Григорьев Владимир. A2 **ID A2i:** 293090927

Исходный код mergeSort+InsertionSort:

```cpp
#include <iostream>
#include <vector>

void insertionSort(std::vector<int>& array, int left, int right) {
  for (int i = left + 1; i <= right; ++i) {
    int key = array[i];
    int j = i - 1;
    while (j >= left && array[j] > key) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = key;
  }
}

void merge(std::vector<int>& array, int left, int middle, int right) {
  int n1 = middle - left + 1;
  int n2 = right - middle;

  std::vector<int> left_array(n1);
  std::vector<int> right_array(n2);

  for (size_t i = 0; i < n1; ++i) {
    left_array[i] = array[left + i];
  }

  for (size_t i = 0; i < n2; ++i) {
    right_array[i] = array[middle + 1 + i];
  }

  int i = 0;
  int j = 0;
  int k = left;

  while (i < n1 && j < n2) {
    if (left_array[i] <= right_array[j]) {
      array[k++] = left_array[i++];
    } else {
      array[k++] = right_array[j++];
    }
  }

  while (i < n1) {
    array[k++] = left_array[i++];
  }

  while (j < n2) {
    array[k++] = right_array[j++];
  }
}

void hybridSort(std::vector<int>& array, int left, int right) {
  const int threshold = 15;

  if (right - left + 1 <= threshold) {
    insertionSort(array, left, right);
  } else {
    int middle = left + (right - left) / 2;
    hybridSort(array, left, middle);
    hybridSort(array, middle + 1, right);
    merge(array, left, middle, right);
  }
}

int main() {
  int n;
  std::cin >> n;
```

main

65:13 LF UTF-8 .clang-tidy 2 spaces C++: 5

```cpp
}

int main() {
  int n;
  std::cin >> n;

  if (n == 0) {
    return 0;
  }

  std::vector<int> array(n);
  for (size_t i = 0; i < n; ++i) {
    std::cin >> array[i];
  }

  hybridSort(array, 0, n - 1);

  for (size_t i = 0; i < n; ++i) {
    std::cout << array[i] << " ";
  }

  std::cout << "\n";
  return 0;
}

```

Этап 1: Реализация класса **ArrayGenerator**

```cpp

struct TestResult {
    int size;
    std::string type;
    double time;

    bool operator<(const TestResult& other) const;
};

class ArrayGenerator {
public:
    static std::vector<int> generateRandomArray(int size, int range);
    static std::vector<int> generateReversedArray(int size);
    static std::vector<int> generateNearlySortedArray(int size, int swaps);
};

```

```cpp
bool TestResult::operator<(const TestResult& other) const {
  if (size == other.size) {
    return type < other.type;
  }
  return size < other.size;
}

std::vector<int> ArrayGenerator::generateRandomArray(int size, int range) {
  std::vector<int> array(size);
  std::mt19937 gen(std::random_device{}());
  std::uniform_int_distribution<> dist(0, range);
  std::generate(array.begin(), array.end(), [&]() -> int  { return dist(gen); });
  return array;
}

std::vector<int> ArrayGenerator::generateReversedArray(int size) {
  std::vector<int> array(size);
  std::iota(array.rbegin(), array.rend(), 0);
  return array;
}

std::vector<int> ArrayGenerator::generateNearlySortedArray(int size, int swaps) {
  std::vector<int> array(size);
  std::iota(array.begin(), array.end(), 0);
  std::mt19937 gen(std::random_device{}());
  for (int i = 0; i < swaps; ++i) {
    int idx1 = gen() % size;
    int idx2 = gen() % size;
    std::swap(array[idx1], array[idx2]);
  }
  return array;
}
```

