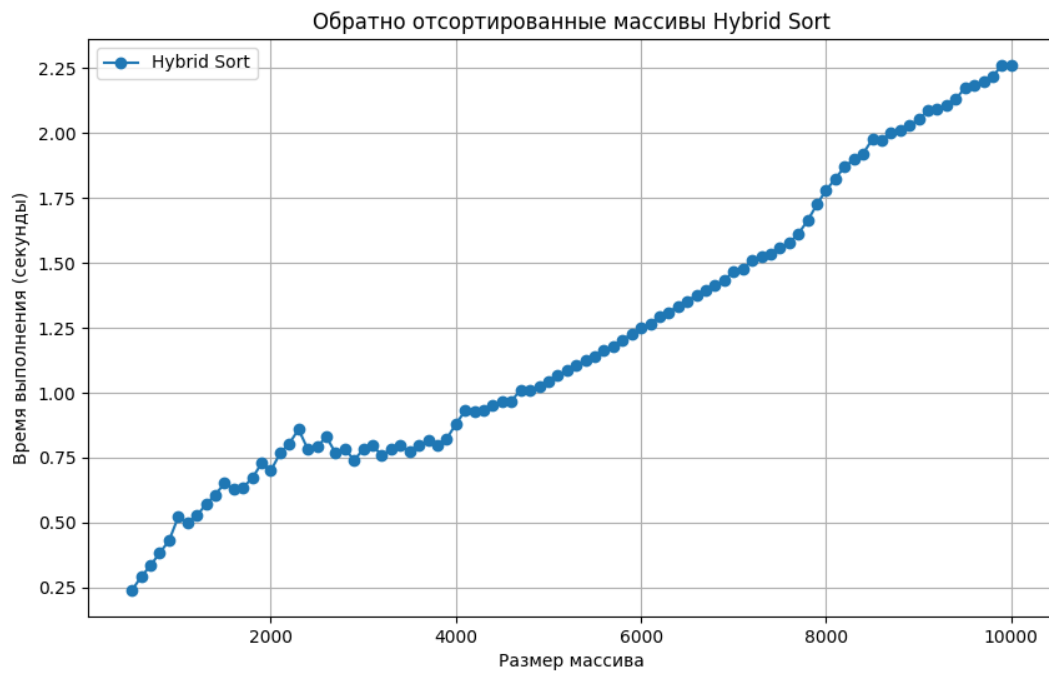
40
41 → void SortTester::hybridSort(std::vector<int>& array, int left, int right) {
42     const int threshold = 15;
43     if (right - left + 1 <= threshold) {
44         insertionSort(array, left, right);
45     } else {
46         int middle = left + (right - left) / 2;
47         hybridSort(array, left, middle);
48         hybridSort(array, middle + 1, right);
49         merge(array, left, middle, right);
50     }
51 }
52
53 → double SortTester::measureHybridSort(std::vector<int> array) {
54     auto start : time_point<...> = std::chrono::high_resolution_clock::now();
55     hybridSort(array, 0, array.size() - 1);
56     auto elapsed : duration<...> = std::chrono::high_resolution_clock::now() - start;
