Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

Выполнил студент группы КС-36 Сары Кристина Ивановна

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/SaryKI\_36\_ALG.git

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 24.02.2025

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

В рамках лабораторной работы необходимо изучить и реализовать метод сортировки слиянием (вариант 2). Лабораторная работа предполагает анализ алгоритма с точки зрения временной сложности, количества вызовов рекурсивной функции, глубины рекурсии, времени исполнения и потребления дополнительной памяти. Исследование сопровождается построением графиков для различных параметров алгоритма.

**Задание: Вариант 2**

Используя предыдущий код посерийного выполнения алгоритма сортировки и измерения времени требутеся реализовать метод сортировки слиянием.

* Реализовать проведения тестирования алгоритма сериями расчетов для измерения параметров времени.  
  За один расчет выполняется следующие операции:
  1. Генерируется массив случайных значений
  2. Запоминается время начала расчета алгоритма сортировки
  3. Вполняется алгоритм сортировки
  4. Вычисляется время затраченное на сортировку: текущее время - время начала
  5. Сохраняется время для одной попытки После этого расчет повторяется до окончания серии.
  6. Алгоритм вычисляется 8 сериями по 20 раз за серию.
  7. Алгоритм в каждой серии вычисляется для массива размером M. (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000)
  8. Массив заполняется значения числами с плавающей запятой в интервале от -1 до 1.
  9. Для серии запоминаются все времена которые были замерены
* При работе сортировки подсчитать:
  1. колличество вызовов рекурсивной функции
  2. глубину рекурсии
  3. времени исполнения сортировки
  4. максимальное потребление дополнительной памяти (память за вычетом памяти до начала сортировки)
* По полученным данным времени построить графики зависимости времени от числа элементов в массиве:
  1. Совмешеееый график наихудшего времени выполнения сортировки и сложности алгоритма указанной в нотации O большое.

Для построения графика вычисляется O большое для каждого размера массива. При этом при вычислении функции O(c \* g(N)) подбирается такая константа с, что бы при значении > 1000 график O(N) был выше графика наихудшего случая, но второй график на его фоне не превращался в прямую линию

* 1. Совмещенный график среднего, наилучшего и наихудшего времени исполнения.
  2. Совмещенный график среднего, наилучшего и наихудшего глубины рекурсии.
  3. Совмещенный график среднего, наилучшего и наихудшего потребления дополнительной памяти.
  4. Совмещенный график среднего, наилучшего и наихудшего вызовов рекурсивной функции.
* По результатом расчетов оформляется отчет по предоставленной форме, в отчете:
  1. Приводится описание алгоритма.
  2. Приводится описания выполнения задачи (Описание кода и спецефических элементов реализации)
  3. Приводятся выводы (Графики и их анализ). Требуется ответить на вопрос о поведении алгоритма изученного в процессе выполнения лаборатной работы и зафиксировать его особенности.

# Описание метода/модели.

Сортировка слиянием (**Merge Sort**) — это алгоритм сортировки, который использует принцип **"разделяй и властвуй"**. Он рекурсивно делит массив на две части, сортирует их и затем **сливает** обратно в один отсортированный массив.

**Принцип работы**

1. **Разделение массива**
   * Исходный массив **разделяется пополам** до тех пор, пока не останутся массивы длиной 1.
2. **Сортировка и слияние**
   * После разбиения массивов выполняется **обратный процесс слияния**:
     + Берутся два отсортированных подмассива.
     + Они **сравниваются и объединяются** в один отсортированный массив.

**Анализ сложности выполнения сортировки выбором:**

Merge Sort выполняется за O(n log n) во всех случаях:

**Лучший случай: O(n log n)**

**Средний случай: O(n log n)**

**Худший случай: O(n log n)**

**Уровень потребляемой памяти**

**Дополнительная память: O(n)**

* Так как создаются временные массивы для хранения подмассивов при слиянии, Merge Sort требует **дополнительную память**.
* В отличие от **быстрой сортировки (Quick Sort), которая использует O(log n) памяти**, Merge Sort требует **O(n), что является его недостатком**.

**Преимущества и недостатки алгоритма**

**Преимущества:**

* **Стабильность** – одинаковые элементы сохраняют свой первоначальный порядок в массиве.
* **Предсказуемость** – работает за O(n log n) в любом случае, в отличие от быстрой сортировки.
* Подходит для сортировки больших данных – особенно при работе с связными списками, так как не требует случайного доступа к памяти.
* **Хорош для многопоточной обработки** – можно легко распараллелить слияние.