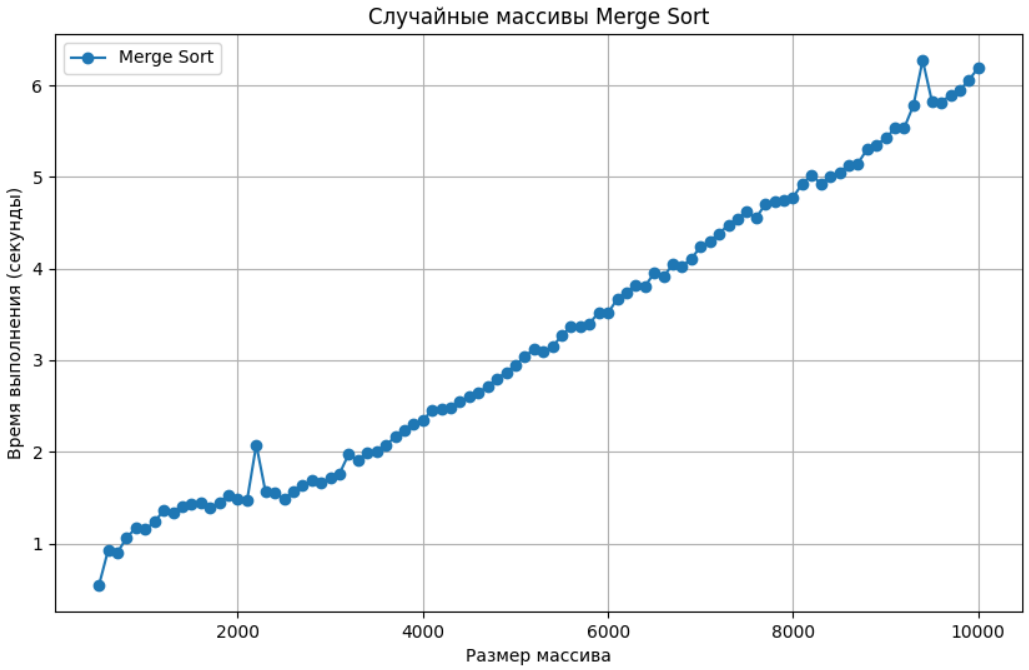
Для удобства была сделана структура **TestResult**, которая хранит размер массива, тип измеряемого массива (рандомный, отсортирован в обратном порядке и почти отсортирован).

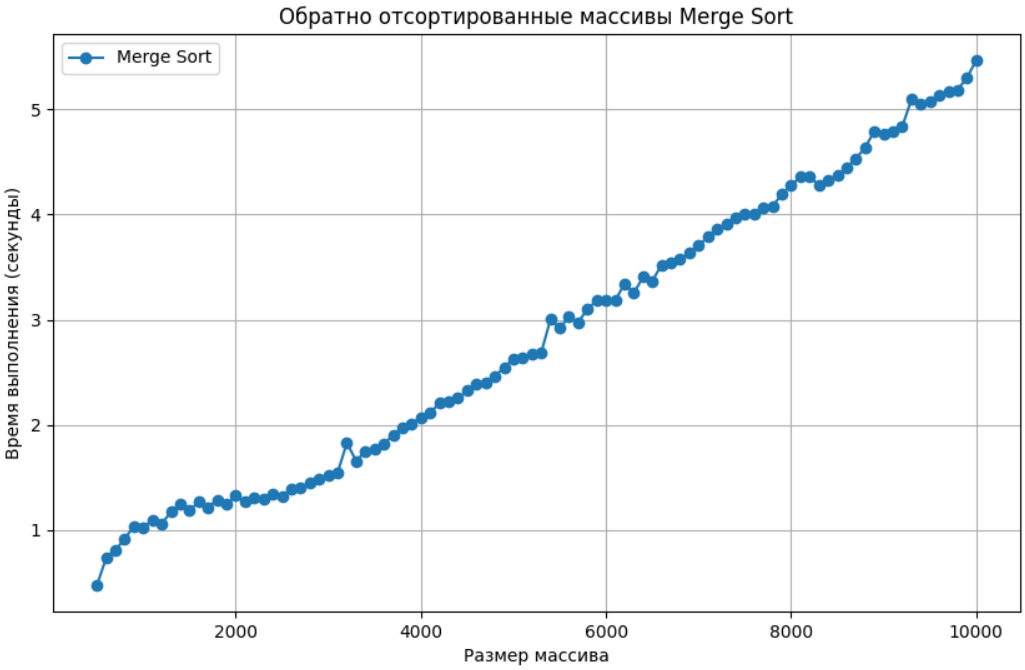
Этап 2: **Эмпирический анализ стандартного алгоритма MERGE SORT**
Реализация анализа алгортма:

