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-30 Суханова Евгения Валерьевна
Ссылка на репозиторий:
https://github.com/MUCTR-IKT-CPP/EVSuhanova_30/blob/main/Algoritms/laba2.cpp

Приняли: Пысин Максим Дмитриевич
Краснов Дмитрий Олегович

Дата сдачи: 03.03.2023

Оглавление

Описание задачи.

Необходимо реализовать метод быстрой сортировки.

Для реализованного метода сортировки необходимо провести серию тестов для всех значений N из списка (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000), при этом:

- в каждом тесте необходимо по 20 раз генерировать вектор, состоящий из N элементов
- каждый элемент массива заполняется случайным числом с плавающей запятой от -1 до 1

На основании статьи реализовать проверки негативных случаев и устроить на них серии тестов аналогичные второму пункту:

- Отсортированный массив
- Массив с одинаковыми элементами
- Массив с максимальным количеством сравнений при выборе среднего элемента в качестве опорного
- Массив с максимальным количеством сравнений при детерминированном выборе опорного элемента

При работе сортировки подсчитать количество вызовов рекурсивной функции, и высоту рекурсивного стека. Построить график худшего, лучшего, и среднего случая для каждой серии тестов.

Для каждой серии тестов построить график худшего случая.

Подобрать такую константу c, чтобы график функции c*n*log(n) находился близко к графику худшего случая, если возможно построить такой график.

Проанализировать полученные графики и определить есть ли на них следы деградации метода относительно своей средней сложности.

Описание метода/модели.

Быстрая сортировка является одним из самых быстрых алгоритмов сортировки массивов.

Общая идея алгоритма состоит в следующем:

- Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.
- Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
- Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Выполнение задачи.

Для реализации данного метода сортировки использовался язык программирования C++.

Алгоритм самой сортировки был описан выше. С помощью этой сортировки было выполнено по 20 тестов на массивы с различной длиной. Было зафиксировано время, затраченное на сортировку и количество вызовов рекурсий. То есть перед началом сортировки массива (в нашем случае это был вектор) запускался таймер и останавливался, когда сортировка заканчивалась. Помимо этого, был реализован счетчик вызовов рекурсий, что дало высоту стека рекурсий. Чтобы сделать тесты на массивах с разной длиной, мы создаем массив N с размерностям.

Проведя все необходимые тесты, получился график со всеми значениями времени для каждой N и график зависимости количества рекурсий от количества N.

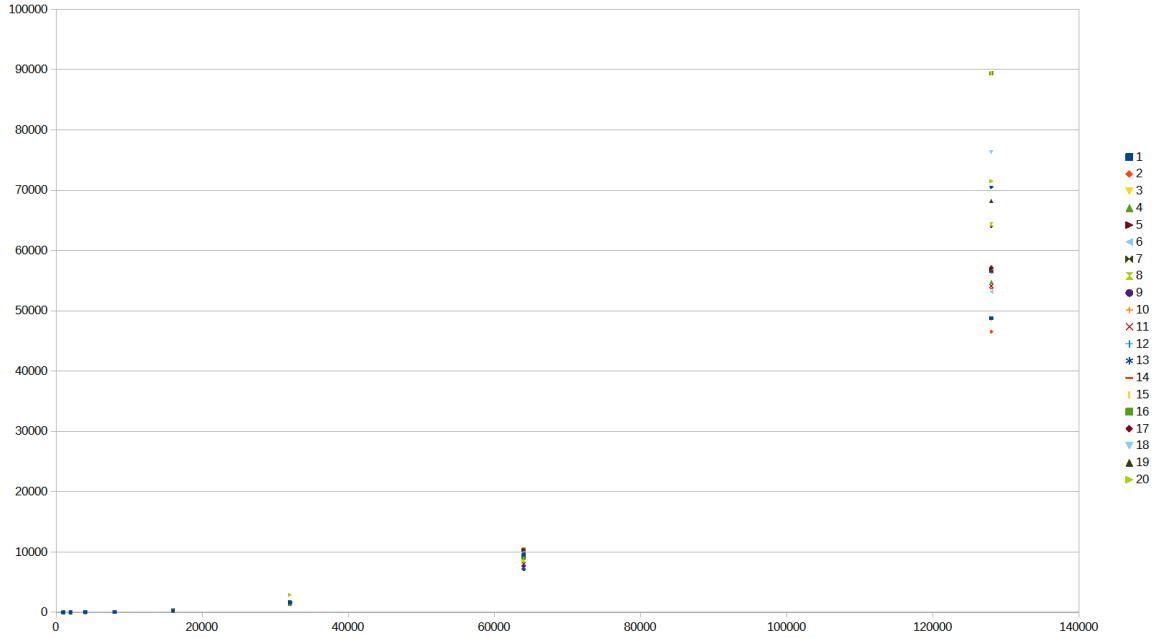


График зависимости времени от N для обычного массива

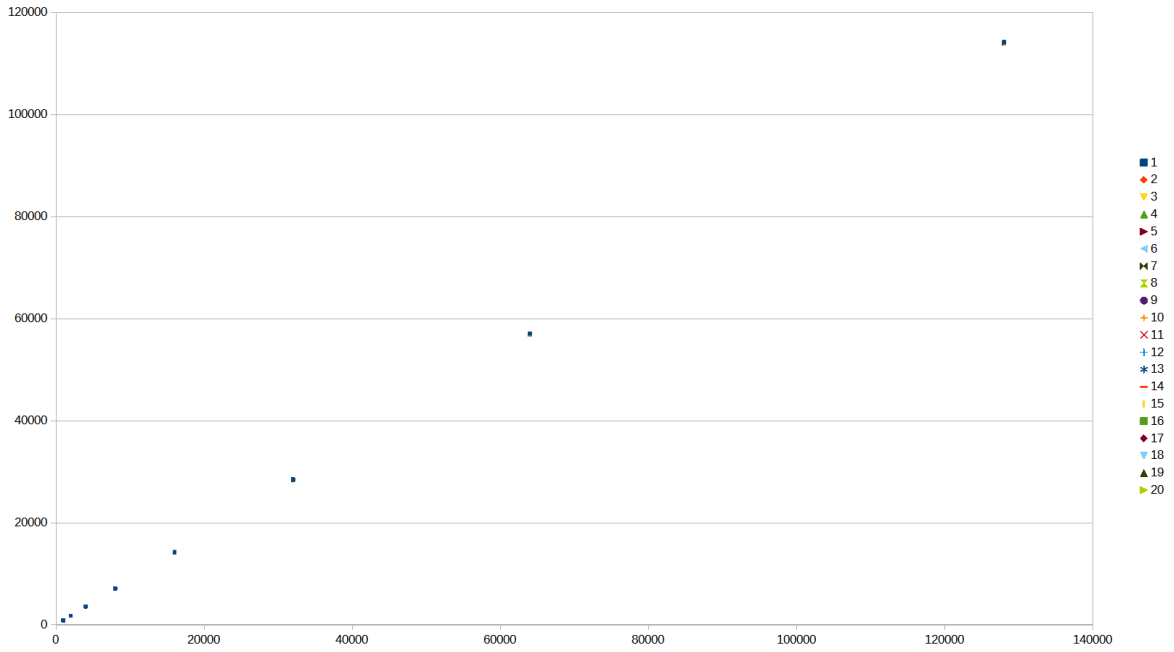
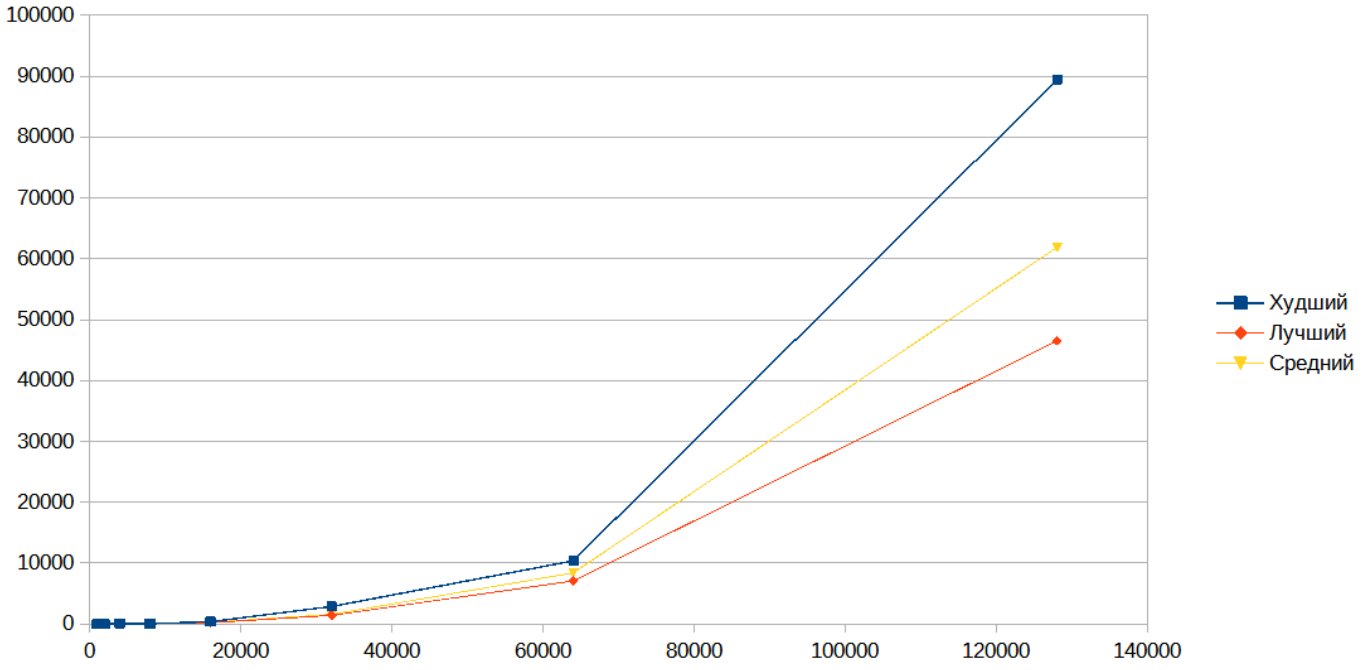
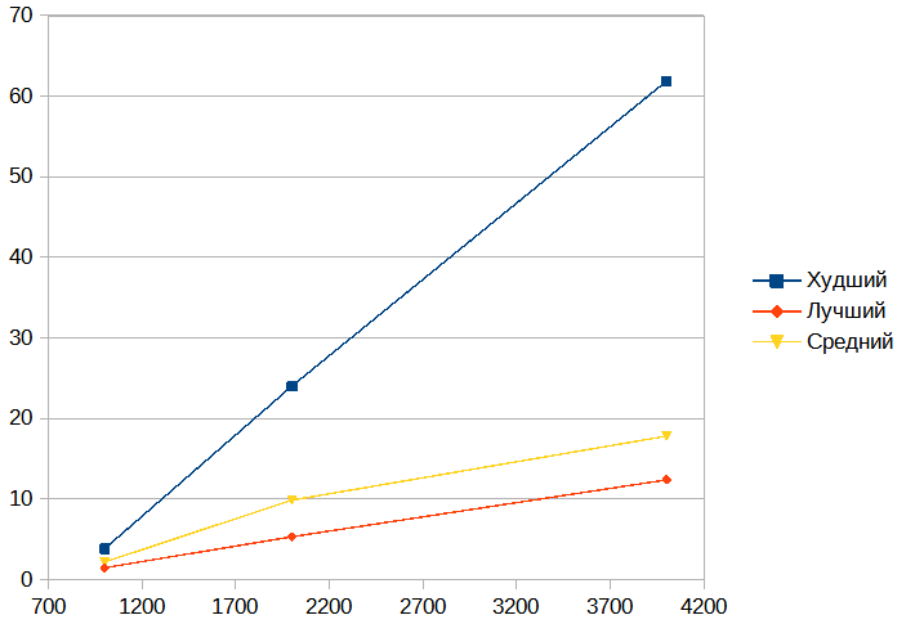


График зависимости количества рекурсий от N для обычного массива

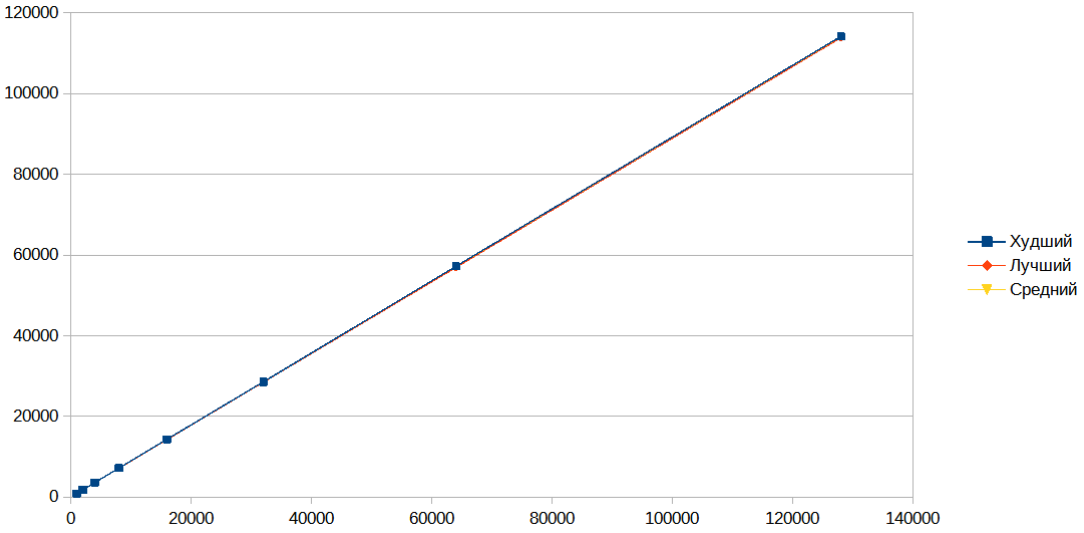
Для того, чтобы получить лучший и худший случаи, необходимо соединить точки с наименьшим и наибольшим временем/количеством рекурсий соответственно. Также, посчитав среднее значение времени для каждого N, можно получить средний случай.