**Недостатки:**

* **Большое потребление памяти (O(n))** – создаются временные массивы при слиянии.
* **Медленнее Quick Sort в среднем** – из-за большего количества вспомогательных операций и использования дополнительной памяти.\
* **Неэффективен для небольших массивов** – из-за рекурсивных вызовов и выделения памяти.

# Выполнение задачи.

Программа реализована на языке C++.

**Код:**

#include <iostream>

#include <vector>

#include <chrono>

#include <random>

#include <algorithm>

#include <limits>

#include <fstream>

#include <windows.h>

#include <psapi.h> // Для GetProcessMemoryInfo

using namespace std;

struct MergeSortTracker {

int recursion\_calls = 0; // Счётчик вызовов рекурсивной функции

int max\_depth = 0; // Максимальная глубина рекурсии

int current\_depth = 0; // Текущая глубина рекурсии

size\_t max\_memory\_used = 0; // Максимальное использование памяти

void mergeSort(vector<double>& arr, int left, int right) {

recursion\_calls++; // Увеличиваем счётчик вызовов рекурсии

current\_depth++; // Увеличиваем текущую глубину рекурсии

max\_depth = max(max\_depth, current\_depth); // Обновляем максимальную глубину

max\_memory\_used = max(max\_memory\_used, getMemoryUsage()); // Обновляем максимальную память

// Базовый случай рекурсии

if (left >= right) {

current\_depth--; // Возвращаемся на уровень вверх по рекурсии (когда левая граница больше

// или равна правой, значит, массив уже отсортирован — возвращаемся назад.)

return;

}

// массив делится на две части, а затем обе части сортируются рекурсивно.

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

mergeSort(arr, left, mid); // Рекурсивно сортируем левую часть

mergeSort(arr, mid + 1, right); // Рекурсивно сортируем правую часть

merge(arr, left, mid, right); // Сливаем отсортированные части

current\_depth--; // Возвращаемся на уровень вверх по рекурсии

}

//Слияние двух отсортированных массивов

void merge(vector<double>& arr, int left, int mid, int right) {

vector<double> leftArr(arr.begin() + left, arr.begin() + mid + 1);

vector<double> rightArr(arr.begin() + mid + 1, arr.begin() + right + 1);

//Создаём два временных массива:

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

while (i < leftArr.size() && j < rightArr.size()) {

if (leftArr[i] < rightArr[j]) {

arr[k++] = leftArr[i++];

} else {

arr[k++] = rightArr[j++];

}

}

// Сравниваем элементы из двух половин и помещаем наименьший из них в arr

while (i < leftArr.size()) arr[k++] = leftArr[i++];

while (j < rightArr.size()) arr[k++] = rightArr[j++];

}

// Функция для получения текущего использования памяти (в KB)

size\_t getMemoryUsage() {

PROCESS\_MEMORY\_COUNTERS memInfo;

GetProcessMemoryInfo(GetCurrentProcess(), &memInfo, sizeof(memInfo));

return memInfo.WorkingSetSize / 1024; // Возвращаем в килобайтах

}

};

void runTests() {

vector<int> sizes = {1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000};

int series = 20;

random\_device rd;

mt19937 gen(rd());

uniform\_real\_distribution<double> dist(-1.0, 1.0);

ofstream results\_file("merge\_sort\_results.csv");

results\_file << "Size,AvgTime,BestTime,WorstTime,AvgRecursion,BestRecursion,WorstRecursion,"

"AvgDepth,BestDepth,WorstDepth,AvgMemory,BestMemory,WorstMemory\n";

// Для каждого размера массива создаём списки для хранения результатов.

for (int size : sizes) {

vector<double> time\_results, recursion\_results, depth\_results, memory\_results;

//заполняем массив случайными числами и создаём объект tracker

for (int i = 0; i < series; ++i) {

vector<double> arr(size);

generate(arr.begin(), arr.end(), [&]() { return dist(gen); });

MergeSortTracker tracker;

size\_t memory\_before = tracker.getMemoryUsage();

auto start\_time = chrono::high\_resolution\_clock::now();

//Фиксируем стартовое использование памяти и время.

tracker.mergeSort(arr, 0, size - 1);

auto end\_time = chrono::high\_resolution\_clock::now();

size\_t memory\_after = tracker.getMemoryUsage();

