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: 06.03.2023

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 быстрой сортировки

1. Функция quickSort принимает на вход вектор v элементов, начальный и конечный элементы сортировки begin и last, а также глубина данной ветки стека depth.
   Функция начинается с увеличения значений количества рекурсий count_of_recursion и глубины стека height.
   Далее функция проверяется является ли начальный элемент сортировки меньше конечного. Если да, то с помощью функции partition находится индекс опорного элемента. После чего вызываются две рекурсивные функции quickSort для части вектора до опорного и после него.

```
void quickSort(vector <double> v, int begin, int last, int depth) {
    double pivot;
    count_of_recursion++;
    if (depth > height) height = depth;
    if (begin < last) {
        pivot = partition(v, begin, last);
        quickSort(v, begin, last: pivot - 1, depth + 1);
        quickSort(v, begin: pivot + 1, last, depth + 1);
    }
}
```

2. Функция partition принимает на вход вектор элементов v, начальный и конечный элементы сортировки begin и last.
   Выбирается значение опорного элемента pivot_value. Далее идет перестановка элементов относительно опорного элемента. После итерации свободный элемент заменяется элементом с индексом i+1. Функция возвращает индекс опроного элемента.

```
int partition(vector <double> v, int begin, int last) {
    double pivot_value = v[last];
    int i = begin - 1;
    double temp;
    for (int j = begin; j < last; j++) {
        if (v[j] <= pivot_value) {
            i++;
            temp = v[i];
            v[i] = v[j];
            v[j] = temp;
        }
    }
    temp = v[i + 1];
    v[i + 1] = v[last];
    v[last] = temp;
    return i + 1;
}
```

Начальные значения

1. В качестве начальных значений были заданы:
    - Количество тестов M = 20;
    - Массив длин вектора N[] = {1000, 2000, 4000, 8000, 16000, 32000};
2. Задание вектора элементов

```
// генерация случайных чисел от -1 до 1
mt19937 engine(time( _Time: 0));
uniform_real_distribution<double> gen(-1.0, 1.0);
for (int el = 0; el < N[j]; el++) {
    v.push_back( _Val: gen(engine));
}
```

   В качестве вектора из одинаковых элементов, был создан вектор из нулей.
3. Перед первым вызовом функции quickSort обнуляются значения глубины стека и количества рекурсий (depth, height и count_of_recursion). Также задаются начальные значения индексов начального и конечного элементов сортировки (begin = 0, last = v.size() - 1).

Запись в файл

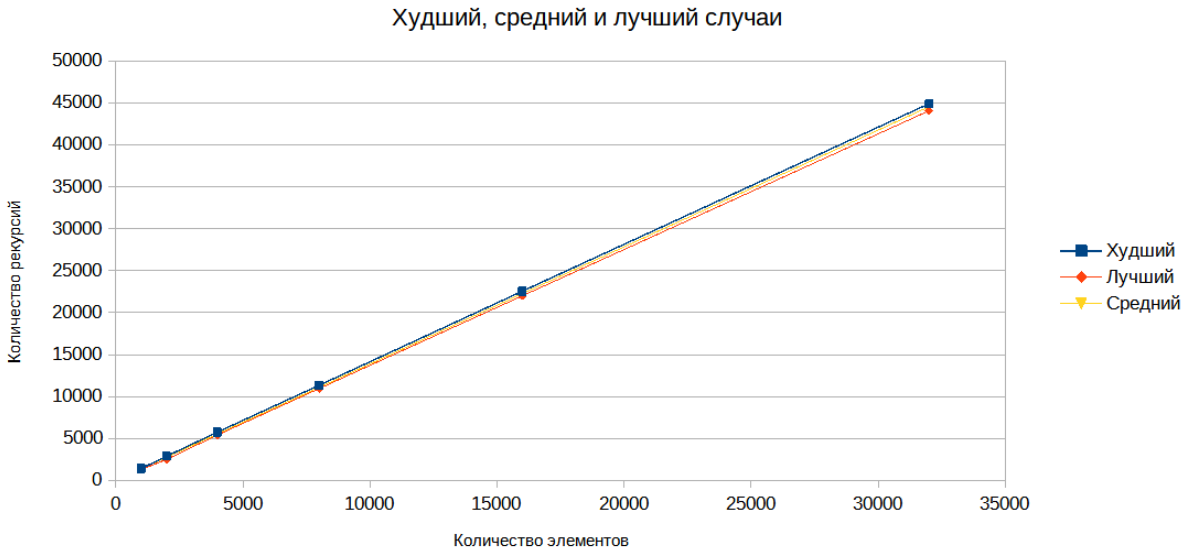
1. С помощью функции writer полученные результаты записываются в файл. А именно: затраченное на сортировку время в миллисекундах, количество рекурсий и максимальная глубина стека.