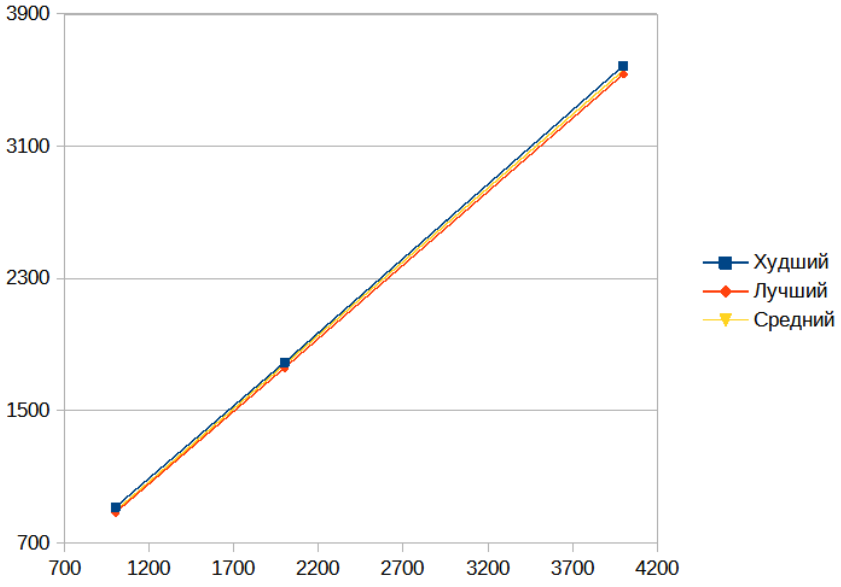
Лучший, средний и худший случаи для графика времени



Лучший, средний и худший случаи для графика времени (первые 3 точки)



Лучший, средний и худший случаи для графика рекурсий

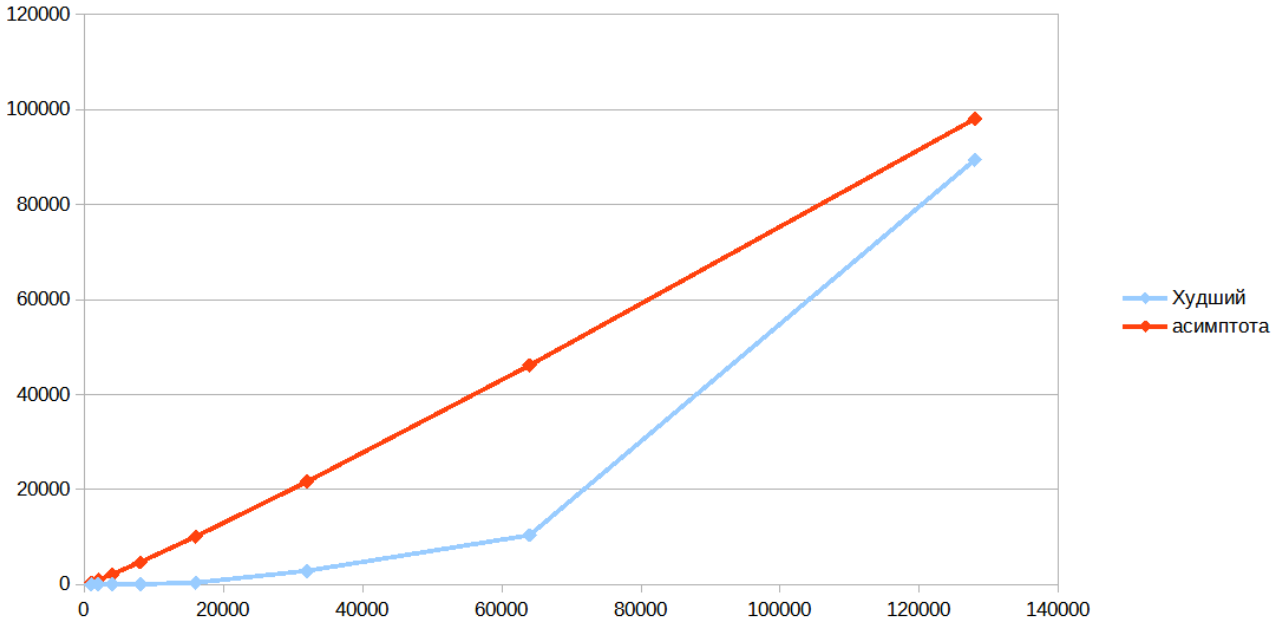


Лучший, средний и худший случаи для графика рекурсий (первые 3 точки)

Таким образом, мы получили график времени, который показывает за какое минимальное, максимальное и среднее время будет отработан алгоритм. А также график количества рекурсий, показывающий за какое минимальное, максимальное и среднее количества вызовов функции будет отсортирован массив.

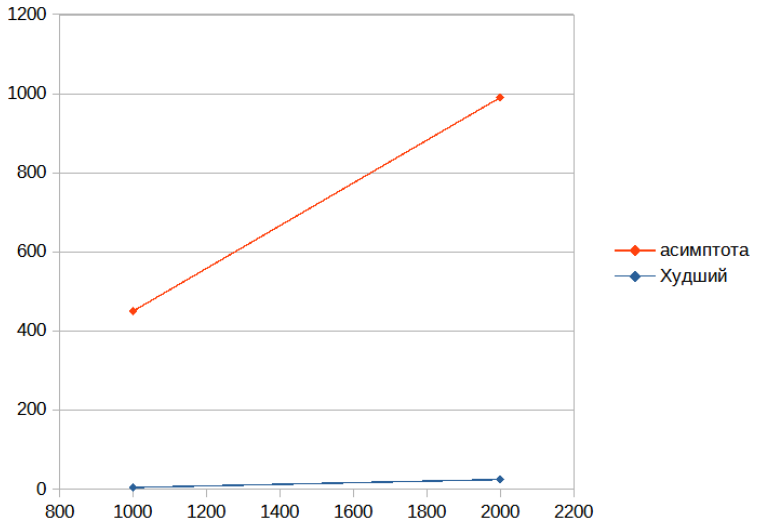
Также были построены график худшего случая, и график O(c * g(N*LOG(N))), где g(N*LOG(N)) соответствует асимптотической сложности рассматриваемого метода сортировки. Значение C было подобрано так, что начиная с N ~ 1000 график асимптотической сложности возрастал быстрее, чем полученное худшее время, но при этом был различим на графике.

Таким образом, мы получили асимптотическую функцию 0,15 * N*LOG(N)

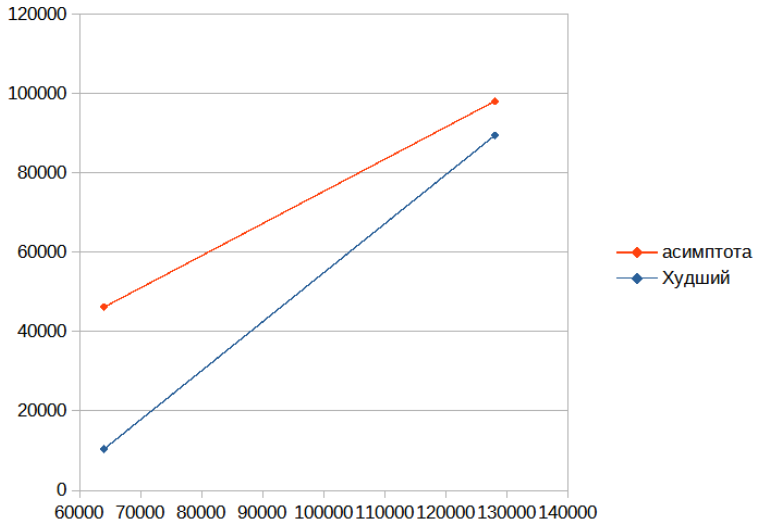


Худший случай и асимптотическая функция для графика времени

Первые 2 элемента:

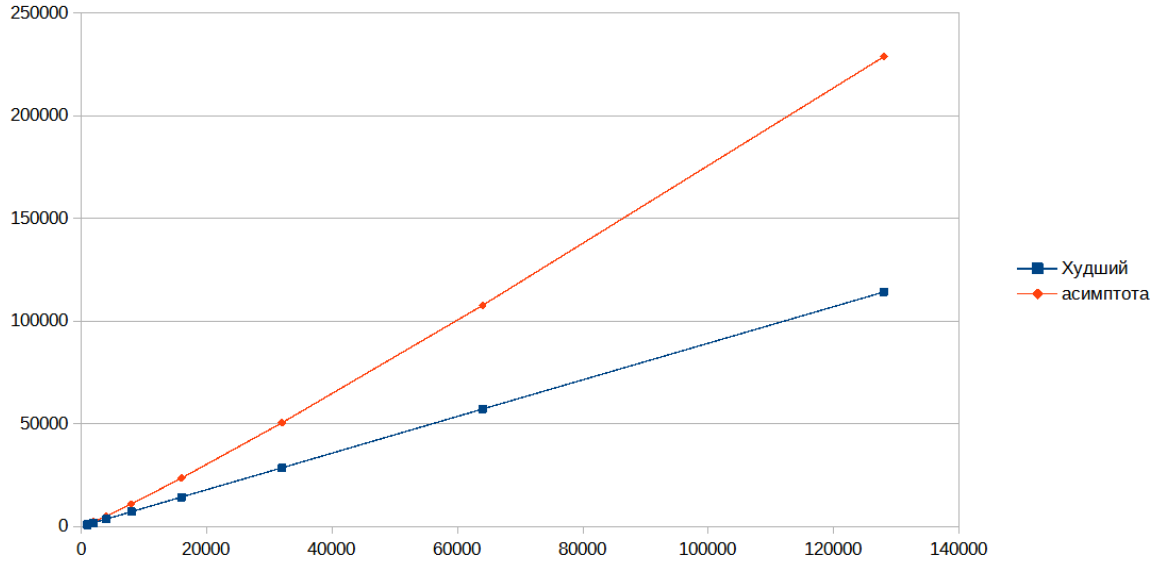


Последние 2 элемента:



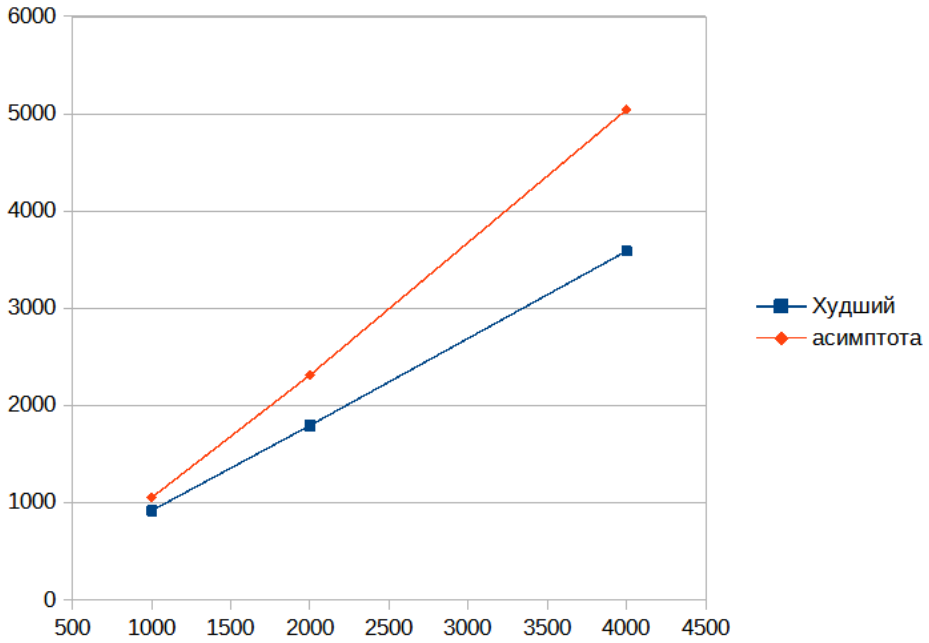
Аналогично были построены графики для зависимости количества рекурсий от N.

Асимптотическая функция 0,35 * N*LOG(N)



Худший случай и асимптотическая функция для графика рекурсий

Первые 3 элемента:


6000
5000
4000
3000
2000
1000
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Худший
асимптота

Отсортированный массив

Также была выполнена сортировка уже отсортированного массива. Полученные результаты показали, что на такую сортировку ушло столько же вызовов рекурсий, сколько и на обычную сортировку этого массива. Однако время, затраченное на нее все же различалось.