// Учитываем корректное потребление памяти

size\_t memory\_used = (memory\_after >= memory\_before) ? (memory\_after - memory\_before) : 0;

memory\_used = max(memory\_used, tracker.max\_memory\_used); // Учитываем максимальное потребление

chrono::duration<double> elapsed = end\_time - start\_time;

time\_results.push\_back(elapsed.count());

recursion\_results.push\_back(tracker.recursion\_calls);

depth\_results.push\_back(tracker.max\_depth);

memory\_results.push\_back(memory\_used);

// Вывод результатов в консоль

cout << "Size: " << size << " | Time: " << elapsed.count() << " sec"

<< " | Recursion Calls: " << tracker.recursion\_calls

<< " | Max Depth: " << tracker.max\_depth

<< " | Memory Used: " << memory\_used << " KB" << endl;

}

double avg\_time = accumulate(time\_results.begin(), time\_results.end(), 0.0) / series;

double best\_time = \*min\_element(time\_results.begin(), time\_results.end());

double worst\_time = \*max\_element(time\_results.begin(), time\_results.end());

double avg\_recursion\_calls = accumulate(recursion\_results.begin(), recursion\_results.end(), 0.0) / series;

double best\_recursion\_calls = \*min\_element(recursion\_results.begin(), recursion\_results.end());

double worst\_recursion\_calls = \*max\_element(recursion\_results.begin(), recursion\_results.end());

double avg\_max\_depth = accumulate(depth\_results.begin(), depth\_results.end(), 0.0) / series;

double best\_max\_depth = \*min\_element(depth\_results.begin(), depth\_results.end());

double worst\_max\_depth = \*max\_element(depth\_results.begin(), depth\_results.end());

double avg\_memory = accumulate(memory\_results.begin(), memory\_results.end(), 0.0) / series;

double best\_memory = \*min\_element(memory\_results.begin(), memory\_results.end());

double worst\_memory = \*max\_element(memory\_results.begin(), memory\_results.end());

results\_file << size << "," << avg\_time << "," << best\_time << "," << worst\_time << ","

<< avg\_recursion\_calls << "," << best\_recursion\_calls << "," << worst\_recursion\_calls << ","

<< avg\_max\_depth << "," << best\_max\_depth << "," << worst\_max\_depth << ","

<< avg\_memory << "," << best\_memory << "," << worst\_memory << "\n";

}

results\_file.close();

}

int main() {

runTests();

return 0;

}

**Полученные результаты:**

C:\Users\user\CLionProjects\untitled14\cmake-build-debug\untitled14.exe

Size: 1000 | Time: 0.0008237 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3772 KB

Size: 1000 | Time: 0.0008031 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3876 KB

Size: 1000 | Time: 0.0008695 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3876 KB

Size: 1000 | Time: 0.0009283 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3876 KB

Size: 1000 | Time: 0.0008151 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3876 KB

Size: 1000 | Time: 0.0008506 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3880 KB

Size: 1000 | Time: 0.0009584 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3880 KB

Size: 1000 | Time: 0.0007846 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3880 KB

Size: 1000 | Time: 0.0008044 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3880 KB

Size: 1000 | Time: 0.0008091 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3880 KB

Size: 1000 | Time: 0.0008192 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 1000 | Time: 0.0007989 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 1000 | Time: 0.0007835 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 1000 | Time: 0.0007447 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 1000 | Time: 0.0007827 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 1000 | Time: 0.0007878 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 1000 | Time: 0.0008205 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 1000 | Time: 0.000784 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 1000 | Time: 0.0008936 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 1000 | Time: 0.0008997 sec | Recursion Calls: 1999 | Max Depth: 11 | Memory Used: 3884 KB

Size: 2000 | Time: 0.0016025 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3900 KB

Size: 2000 | Time: 0.0017224 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0015624 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0016886 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0015488 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0015397 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.001887 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0015635 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0015374 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0015596 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0015522 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.001549 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0016314 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0016335 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0016973 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0016878 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3904 KB

Size: 2000 | Time: 0.0015316 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3912 KB

Size: 2000 | Time: 0.0015512 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3912 KB

Size: 2000 | Time: 0.0015545 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3912 KB

Size: 2000 | Time: 0.0016297 sec | Recursion Calls: 3999 | Max Depth: 12 | Memory Used: 3912 KB