Обычный вектор

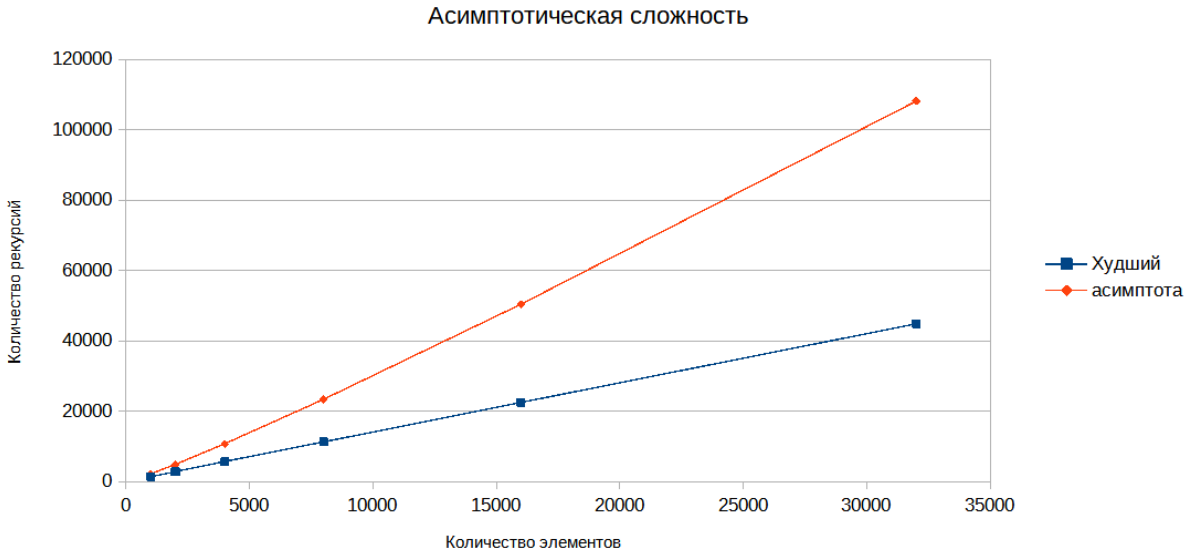
1. Обычный вектор элементов был получен с помощью генерации случайных чисел от -1 до 1. Было проведено 20 тестов для различных размеров вектора (а именно, 1000, 2000, 4000, 8000, 16000, 32000).
2. Количество рекурсий в зависимости от длины вектора.
    - Полученные результаты

Количество рекурсий						
	1000	2000	4000	8000	16000	32000
1	1364	2754	5756	11018	22544	44538
2	1300	2714	5580	11156	22266	44394
3	1318	2694	5482	11002	22360	44590
4	1400	2504	5706	11188	22212	44754
5	1370	2796	5414	11028	22306	44594
6	1396	2688	5438	11038	22194	44534
7	1374	2830	5560	11080	22380	44360
8	1390	2776	5478	11056	22462	44632
9	1422	2788	5582	11248	22162	44704
10	1392	2856	5536	11224	22288	44836
11	1416	2718	5466	11098	22214	44050
12	1386	2698	5504	11018	22316	44374
13	1428	2748	5570	11032	21994	44880
14	1418	2840	5500	11004	22120	44042
15	1372	2756	5496	11102	22032	44302
16	1406	2748	5520	10942	22084	44712
17	1408	2884	5470	11148	22308	44460
18	1372	2788	5514	11328	22264	44812
19	1394	2794	5708	11002	22154	44506
20	1404	2740	5542	11096	22440	44322
Худший	1428	2884	5756	11328	22544	44880
Лучший	1300	2504	5414	10942	21994	44042
Средний	1386,5	2755,7	5541,1	11090,4	22255	44519,8
0,75	2250	4951,544993	10806,17997	23418,53992	50449,43979	108123,5995

- График худшего, среднего и лучшего случаев для количества рекурсий в зависимости от длины вектора.