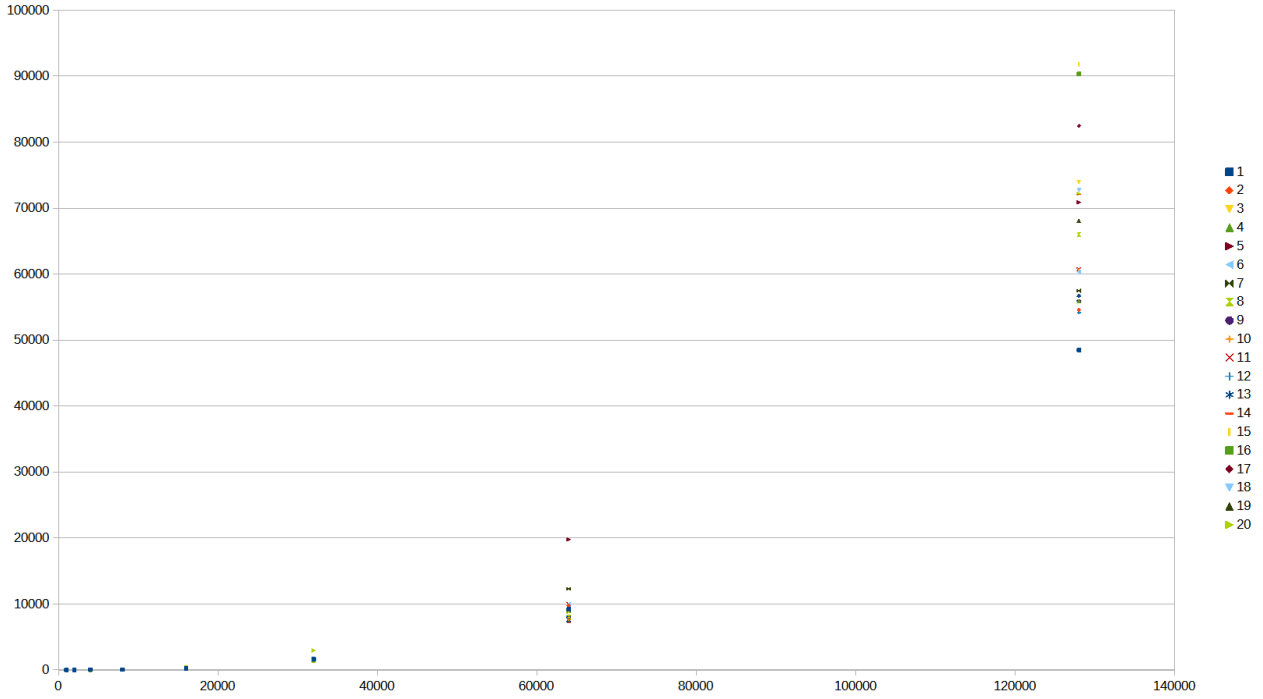
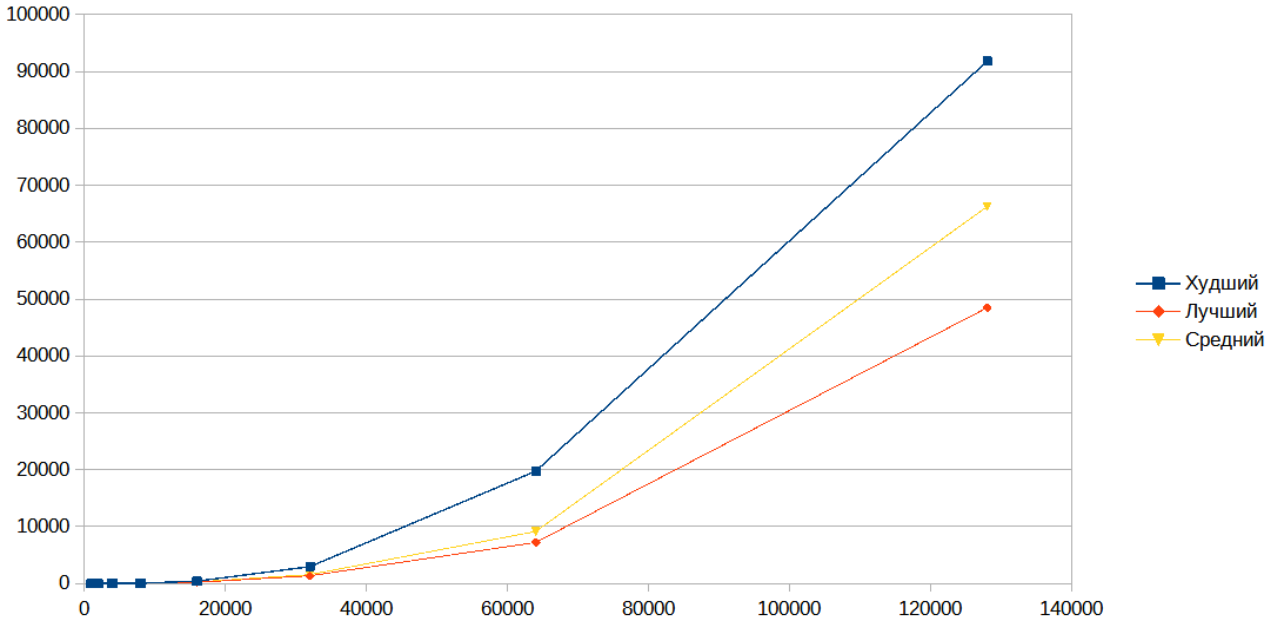
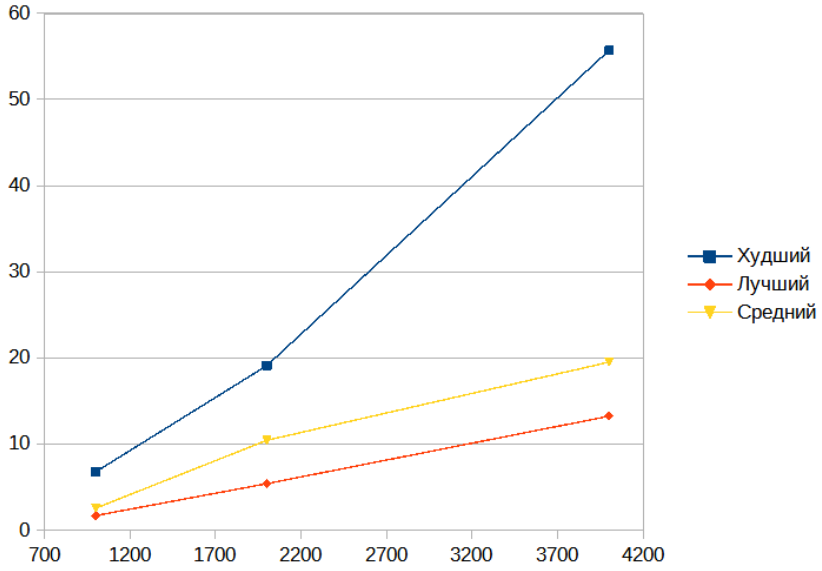


График зависимости времени от N для заранее отсортированного массива

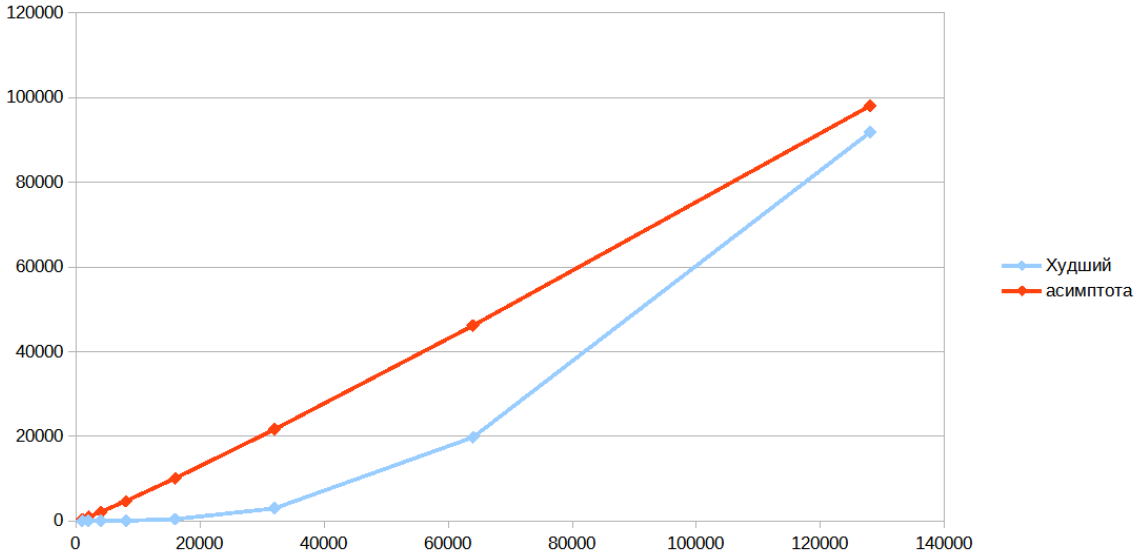


Лучший, средний и худший случаи для графика времени



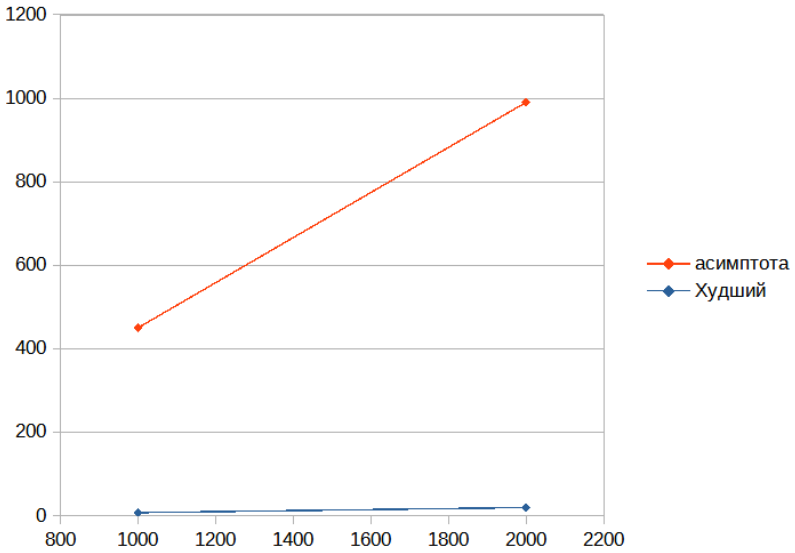
Лучший, средний и худший случаи для графика времени (первые 3 точки)

Аналогично была построена асимптотическая функция. Она совпадала с асимптотической функцией для обычного массива.

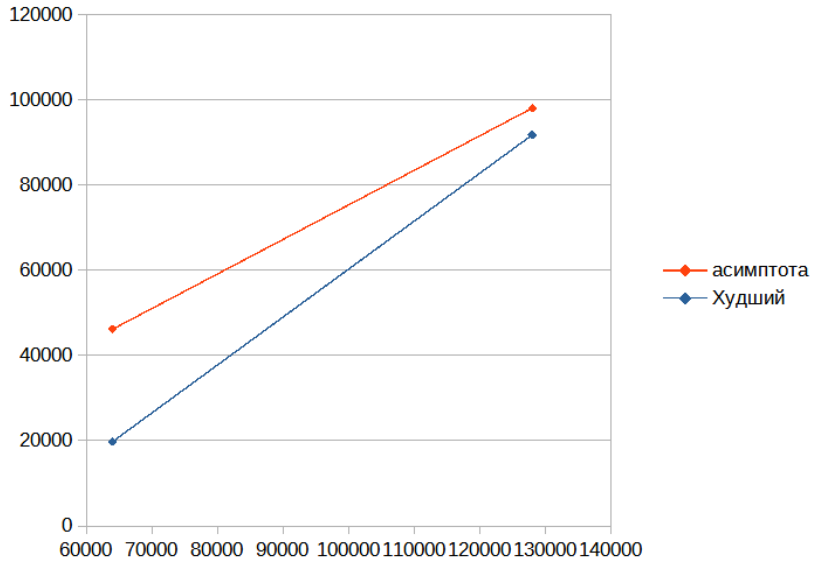


Худший случай и асимптотическая функция для графика времени

Первые 2 элемента:



Последние 2 элемента:



Массив из одинаковый элементов (нулевой массив)

Результаты сортировки массива из одинаковых элементов показали, что на нее уходит намного меньше рекурсий и времени, чем на обычный массив. А также при одинаковом количестве элементов получалось одинаковое количество рекурсий и времени. Поэтому на графиках показано только по одному тесту на каждое N.