57     return std::chrono::duration<double, std::milli>(elapsed).count();
58 }
59

```

График результат тестирований:

Размеры массивов варьировались от 500 до 10000 элементов.

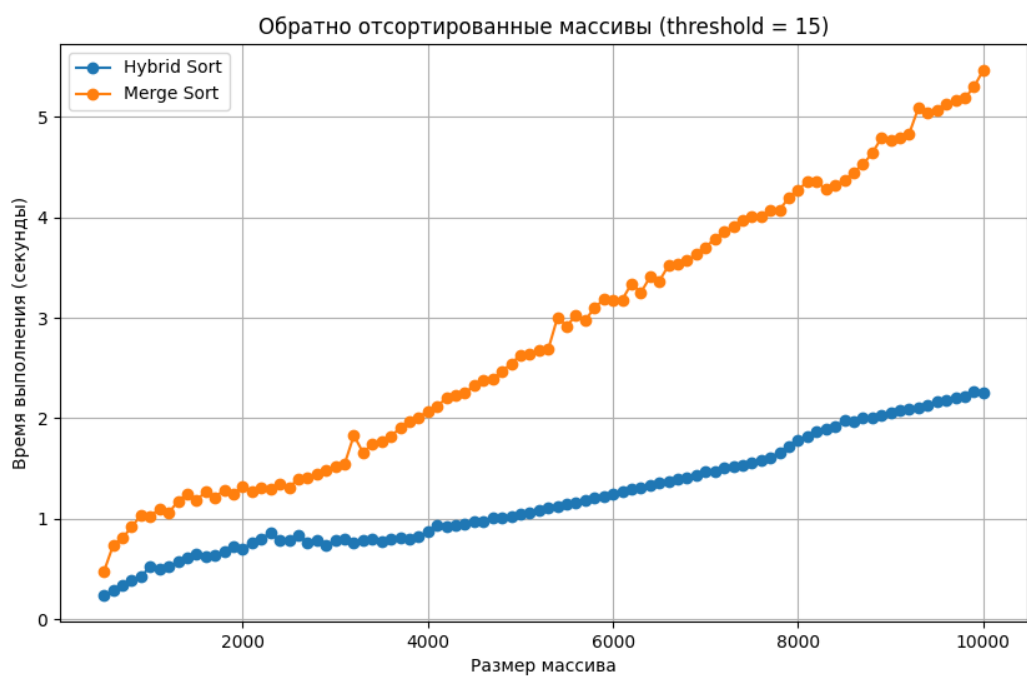
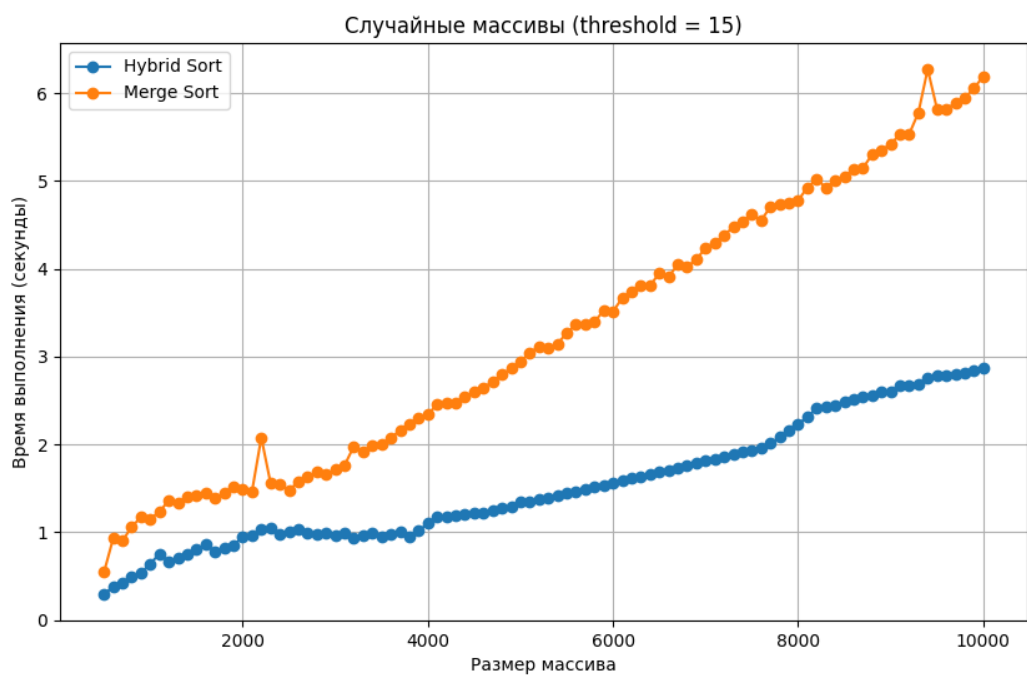


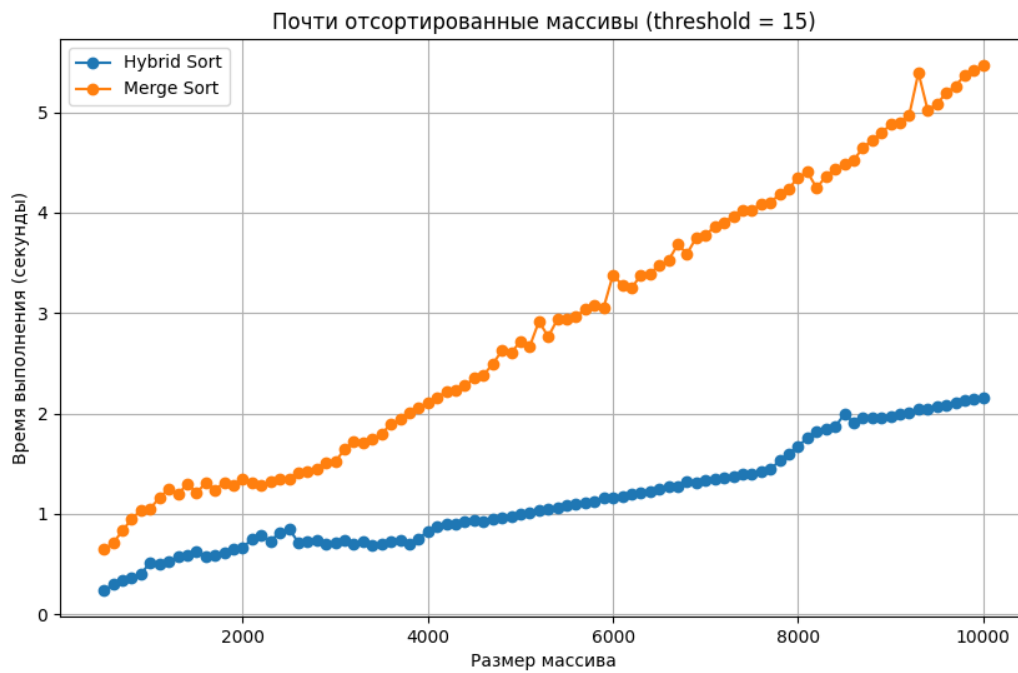


Этап 4: Сравнительный анализ

Давайте сравним два алгоритма с одинаковыми группами:

Переход на Insertion Sort будет происходить при threshold = 15





Здесь мы можем заметить то, что во всех случаях mergeSort+insertionSort будет превосходить по скорости в отличие от mergeSort. Стоит отметить, что в двух группах, в случайных и обратно отсортированных, гибридная сортировка быстрее в 2 раза при больших значениях. А в последней группе она почти в 3 раза быстрее.