- Был построен график асимптотической сложности с помощью формулы c*O(N*LOG(N)). Для данного худшего случая c = 0,75 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

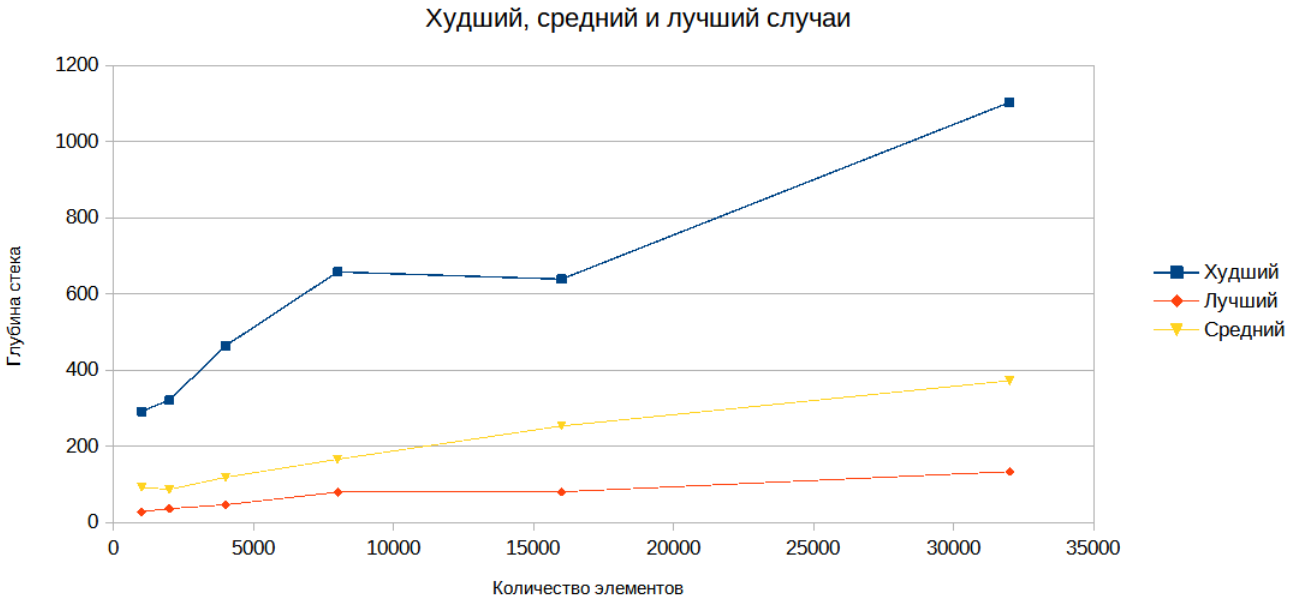


3. Глубина стека в зависимости от длины вектора.