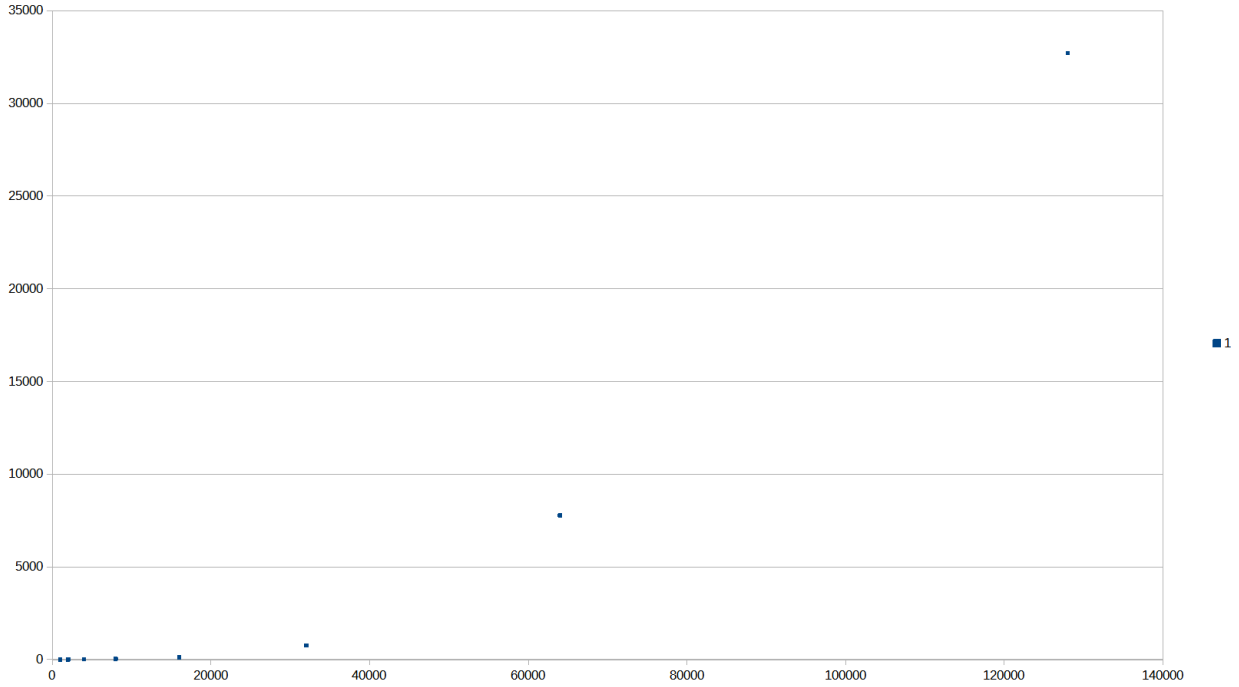
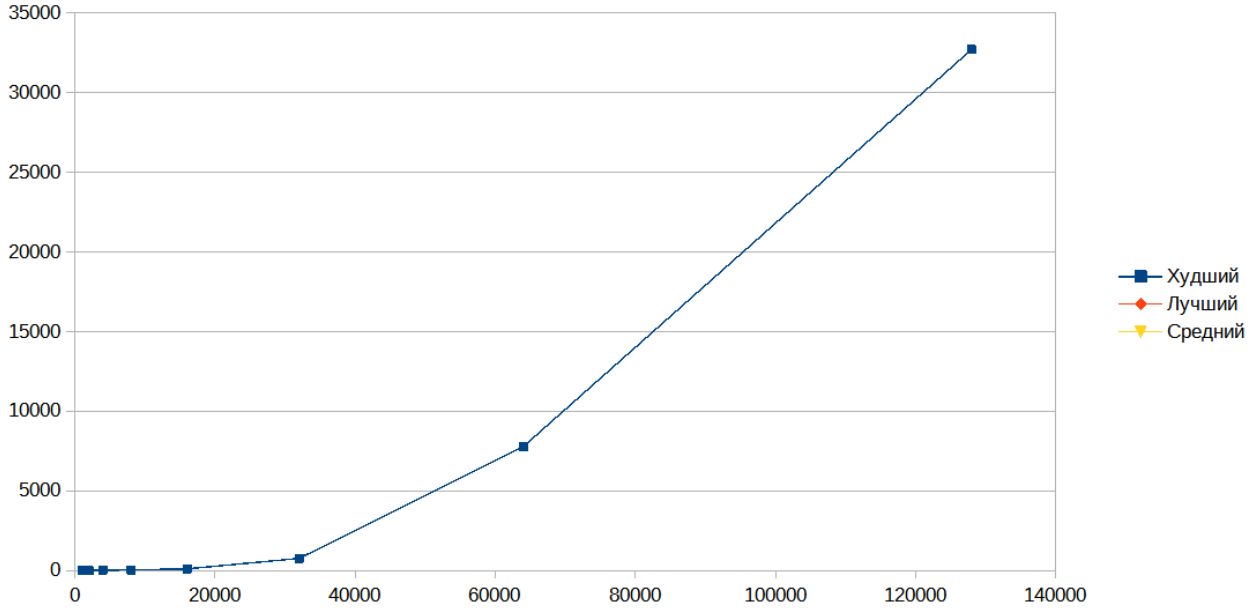
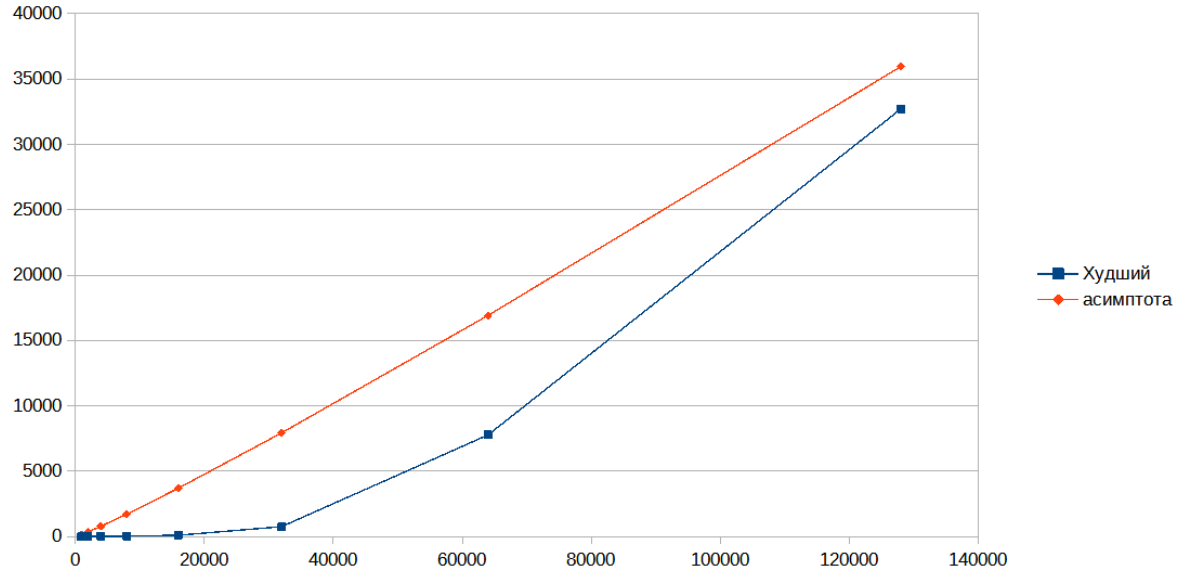


График зависимости времени от N для массива из одинаковых элементов



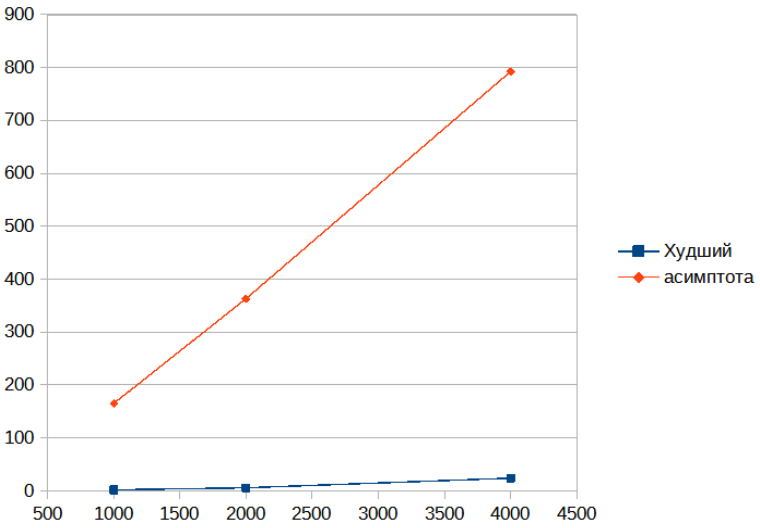
Лучший, средний и худший случаи для графика времени

Также была построена та же асимптотическая функция.



Худший случай и асимптотическая функция для графика времени

Первые 3 элемента:



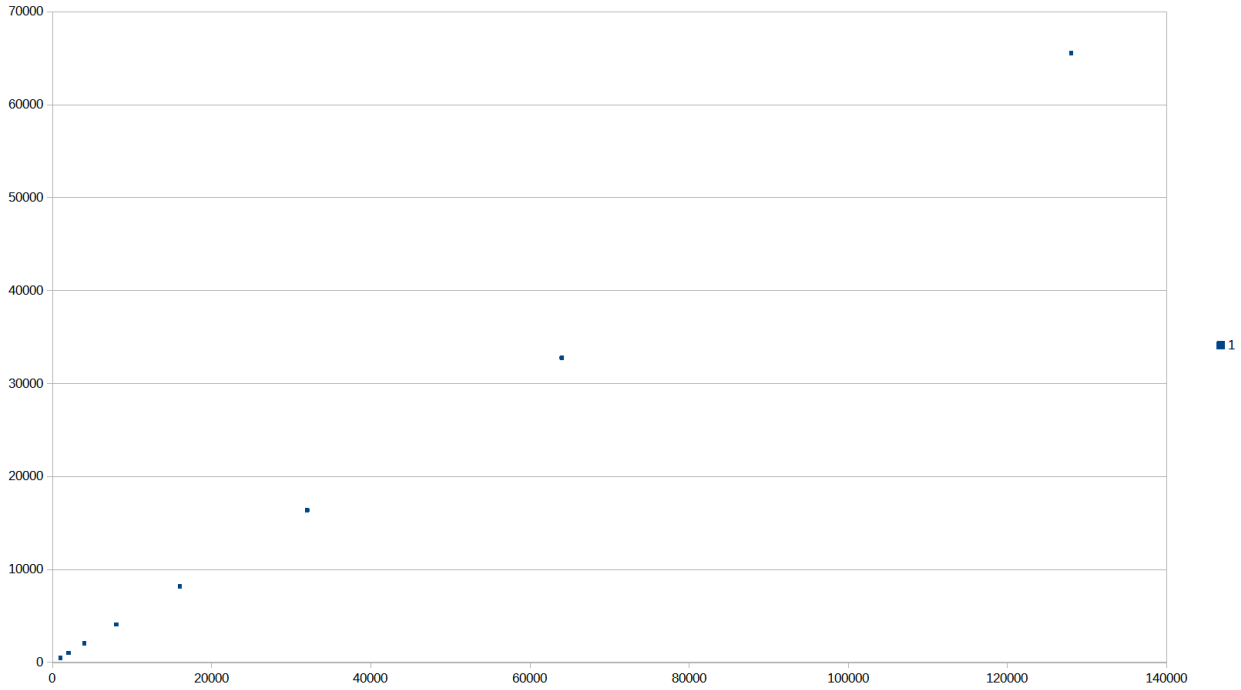
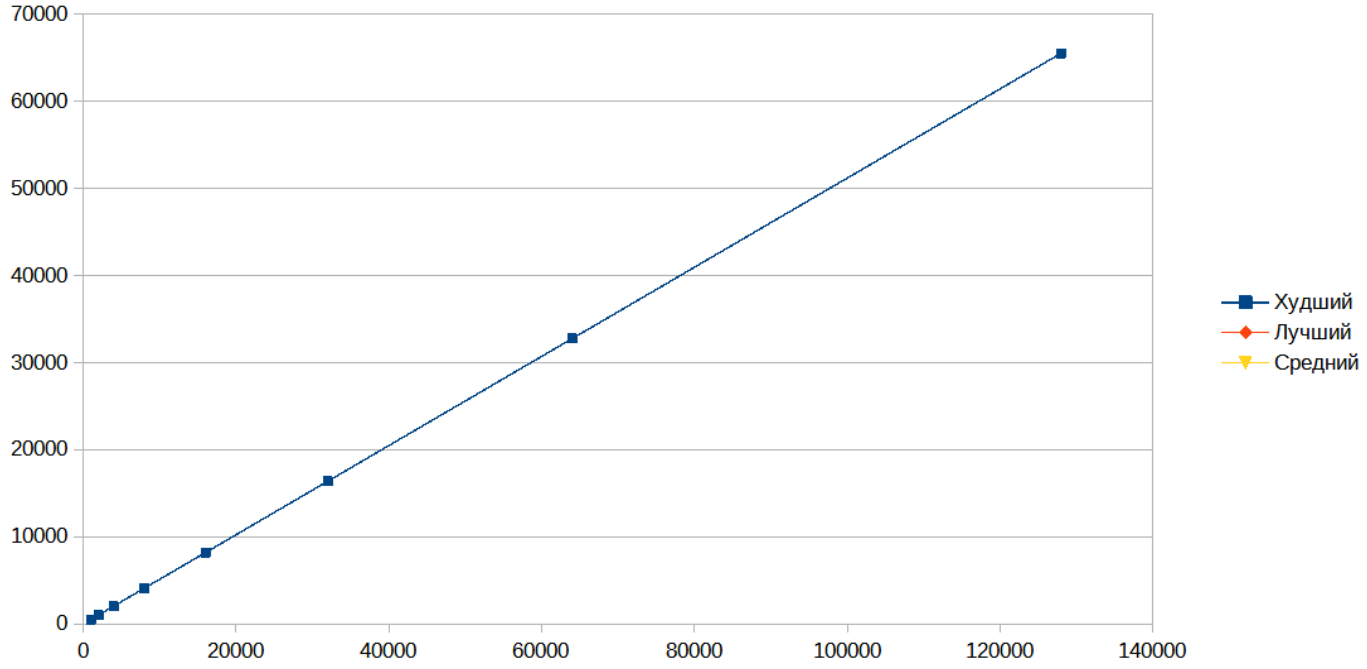
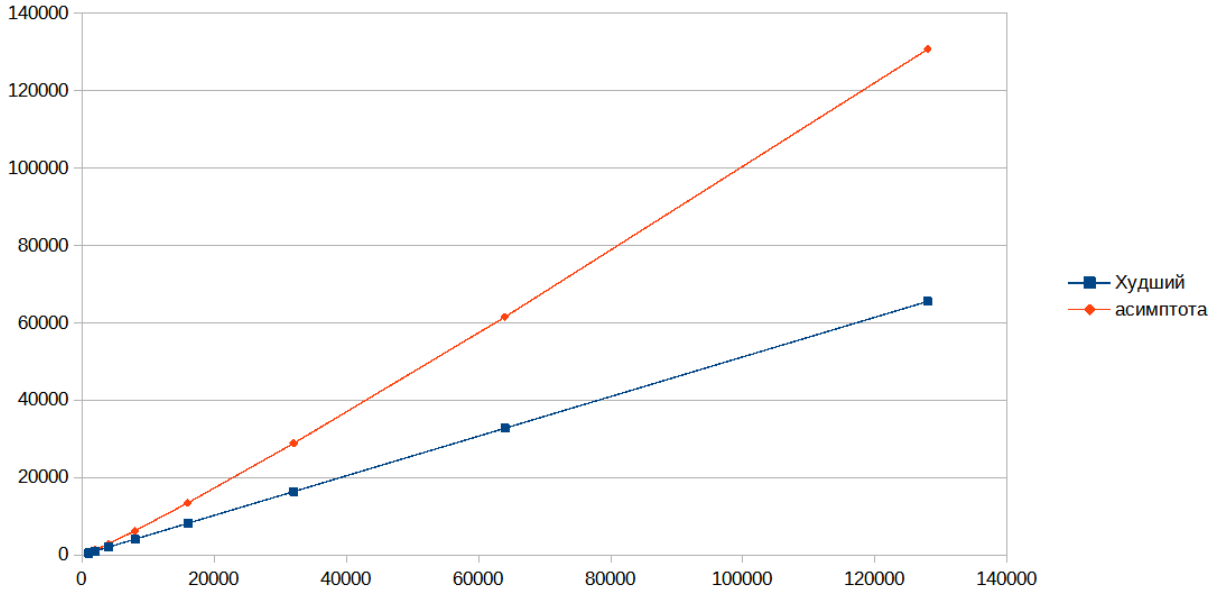