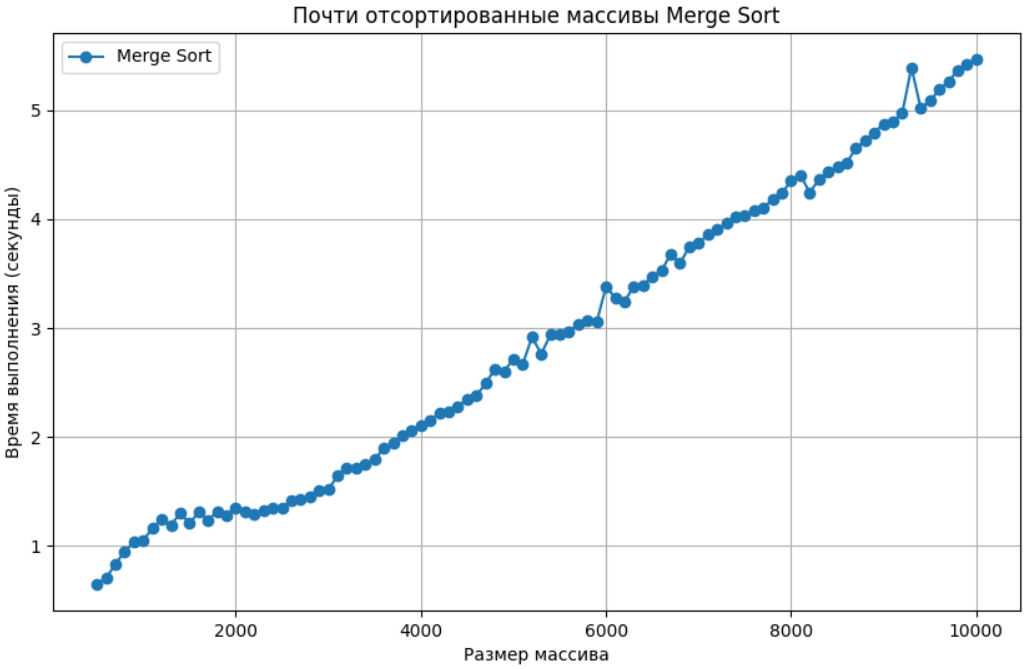
```cpp
class SortTester {
public:
    static double measureMergeSort(std::vector<int> array);
    static void mergeSort(std::vector<int>& array, int left, int right);
private:
    static void merge(std::vector<int>& array, int left, int middle, int right) {
      int n1 = middle - left + 1;
      int n2 = right - middle;

      std::vector<int> left_array(n1);
      std::vector<int> right_array(n2);

      for (size_t i = 0; i < n1; ++i) {
        left_array[i] = array[left + i];
      }

      for (size_t i = 0; i < n2; ++i) {
        right_array[i] = array[middle + 1 + i];
      }

      int i = 0;
      int j = 0;
      int k = left;

      while (i < n1 && j < n2) {
        if (left_array[i] <= right_array[j]) {
          array[k++] = left_array[i++];
        } else {
          array[k++] = right_array[j++];
        }
      }

      while (i < n1) {
        array[k++] = left_array[i++];
      }

      while (i < n1 && j < n2) {
        if (left_array[i] <= right_array[j]) {
          array[k++] = left_array[i++];
        } else {
          array[k++] = right_array[j++];
        }
      }

      while (i < n1) {
        array[k++] = left_array[i++];
      }

      while (j < n2) {
        array[k++] = right_array[j++];
      }
    }
};
```

```cpp
void SortTester::mergeSort(std::vector<int>& array, int left, int right) {
  if (left >= right) {
    return;
  }

  int mid = left + (right - left) / 2;

  mergeSort(array, left, mid);
  mergeSort(array, mid + 1, right);

  merge(array, left, mid, right);
}

double SortTester::measureMergeSort(std::vector<int> array) {
  auto start : time_point<...>  = std::chrono::high_resolution_clock::now();
  mergeSort(array, 0, array.size() - 1);
  auto elapsed : duration<...>  = std::chrono::high_resolution_clock::now() - start;
  return std::chrono::duration<double, std::milli>(elapsed).count();
}
```

График результат тестирований:
Размеры массивов варьировались от 500 до 10000 элементов.