Size: 4000 | Time: 0.0031622 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031564 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3932 KB

Size: 4000 | Time: 0.0031831 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3932 KB

Size: 4000 | Time: 0.0032175 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3932 KB

Size: 4000 | Time: 0.0031472 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3932 KB

Size: 4000 | Time: 0.0031833 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3932 KB

Size: 4000 | Time: 0.0031693 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031612 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031396 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031839 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031462 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031659 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031627 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031934 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031598 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031604 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031637 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3944 KB

Size: 4000 | Time: 0.0031299 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3952 KB

Size: 4000 | Time: 0.0031365 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3952 KB

Size: 4000 | Time: 0.0033421 sec | Recursion Calls: 7999 | Max Depth: 13 | Memory Used: 3952 KB

Size: 8000 | Time: 0.006656 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0066663 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0066489 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.006655 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0066754 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.006747 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0068464 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0066973 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0065992 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0066981 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0068421 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0067326 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0067069 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0067391 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0067606 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0066453 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0067294 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0068926 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.0067361 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 8000 | Time: 0.006628 sec | Recursion Calls: 15999 | Max Depth: 14 | Memory Used: 3988 KB

Size: 16000 | Time: 0.0133801 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4072 KB

Size: 16000 | Time: 0.0134275 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4072 KB

Size: 16000 | Time: 0.0131213 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4072 KB

Size: 16000 | Time: 0.0131425 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4072 KB

Size: 16000 | Time: 0.0130817 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4100 KB

Size: 16000 | Time: 0.0133279 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4100 KB

Size: 16000 | Time: 0.0137545 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4072 KB

Size: 16000 | Time: 0.0133879 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4196 KB

Size: 16000 | Time: 0.0133183 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4184 KB

Size: 16000 | Time: 0.0135659 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4100 KB

Size: 16000 | Time: 0.0136688 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4100 KB

Size: 16000 | Time: 0.0132487 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4072 KB

Size: 16000 | Time: 0.0131437 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4196 KB

Size: 16000 | Time: 0.0130523 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4196 KB

Size: 16000 | Time: 0.0130679 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4228 KB

Size: 16000 | Time: 0.013167 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4104 KB

Size: 16000 | Time: 0.0131514 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4208 KB

Size: 16000 | Time: 0.0134034 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4104 KB

Size: 16000 | Time: 0.0131677 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4132 KB

Size: 16000 | Time: 0.0131022 sec | Recursion Calls: 31999 | Max Depth: 15 | Memory Used: 4228 KB

Size: 32000 | Time: 0.0265378 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4412 KB

Size: 32000 | Time: 0.0263008 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4356 KB

Size: 32000 | Time: 0.0263897 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4236 KB

Size: 32000 | Time: 0.0265978 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4464 KB

Size: 32000 | Time: 0.0264472 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4464 KB

Size: 32000 | Time: 0.0265231 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4476 KB

Size: 32000 | Time: 0.0266765 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4476 KB

Size: 32000 | Time: 0.0274127 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4344 KB

Size: 32000 | Time: 0.0265014 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4476 KB

Size: 32000 | Time: 0.0264635 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4356 KB

Size: 32000 | Time: 0.0262933 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4456 KB

Size: 32000 | Time: 0.0263532 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4264 KB

Size: 32000 | Time: 0.0264971 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4464 KB

Size: 32000 | Time: 0.0268565 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4576 KB

Size: 32000 | Time: 0.0272008 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4328 KB

Size: 32000 | Time: 0.0264925 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4452 KB

Size: 32000 | Time: 0.0264511 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4576 KB

Size: 32000 | Time: 0.0264115 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4576 KB

Size: 32000 | Time: 0.0264964 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4576 KB

Size: 32000 | Time: 0.0267267 sec | Recursion Calls: 63999 | Max Depth: 16 | Memory Used: 4328 KB

Size: 64000 | Time: 0.0539571 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 4708 KB

Size: 64000 | Time: 0.0535254 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5084 KB

Size: 64000 | Time: 0.0536877 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5208 KB

Size: 64000 | Time: 0.0535717 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5208 KB

Size: 64000 | Time: 0.0536103 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5208 KB

Size: 64000 | Time: 0.0538254 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5208 KB

Size: 64000 | Time: 0.0540095 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5208 KB

Size: 64000 | Time: 0.0538023 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 4836 KB