График зависимости рекурсий от N для массива с одинаковыми элементами

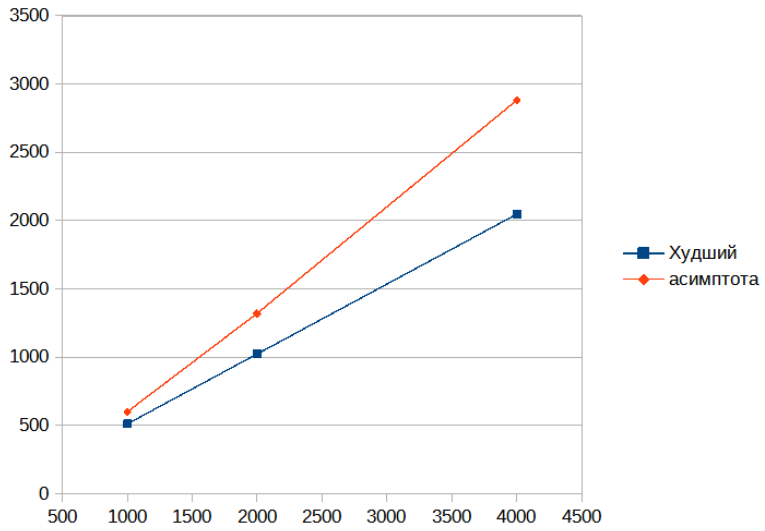


Лучший, средний и худший случаи для графика рекурсий



Худший случай и асимптотическая функция для графика рекурсий

Первые 3 элемента:



Массив с максимальным количеством сравнений при выборе среднего элемента в качестве опорного

На основе статьи отсортированный массив был преобразован по описанному в ней алгоритму.

```
void antiQsort(a: T[n])
    for i = 0 to n - 1
        swap(a[i], a[i / 2])
```

Этот алгоритм позволяет сделать так, чтобы при выполнении быстрой сортировки опорный элемент выбирался так, что массив делился на часть из одного элемента и N-1 элемент. Тем самым сортировка должна была занимать максимальное количество сравнений.

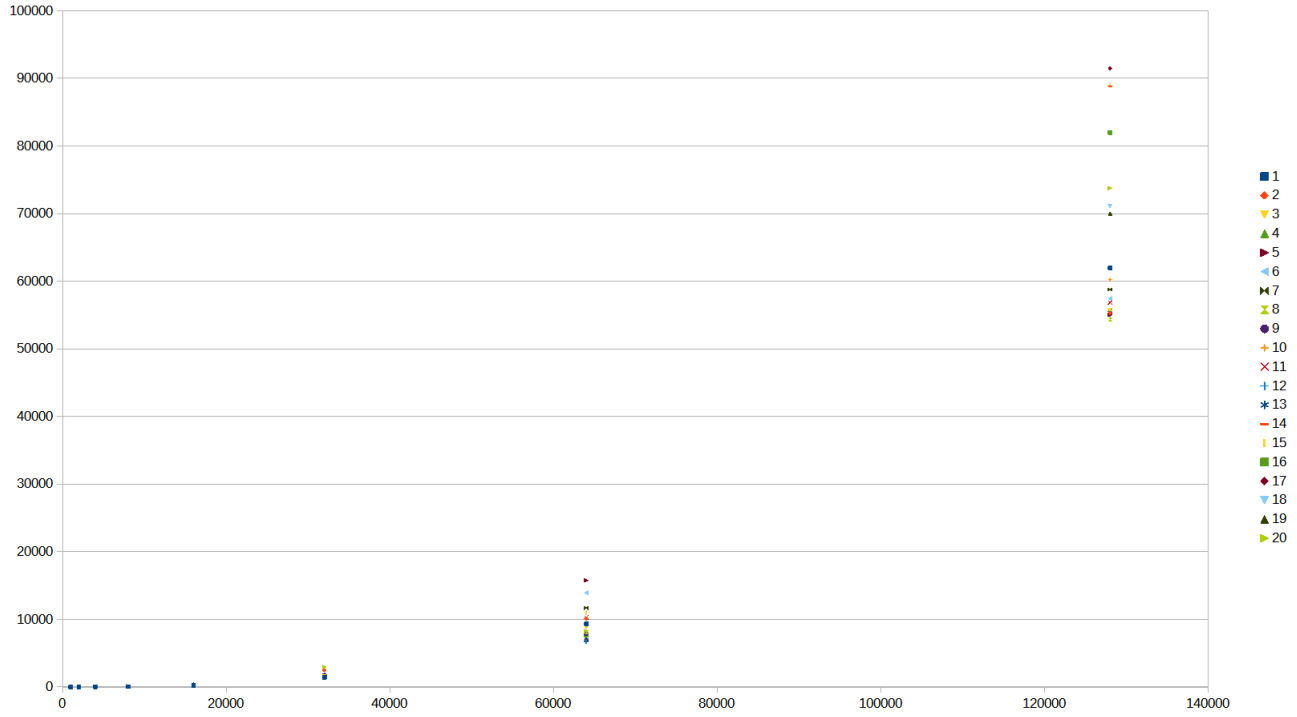
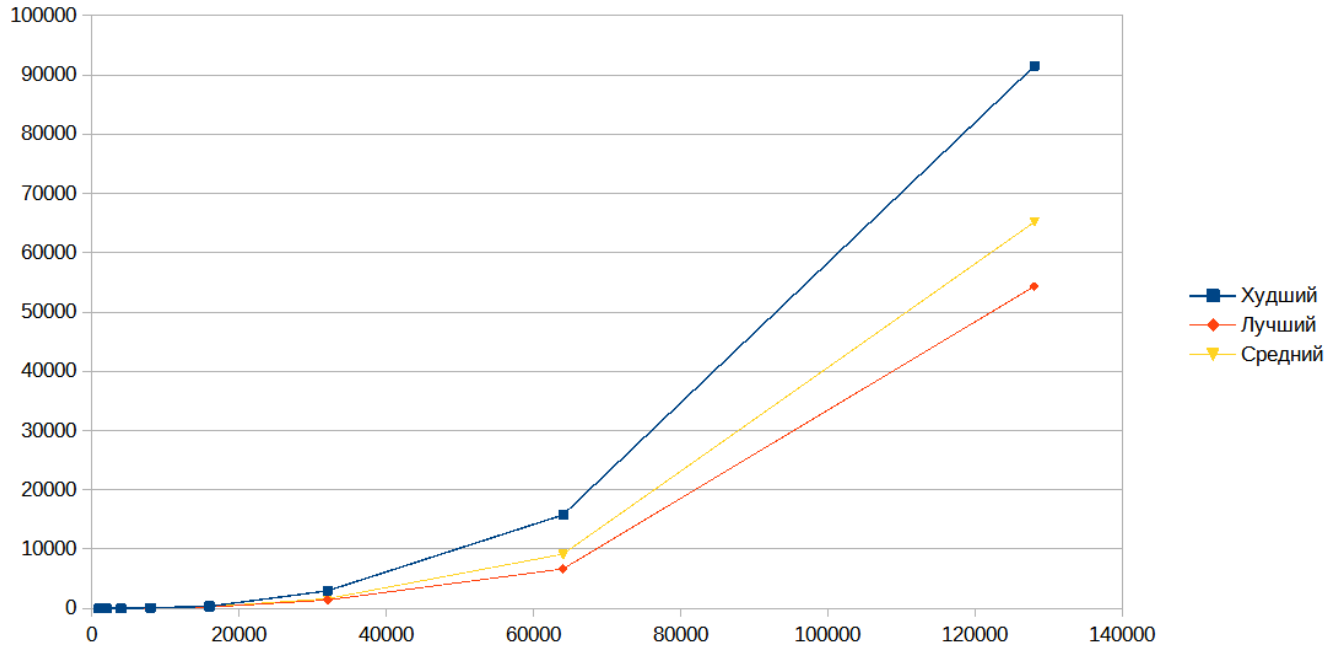
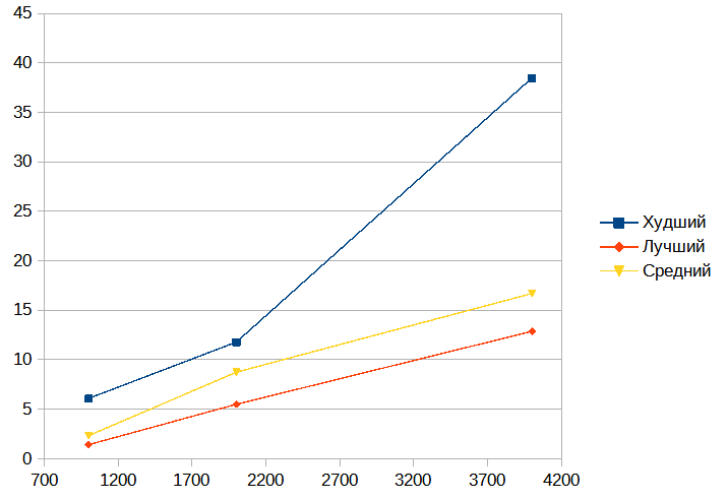


График зависимости рекурсий от N для массива с максимальным количеством сравнений

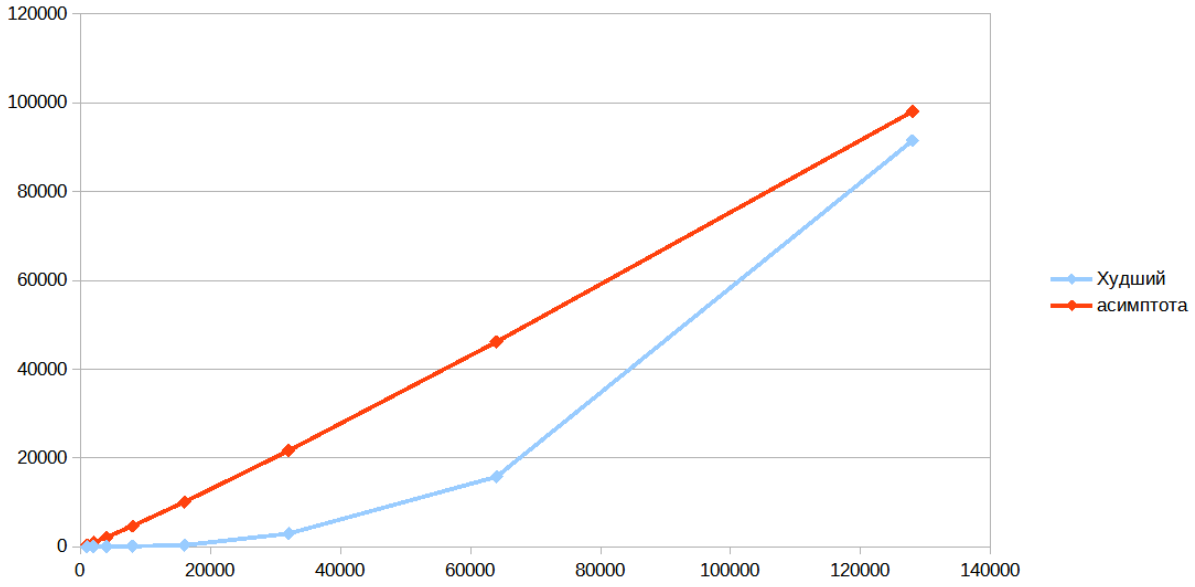


Лучший, средний и худший случаи для графика времени

Первые 3 точки:

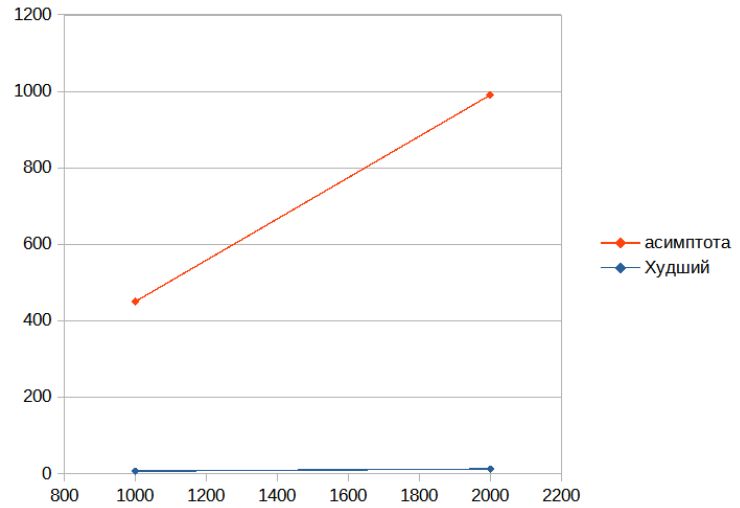


Была аналогично построена асимптотическая функция.