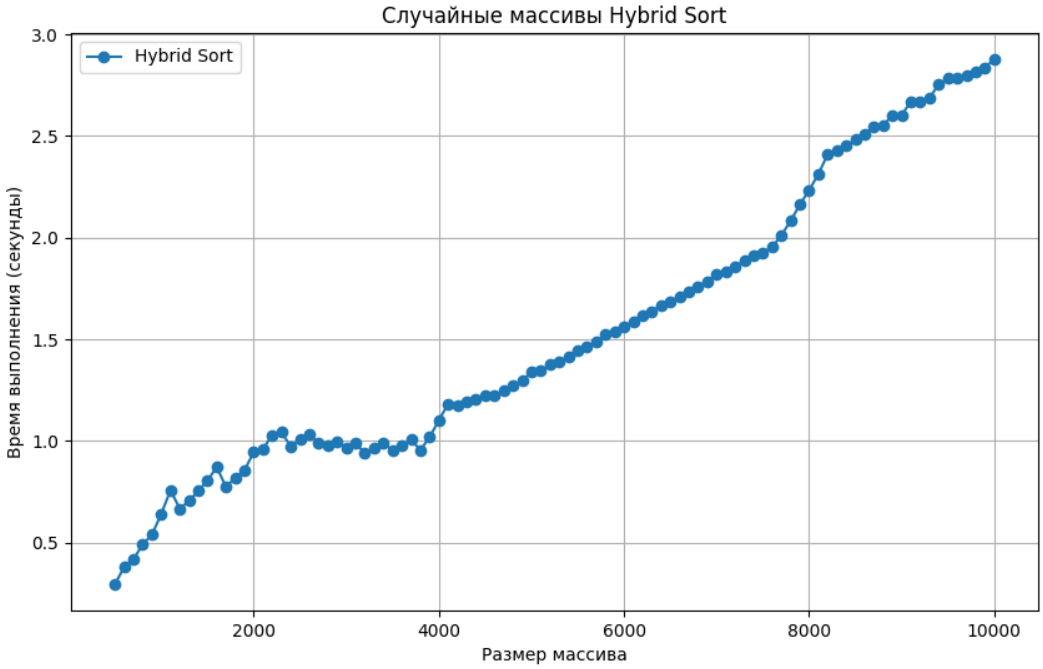
Этап 3: **Эмпирический анализ гибридного алгоритма MERGE+INSERTION SORT**

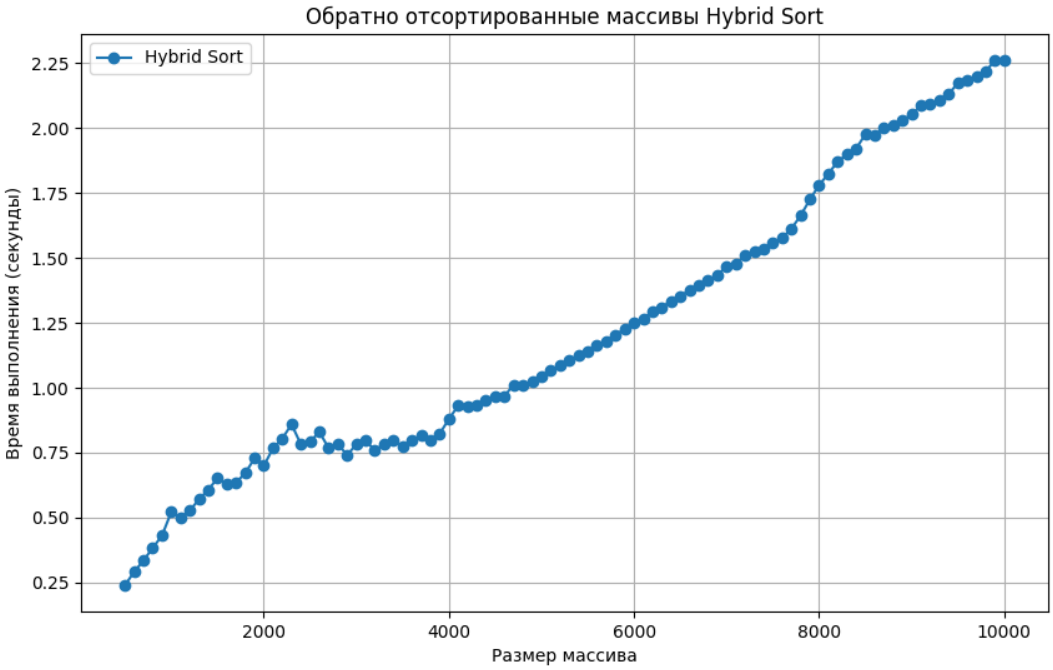
Реализация анализа алгоритма:

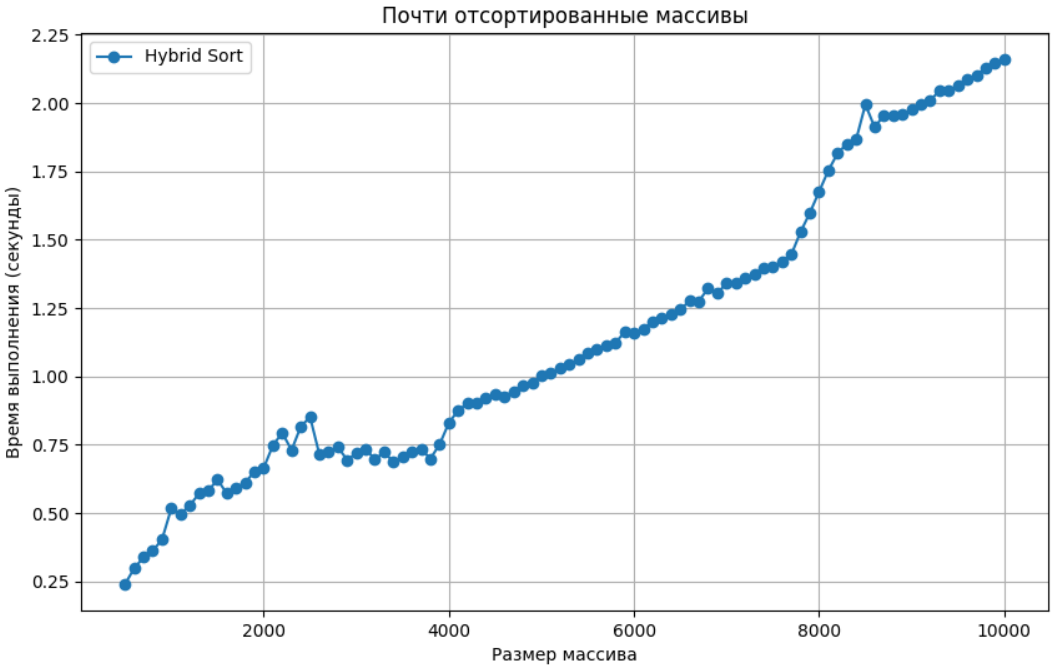
```cpp
class SortTester {
public:
    static double measureHybridSort(std::vector<int> array);
    static void hybridSort(std::vector<int>& array, int left, int right);
private:
    static void insertionSort(std::vector<int>& array, int left, int right) {
      for (int i = left + 1; i <= right; ++i) {
        int key = array[i];
        int j = i - 1;
        while (j >= left && array[j] > key) {
          array[j + 1] = array[j];
          --j;
        }
        array[j + 1] = key;
      }
    }
    static void merge(std::vector<int>& array, int left, int middle, int right) {
      int n1 = middle - left + 1;
      int n2 = right - middle;

      std::vector<int> left_array(n1);
      std::vector<int> right_array(n2);

      for (size_t i = 0; i < n1; ++i) {
        left_array[i] = array[left + i];
      }
      for (size_t i = 0; i < n2; ++i) {
        right_array[i] = array[middle + 1 + i];
      }

      int i = 0, j = 0, k = left;
      while (i < n1 && j < n2) {
        if (left_array[i] <= right_array[j]) {
          array[k++] = left_array[i++];
        } else {
          array[k++] = right_array[j++];
        }
      }

      while (i < n1) {
        array[k++] = left_array[i++];
      }
      while (j < n2) {
        array[k++] = right_array[j++];
      }
    }
};
```

```cpp
void SortTester::hybridSort(std::vector<int>& array, int left, int right) {
  const int threshold = 15;
  if (right - left + 1 <= threshold) {
    insertionSort(array, left, right);
  } else {
    int middle = left + (right - left) / 2;
    hybridSort(array, left, middle);
    hybridSort(array, middle + 1, right);
    merge(array, left, middle, right);
  }
}

double SortTester::measureHybridSort(std::vector<int> array) {
  auto start : time_point<...>  = std::chrono::high_resolution_clock::now();
  hybridSort(array, 0, array.size() - 1);
  auto elapsed : duration<...>  = std::chrono::high_resolution_clock::now() - start;
  return std::chrono::duration<double, std::milli>(elapsed).count();
}
```