Size: 64000 | Time: 0.0538335 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 4836 KB

Size: 64000 | Time: 0.0536299 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 4836 KB

Size: 64000 | Time: 0.0536975 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5208 KB

Size: 64000 | Time: 0.0548347 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5084 KB

Size: 64000 | Time: 0.0537914 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5176 KB

Size: 64000 | Time: 0.0540816 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5172 KB

Size: 64000 | Time: 0.053831 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5172 KB

Size: 64000 | Time: 0.0536998 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5172 KB

Size: 64000 | Time: 0.0536218 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5172 KB

Size: 64000 | Time: 0.0536073 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5172 KB

Size: 64000 | Time: 0.0535982 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5172 KB

Size: 64000 | Time: 0.0535584 sec | Recursion Calls: 127999 | Max Depth: 17 | Memory Used: 5172 KB

Size: 128000 | Time: 0.110982 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6172 KB

Size: 128000 | Time: 0.108564 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.108791 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.109195 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.109121 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.10916 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.108967 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.108735 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.109455 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.110949 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.109124 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.108748 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.109367 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6176 KB

Size: 128000 | Time: 0.109425 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6572 KB

Size: 128000 | Time: 0.10882 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6072 KB

Size: 128000 | Time: 0.108911 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6572 KB

Size: 128000 | Time: 0.109049 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6572 KB

Size: 128000 | Time: 0.108836 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6572 KB

Size: 128000 | Time: 0.109046 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6572 KB

Size: 128000 | Time: 0.109152 sec | Recursion Calls: 255999 | Max Depth: 18 | Memory Used: 6572 KB

Process finished with exit code 0

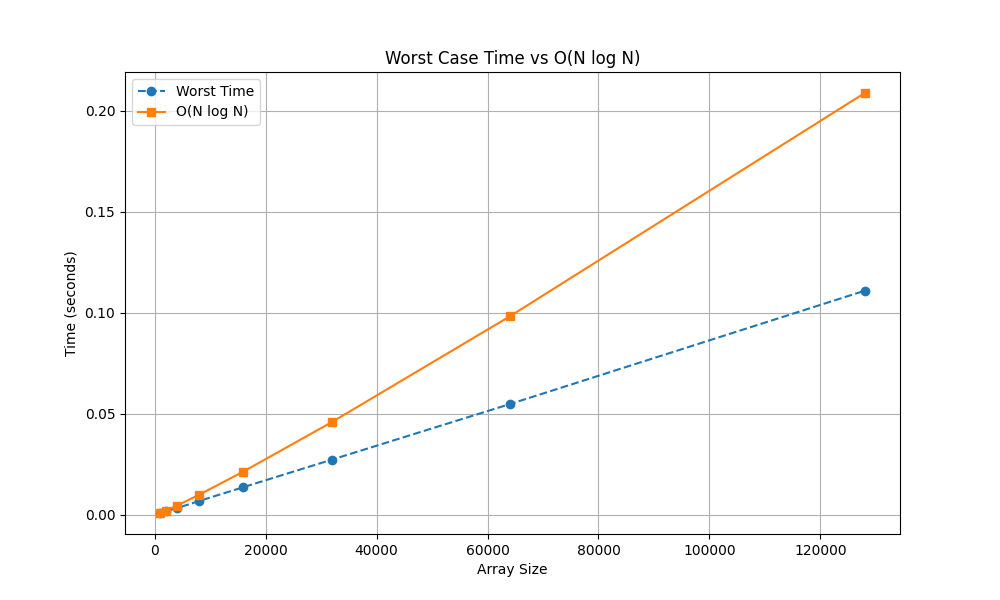
Графики для анализа полученных результатов построены при помощи библиотеки matplotlib языка Python

Рисунок 1 — Совмещённый график наихудшего времени выполнения сортировки и сложности алгоритма указанной в нотации O большое.