Худший случай и асимптотическая функция для графика времени

Первые 2 элемента:



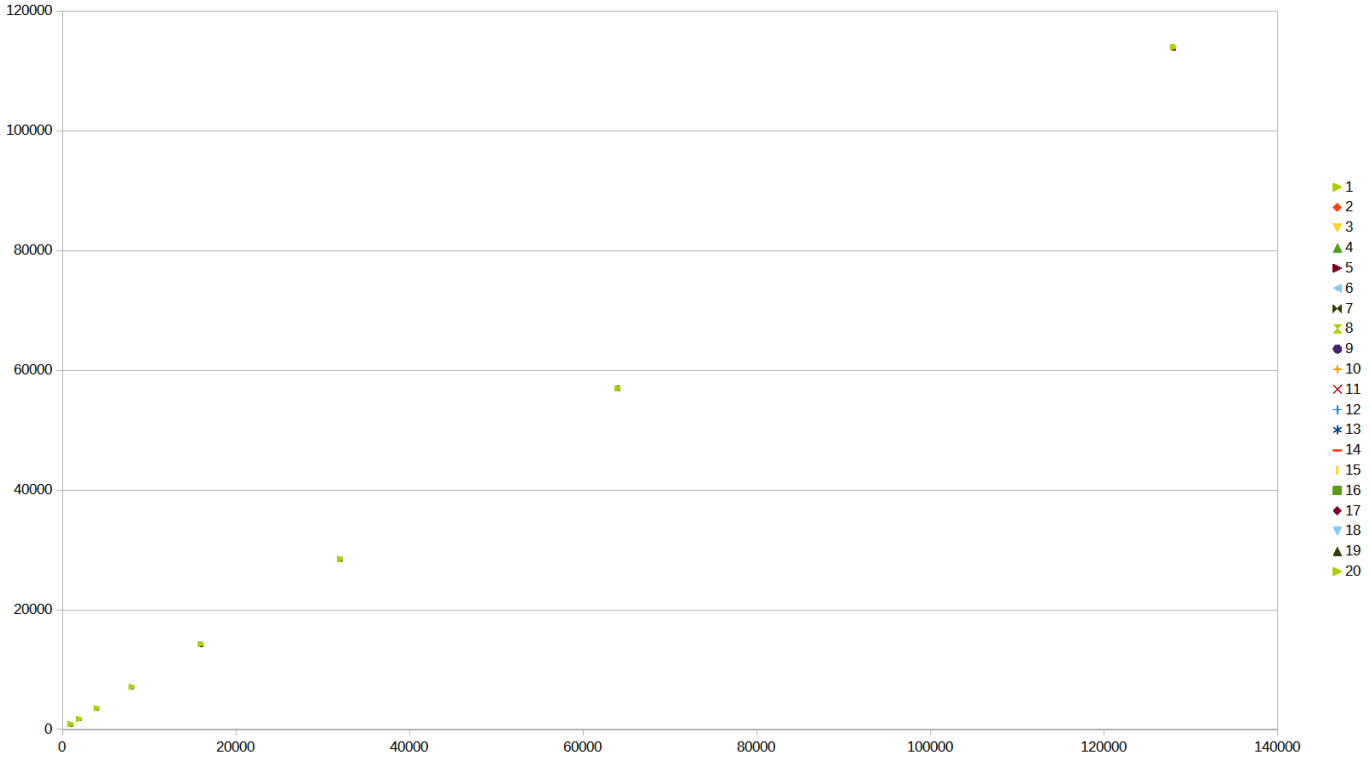
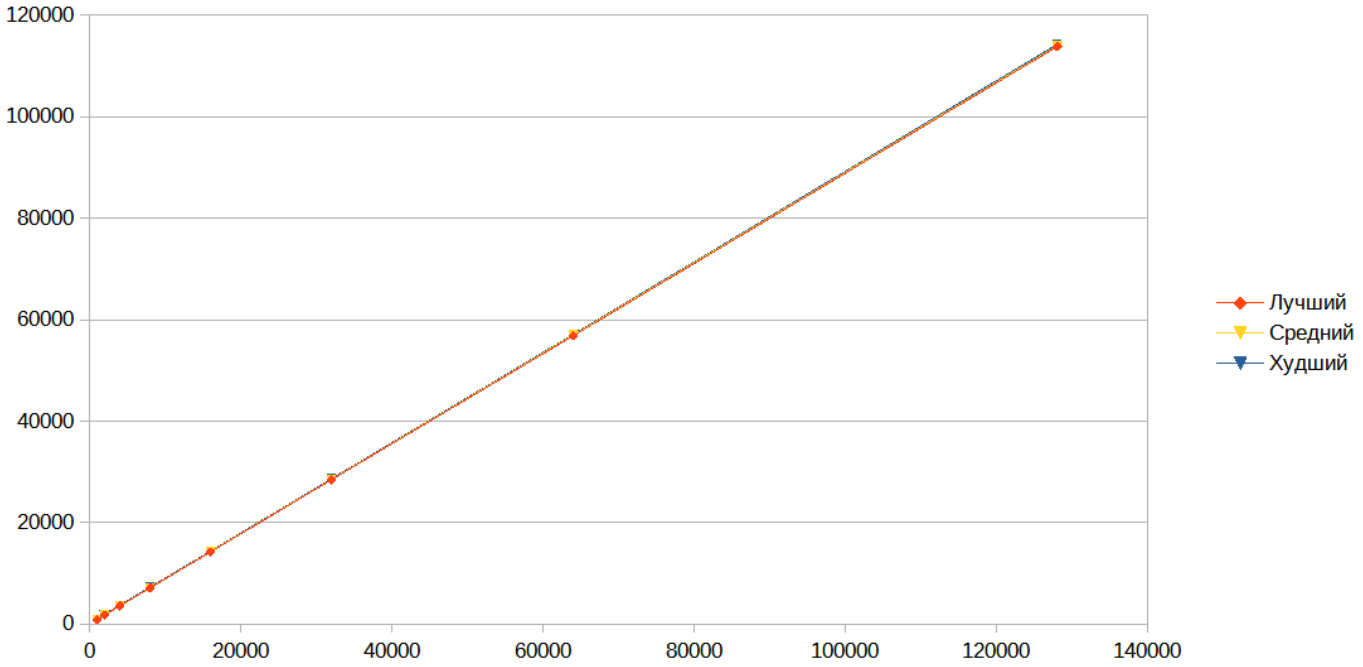
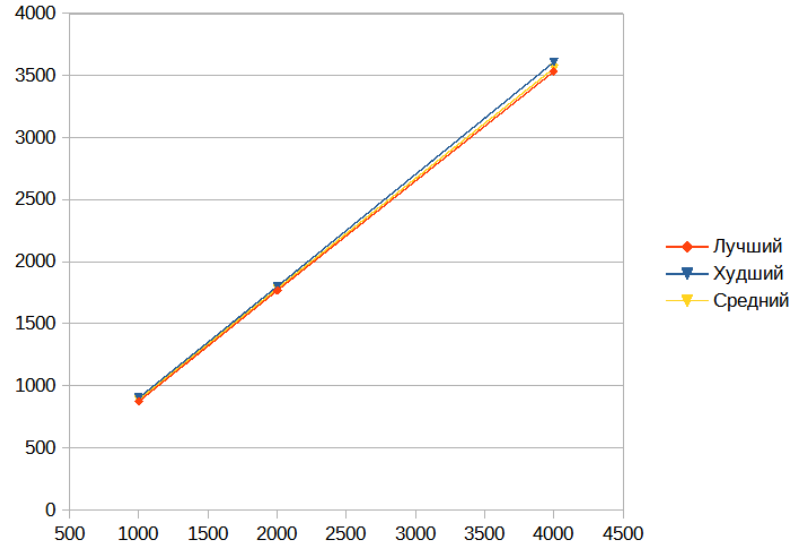


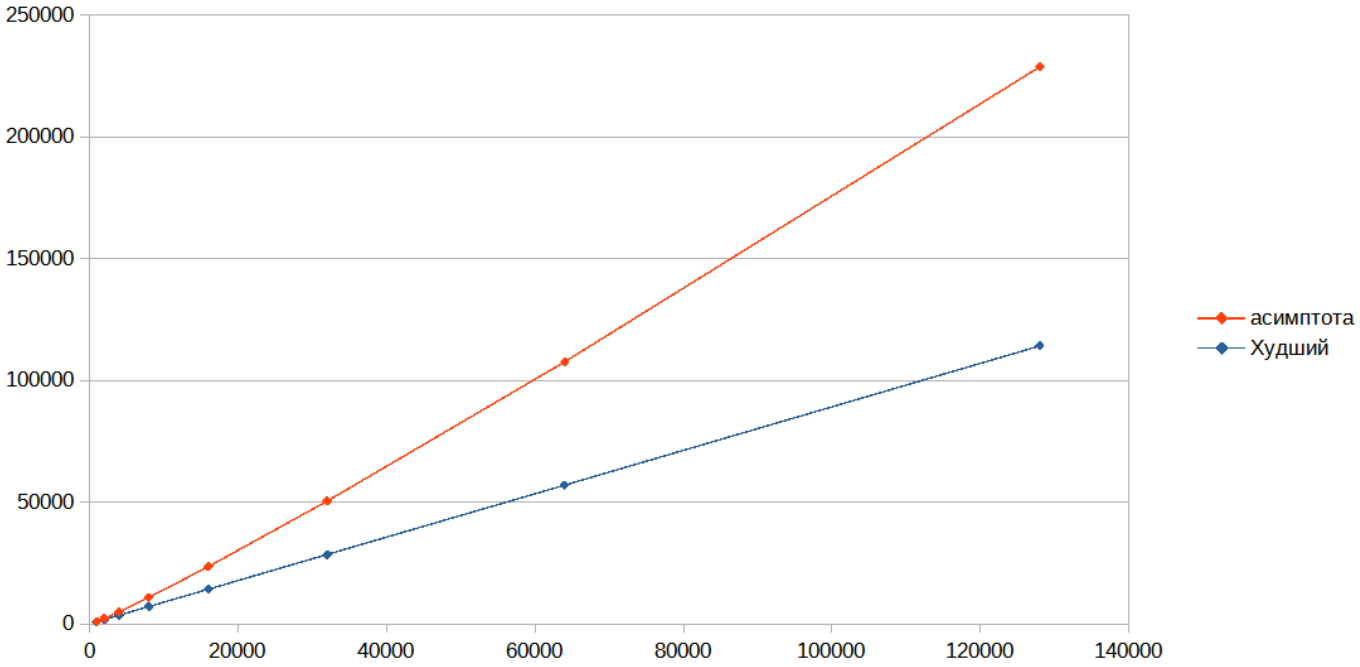
График зависимости рекурсий от N для массива с максимальным количеством сравнений



Лучший, средний и худший случаи для графика рекурсий

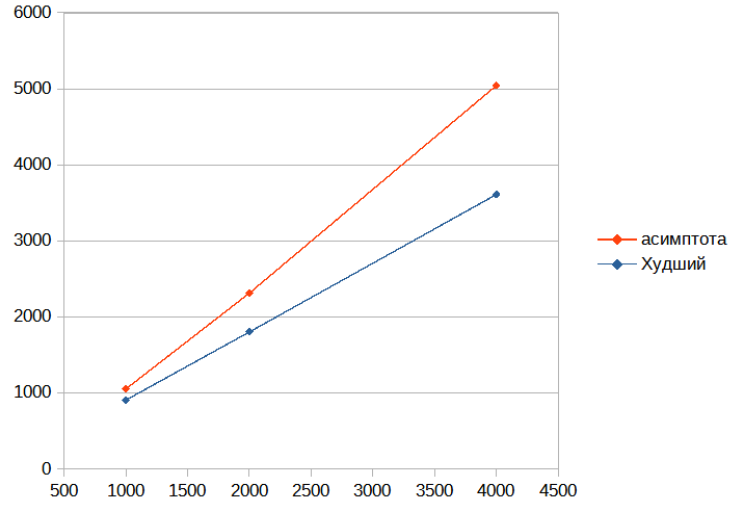
Первые 3 точки:





Худший случай и асимптотическая функция для графика рекурсий

Первые 3 точки:



Массив с максимальным количеством сравнений при детерминированном выборе опорного элемента

Данный алгоритм также преобразовывает массив так, чтобы сортировка занимала максимальное количество сравнений. Однако здесь опорный элемент выбирается детерминировано.