График результат тестирований:
Размеры массивов варьировались от 500 до 10000 элементов.



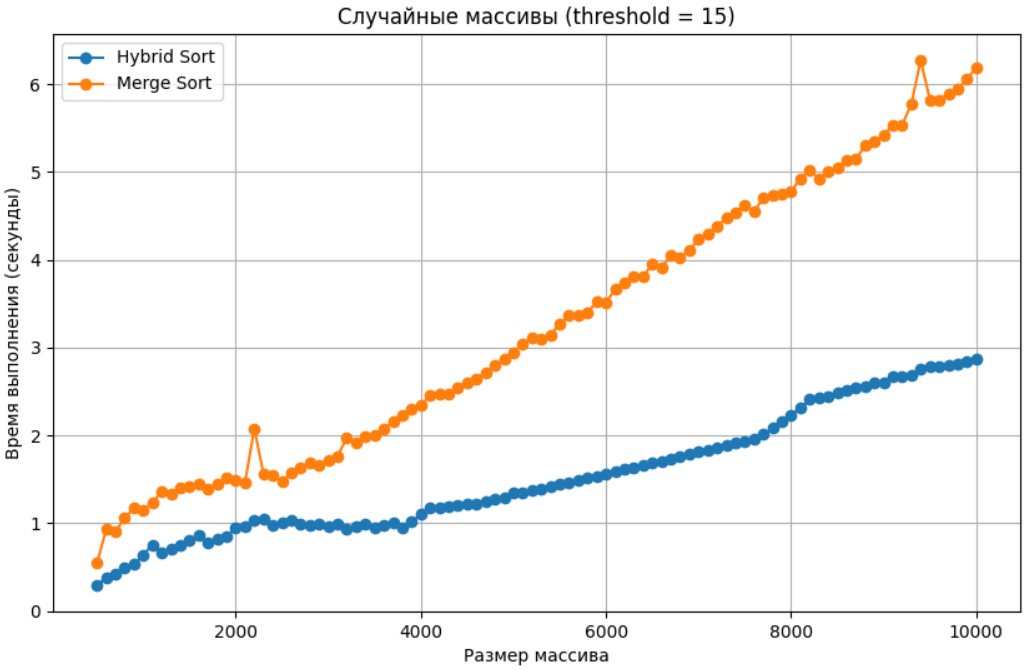


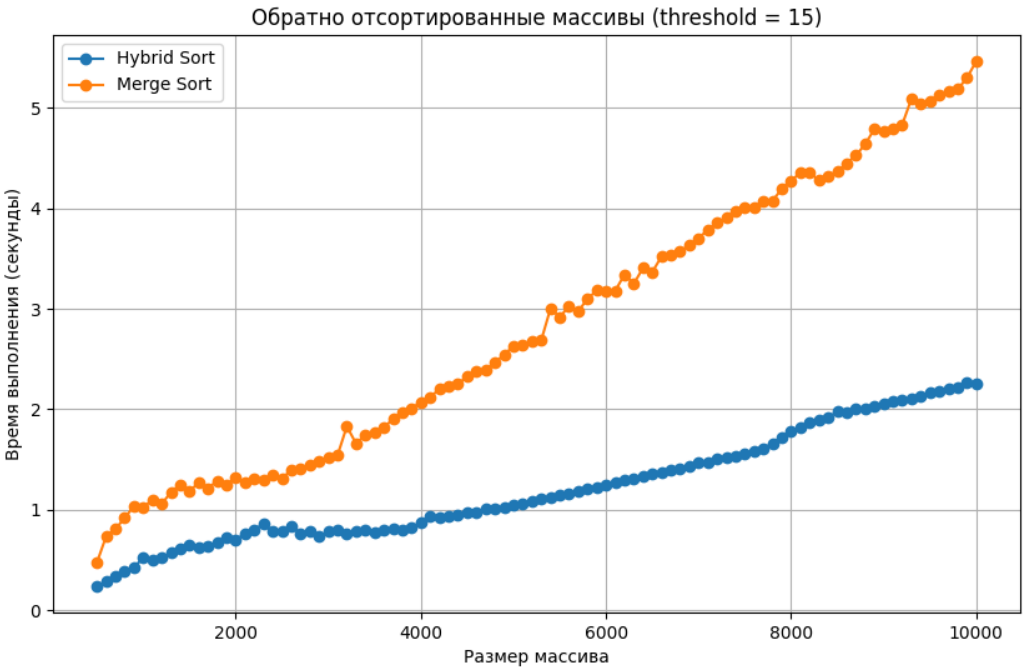


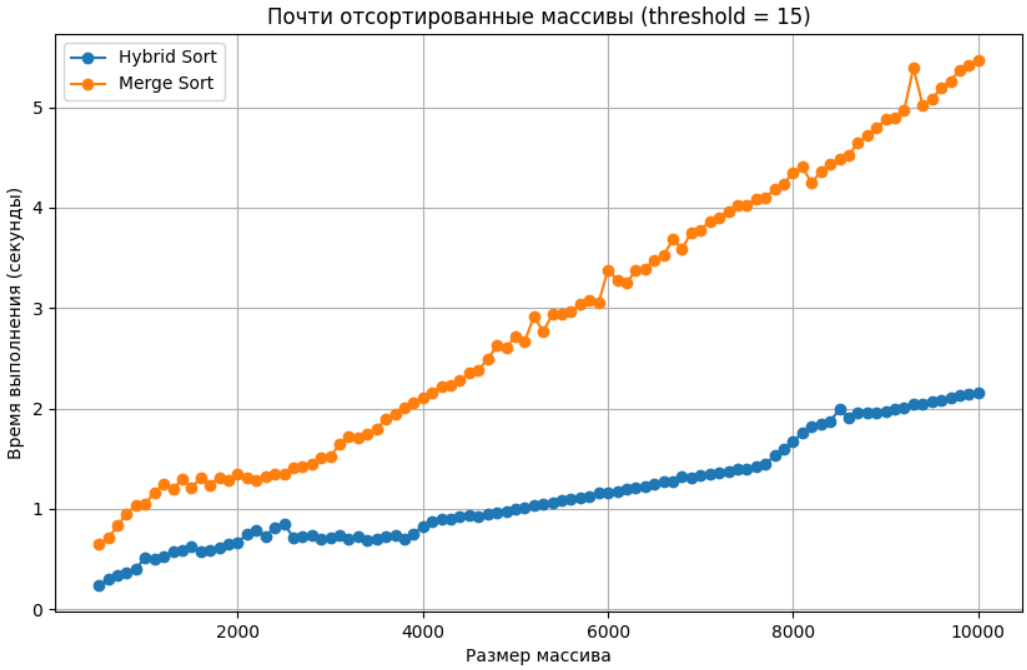
Этап 4: <u>Сравнительный анализ</u>

Давайте сравним два алгоритма с одинаковыми группами:

Переход на Insertion Sort будет происходить при threshold = 15


Случайные массивы (threshold = 15)
Hybrid Sort
Merge Sort
Время выполнения (секунды)
Размер массива
0
1
2
3
4
5
6
2000
4000
6000
8000
10000


Обратно отсортированные массивы (threshold = 15)
Hybrid Sort
Merge Sort
Время выполнения (секунды)
Размер массива
0
1
2
3
4
5
2000
4000
6000
8000
10000



Здесь мы можем заметить то, что во всех случаях mergeSort+insertionSort будет превосходствовать по скорости в отличии 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.