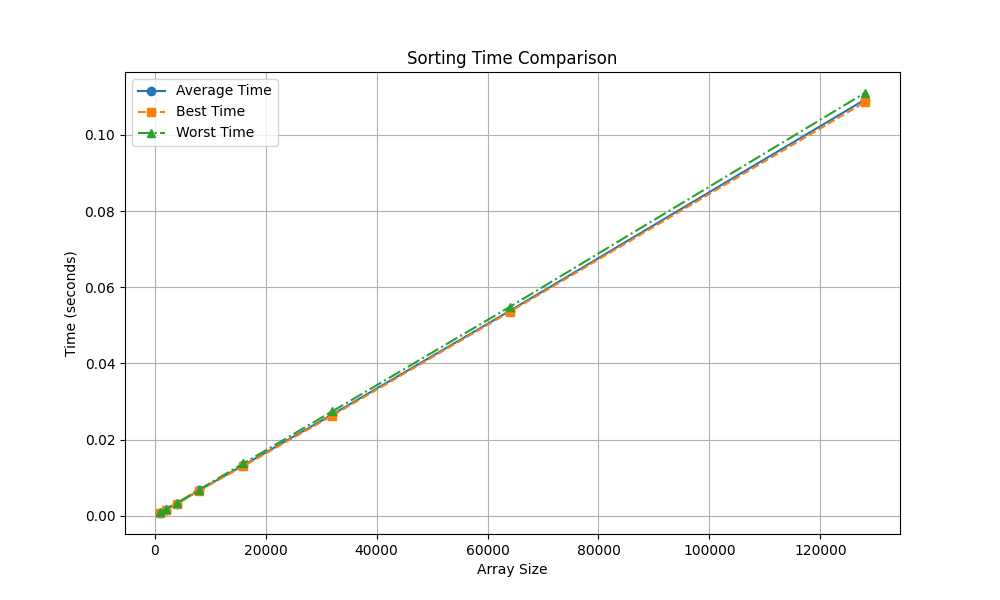


Рисунок 2 — Совмещённый график среднего, наилучшего и наихудшего времени исполнения

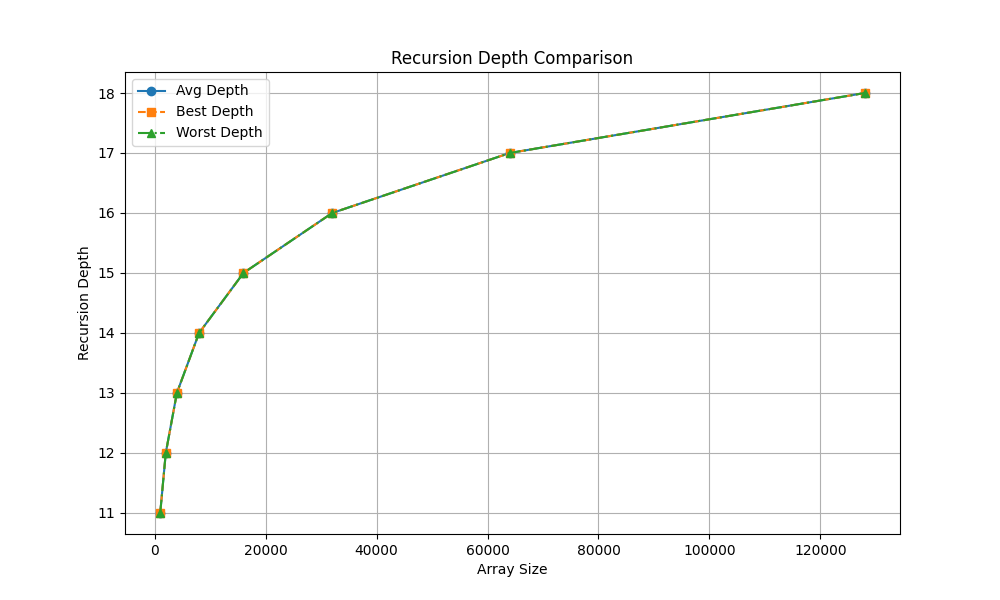


Рисунок 3 — Совмещенный график среднего, наилучшего и наихудшего глубины рекурсии.