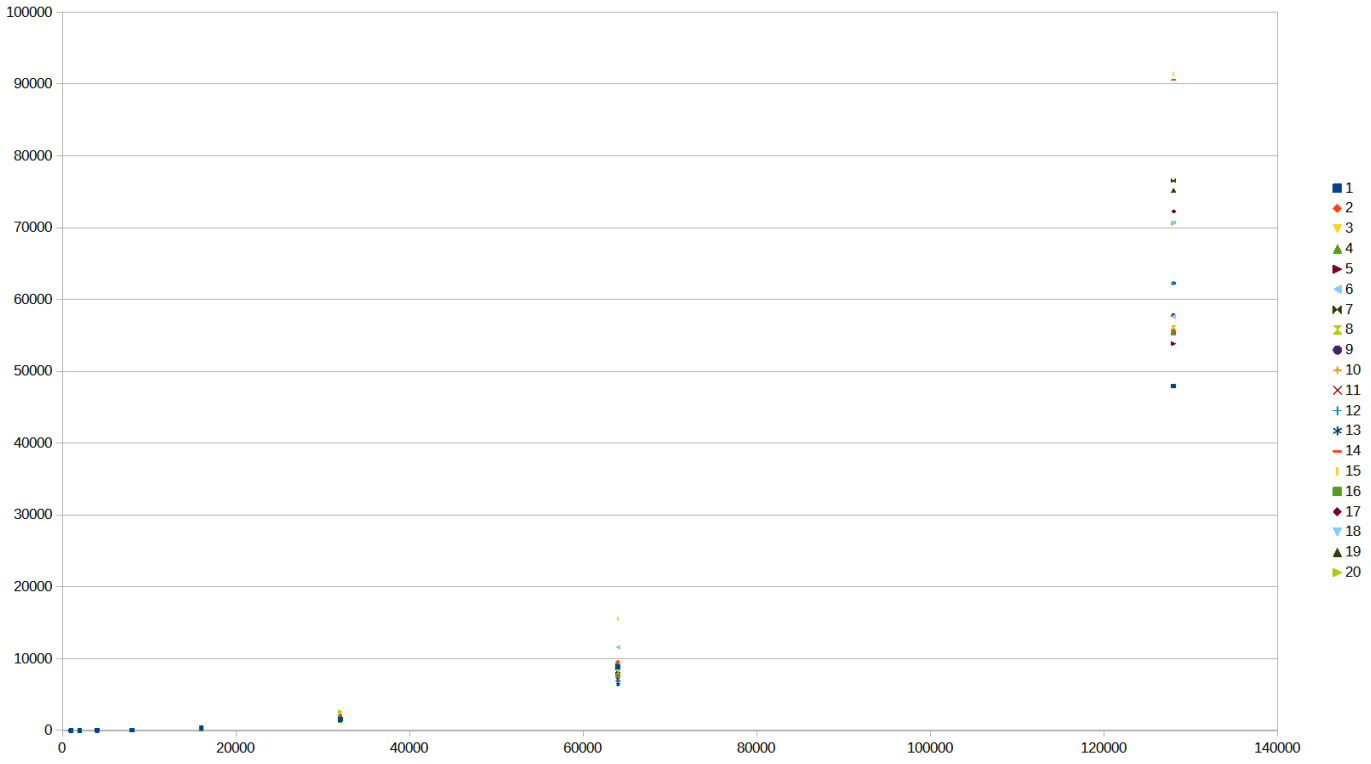
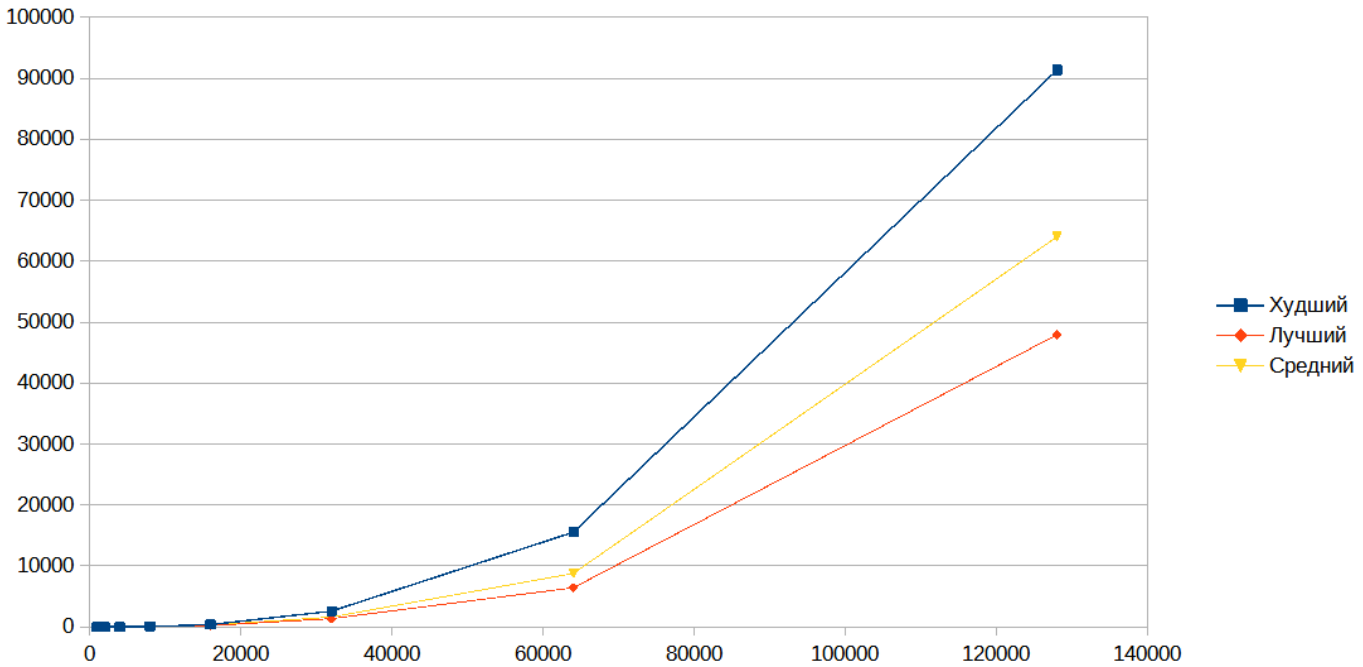
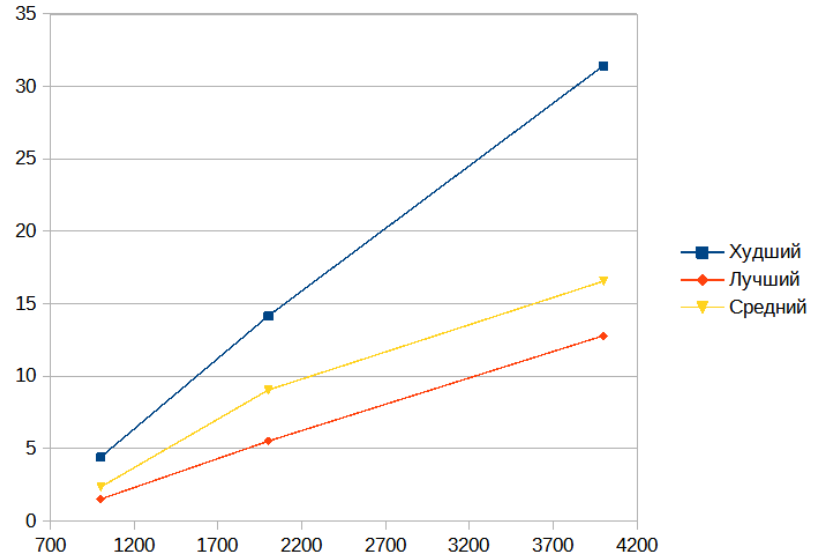


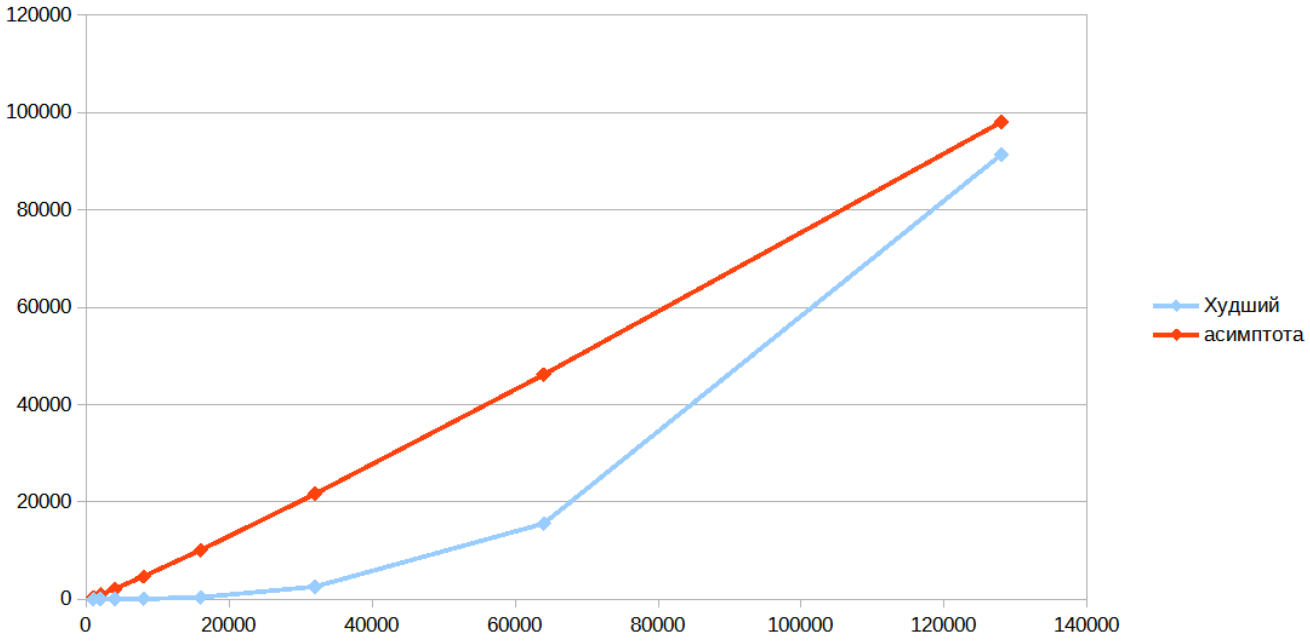
График зависимости рекурсий от N для массива с максимальным количеством сравнений



Лучший, средний и худший случаи для графика времени

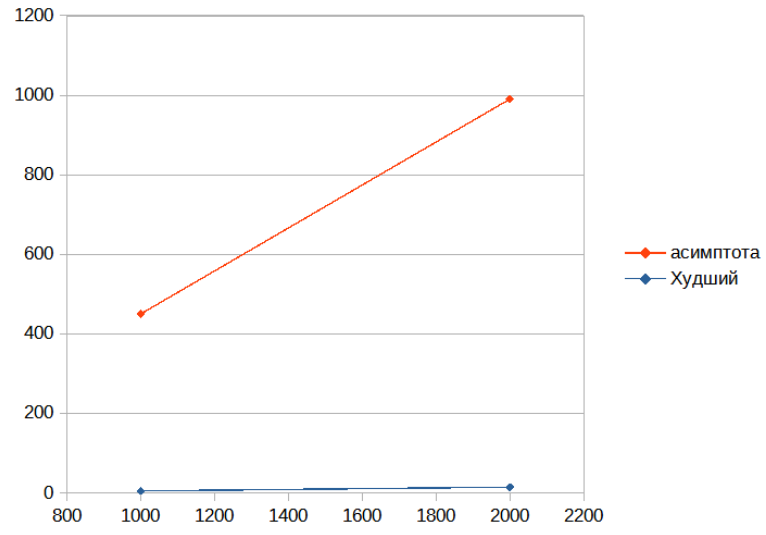
Первые 3 точки:





Худший случай и асимптотическая функция для графика времени

Первые 2 точки:



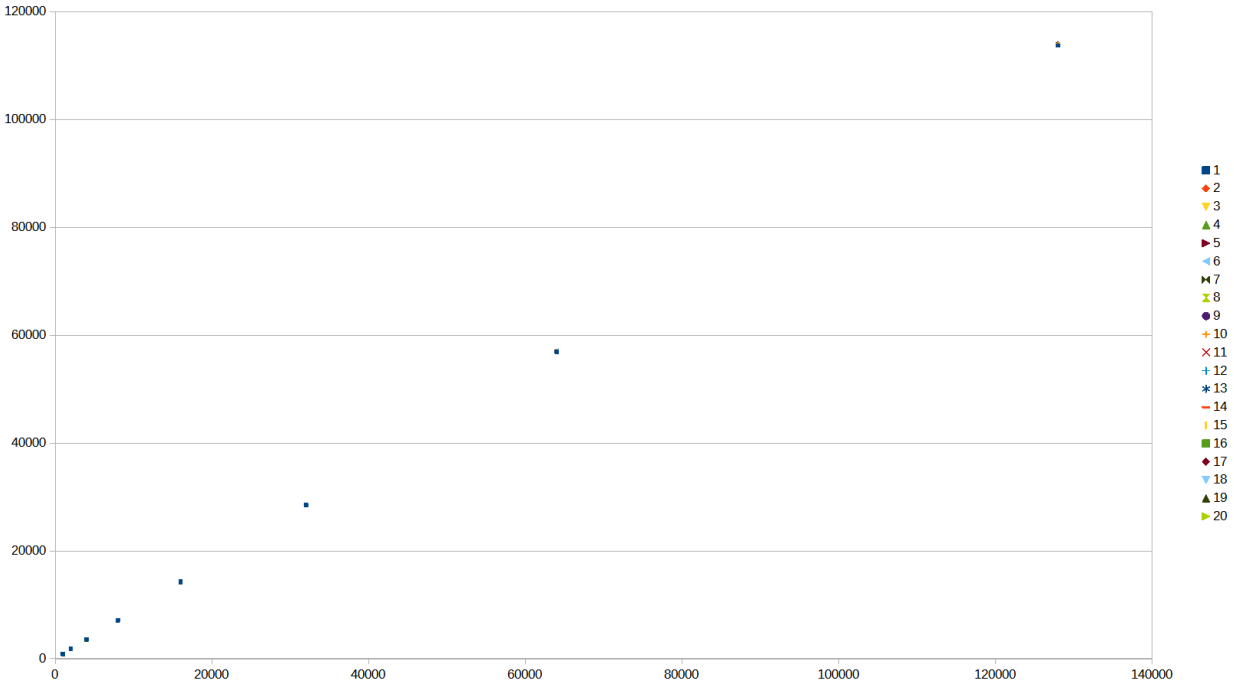
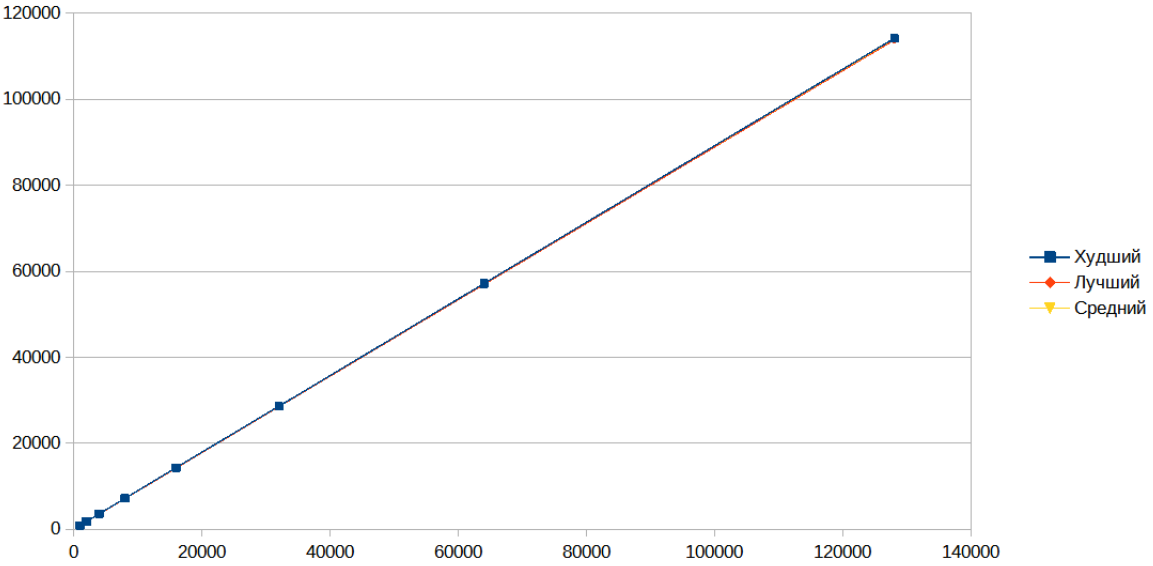
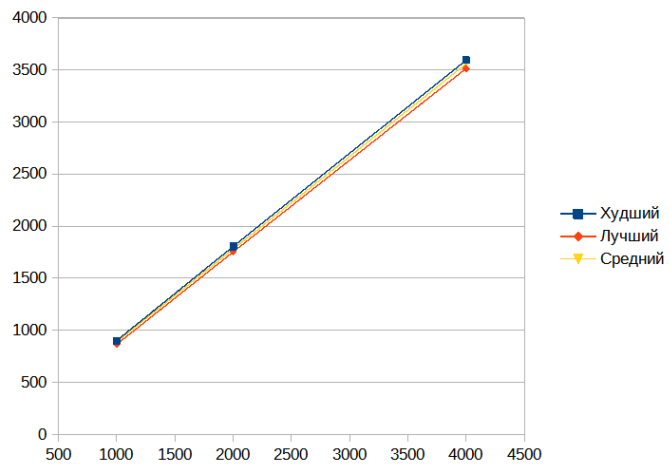


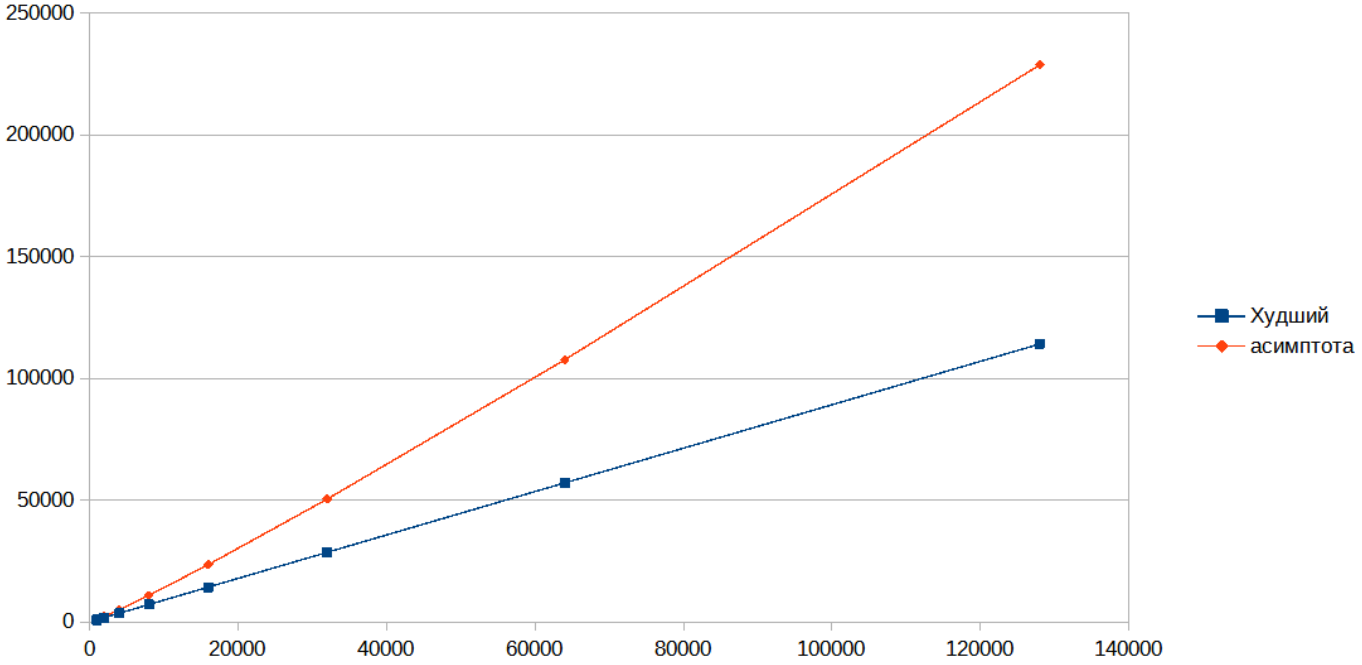
График зависимости рекурсий от N для массива с максимальным количеством сравнений



Лучший, средний и худший случаи для графика рекурсий

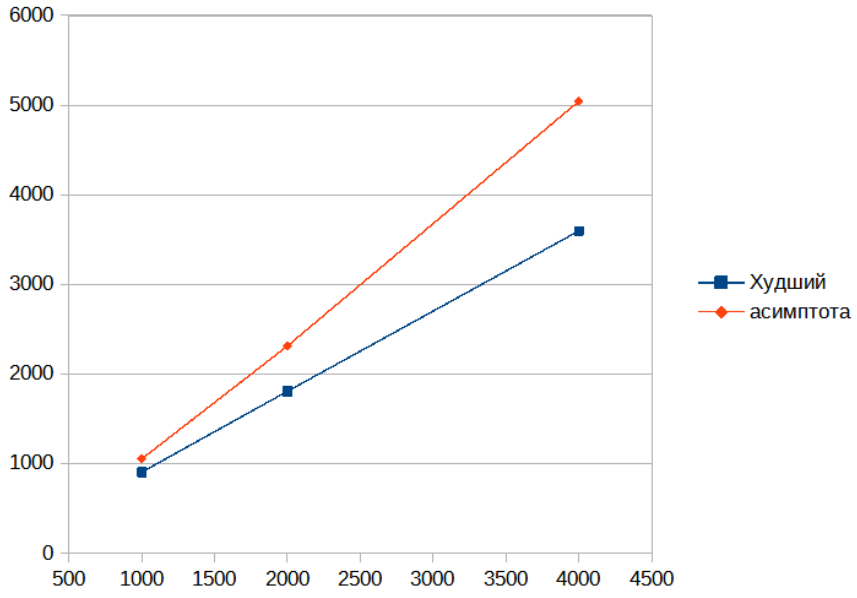
Первые 3 точки:





Худший случай и асимптотическая функция для графика рекурсий

Первые 3 точки:



Далее представлен код на языке C++:

```
#include <iostream>
#include <random>
#include <chrono>
#include <fstream>
#include <vector>

const int M = 5; // Количество тестов
unsigned int count_of_recursion = 0; // Счетчик рекурсий

using namespace std;

/* Очистка файла перед записью */
void clearFile() {
    ofstream out;
    out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data_h.txt");
}

/* Быстрая сортировка
*
* @param pivot опорный элемент
* @param new_begin начало сортировки
* @param new_end конец сортировки
* @param count переменная для хранения
* @param v массив
*/
void quickSort(vector <double> v, int begin, int last) {
    double pivot, count;
    int new_begin = begin, new_last = last;
    count_of_recursion++;
    pivot = v[(new_begin + new_last) / 2]; //вычисление опорного элемента
    do
    {
        while (v[new_begin] < pivot) new_begin++;
        while (v[new_last] > pivot) new_last--;
        if (new_begin <= new_last) //перестановка элементов
```

```
        {
            count = v[new_begin];
            v[new_begin] = v[new_last];
            v[new_last] = count;
            new_begin++;
            new_last--;
        }
    } while (new_begin <= new_last);
    if (begin < new_last) quickSort(v, begin, new_last); // Вызов рекурсии
    if (new_begin < last) quickSort(v, new_begin, last); // Вызов рекурсии
}

/* Запись результата в файл */
void writer(int i, int N, double msec) {
    ofstream out;

    out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data_h.txt", ios::app);
    if (out.is_open())
    {
        out << i << " " << N << " " << msec << " " << count_of_recursion-1 << endl;
    }
}

/* Отделение части записи */
void piece() {
    ofstream out;
    out.open("C:\\Users\\evgen\\Desktop\\Алгоритмы\\data2.txt", ios::app);
    if (out.is_open())
    {
        out << "---------------------------------" << endl;
    }
}

/* Таймер
*
* @param i номер теста
```

```
    * @param v массив
    * @param N количество элементов в массиве
    * @param begin начало сортировки
    * @param last конец сортировки
    */
    void timer(vector <double> v, int N, int i) {
        int begin = 0, last = v.size() - 1;
        count_of_recursion = 0; // Обнуление счетчика
        chrono::high_resolution_clock::time_point start = chrono::high_resolution_clock::now(); // стартовое
время сортировки
        quickSort(v, begin, last); // сортировка
        chrono::high_resolution_clock::time_point end = chrono::high_resolution_clock::now(); // конечное
время сортировки
        chrono::duration<double, nano> nano_diff = end - start; // время в наносекундах
        chrono::duration<double, micro> micro_diff = end - start; // время в микросекундах
        chrono::duration<double, milli> milli_diff = end - start; // время в миллисекундах
        chrono::duration<double> sec_diff = end - start; // время в секундах
        writer((i + 1), v.size(), milli_diff.count()); // запись в файл
    }

    /* Отделение части записи */
    void sortingSort(vector <double> v, int N, int i) {
        ofstream out;
        timer(v, N, i);
    }

    /* Отделение части записи */
    void nullSort(vector <double> v, int N) {
        ofstream out;
        timer(v, N, 0);
    }

    /* Перестановка отсортированного массива */
    void antiQSort(vector <double> v, int N, int i) {
        for (int j = 2; j <= v.size(); j++) {
            int added_index = N - j;
```

```
        int middle = (added_index + (N - 1)) / 2; // корневой индекс для быстрой сортировки
        swap(v[added_index], v[middle]);
      }
      timer(v, v.size(), i);
    }

    /* Перестановка отсортированного массива для детермированного метода */
    void determQSort(vector <double> v, int N, int i) {
      double temp_array;
      int k = 0, n = N - 1;

      for (int j = 0; j < N; j++) {
        temp_array = v[n - j];
        v[n - j] = v[(n - j) / 2];
        v[(n - j) / 2] = temp_array;
      }
      timer(v, N, i);
    }

    int main()
    {
      setlocale(LC_ALL, "Russian");
      mt19937 engine(time(0));
      int N[] = { 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000 }; // массив различных длин
массива
      vector <double> v; // массив
      clearFile(); // очистка файла
      for (int j = 0; j < size(N); j++) {
        vector <double> test_array(N[j]); // Массив из нулей
        nullSort(test_array, v.size()); // Сортировка нулевого массива
        for (int i = 0; i < M; i++) { // M тестов
          piece();

          // генерация случайных чисел от -1 до 1
          uniform_real_distribution<double> gen(-1.0, 1.0);
          for (int el = 0; el < N[j]; el++) {
```

```
            v.push_back(gen(engine));
        }
        timer(v, v.size(), i); // таймер
        sortingSort(v, v.size(), i); // Отсортированный массив
        antiQSort(v, v.size(), i); // Массив с макс колвом сравнений
        determQSort(v, v.size(), i); // Массив с макс колвом сравнений
        v.clear();

    }
    piece();
    test_array.clear();
  }
  cout << "Programm end!";
}
```

Заключение.

В заключении можно заметить, что метод быстрой сортировки действительно занимает меньше времени на сортировку массива. Однако эта скорость зависит от того, какой массив был сгенерирован изначально. В частных случаях опорный элемент может выбираться не эффективно, поэтому массив будет разделен на части из 1 и N-1 элементов. Таким образом, можно сказать, что быстрая сортировка сильно зависит от самого массива.