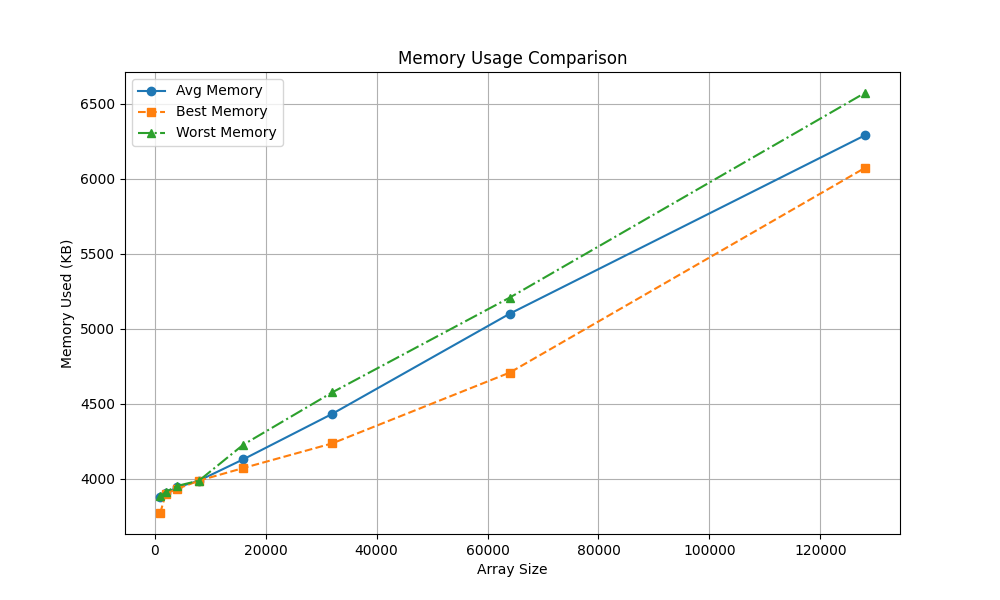


Рисунок 4 — Совмещенный график среднего, наилучшего и наихудшего потребления дополнительной памяти.

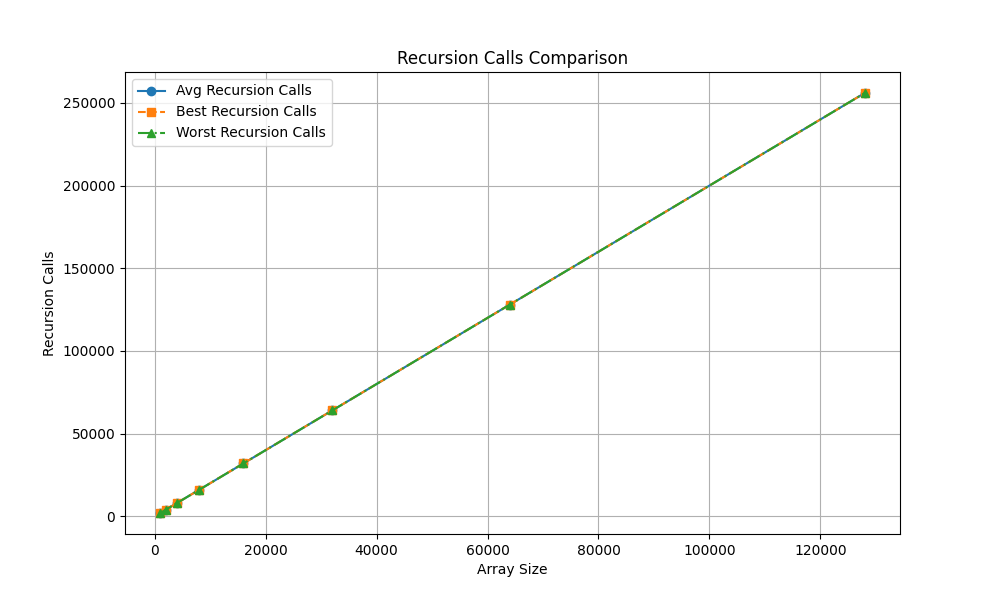


Рисунок 5 — Совмещенный график среднего, наилучшего и наихудшего вызовов рекурсивной функции

# Заключение.

В ходе исследования был рассмотрен алгоритм сортировки слиянием (**Merge Sort**), его принцип работы, сложность и потребление ресурсов. Этот алгоритм демонстрирует стабильную временную сложность **O(n log n)** независимо от исходного порядка элементов в массиве.

Основные **преимущества** метода включают **стабильность, предсказуемость времени работы и возможность параллельной обработки данных**. Однако его **основной недостаток** — это высокое потребление памяти **O(n)** из-за использования временных массивов во время слияния.

Сортировка слиянием **лучше всего подходит для работы с большими массивами данных**, особенно когда стабильность сортировки критична. Однако для небольших массивов или ограниченной памяти **Quick Sort или другие алгоритмы могут быть более эффективными**.

Таким образом, **Merge Sort остается одним из наиболее мощных и универсальных алгоритмов сортировки**, широко используемых в программировании и обработке данных.