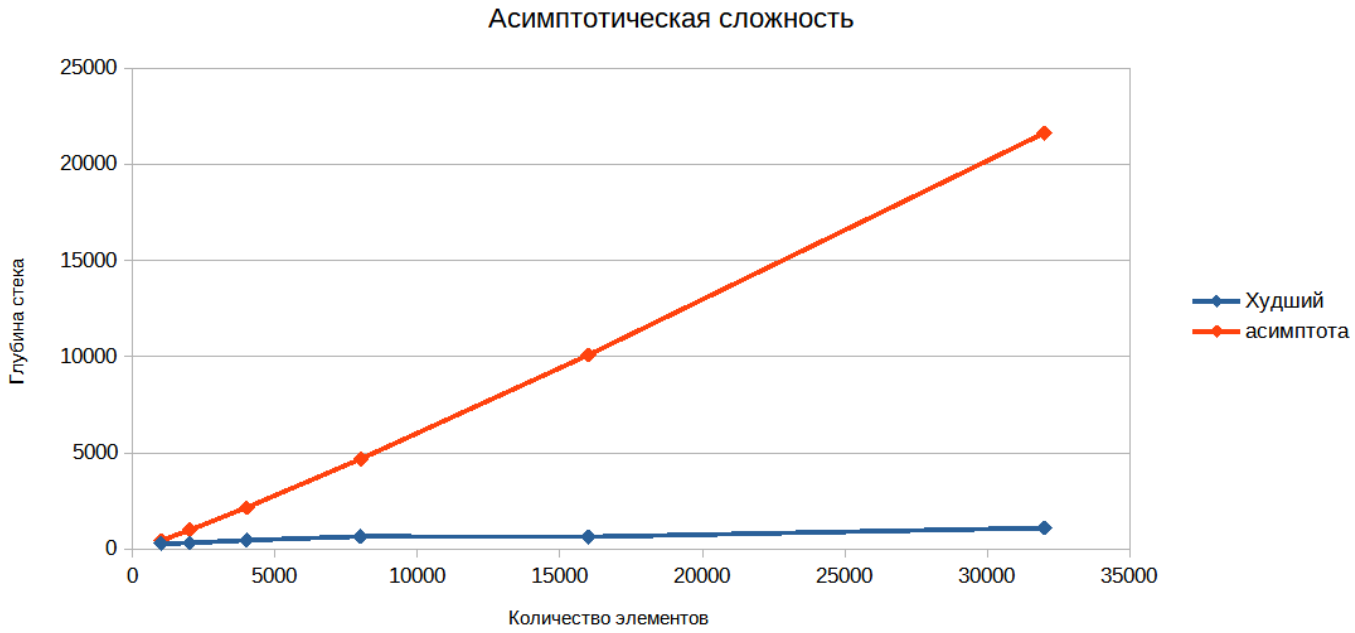
- Полученные результаты

Глубина стека						
	1000	2000	4000	8000	16000	32000
1	44	45	464	107	636	245
2	111	44	97	185	251	391
3	224	110	67	158	292	350
4	43	322	250	116	149	284
5	117	72	63	105	81	270
6	71	37	69	84	114	223
7	66	88	119	105	210	163
8	51	69	104	187	639	332
9	43	47	114	181	170	160
10	113	101	69	80	251	261
11	29	50	80	110	99	1103
12	59	44	75	110	189	188
13	50	98	55	148	142	886
14	292	48	139	254	262	159
15	78	46	47	95	196	154
16	56	241	61	111	264	350
17	58	101	96	274	292	815
18	244	48	85	163	452	164
19	51	77	237	93	142	134
20	54	55	88	658	243	820
Худший	292	322	464	658	639	1103
Лучший	29	37	47	80	81	134
Средний	92,7	87,15	118,95	166,2	253,7	372,6
0,15	450	990,3089987	2161,235995	4683,707984	10089,88796	21624,7199

- График худшего, среднего и лучшего случаев для глубины стека в зависимости от длины вектора.



- Был построен график асимптотической сложности с помощью формулы c*O(N*LOG(N)). Для данного худшего случая c = 0,15 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.



Отсортированный вектор

1. В качестве отсортированного вектора берется тот же вектор, который был создан ранее и отсортирован быстрой сортировкой. Было также проведено по 20 тестов для различных длин вектора. Результаты также были записаны в файл.
2. Проанализировав получившиеся результаты, можно отметить, что и количество рекурсий, и максимальная глубина стека полностью совпадает с сортировкой обычного вектора. Из этого следует, что графики количества рекурсий и глубины стека в зависимости от количества элементов в векторе будут аналогичными как и в обычном векторе.

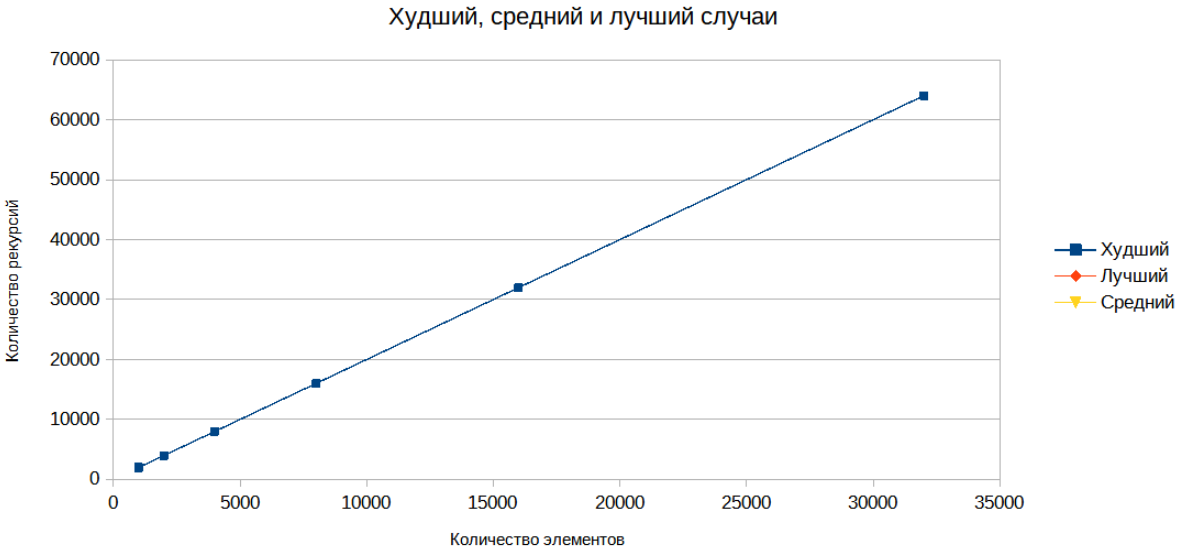
Вектор с одинаковыми элементами (нулевой вектор)

1. Вектор из нулей был получен с помощью изначального задания размера вектора. Было проведено по одному тесту для различных размеров вектора (а именно, 100, 200, 400, 800, 1000). Так как при проведении нескольких тестов на одинаковых по размеру векторах получалось одинаковое количество рекурсий и глубина стека, стало понятно, что есть смысл проводить только по одному тесту на каждую длину.
2. Количество рекурсий в зависимости от длины вектора.
    - Полученные результаты

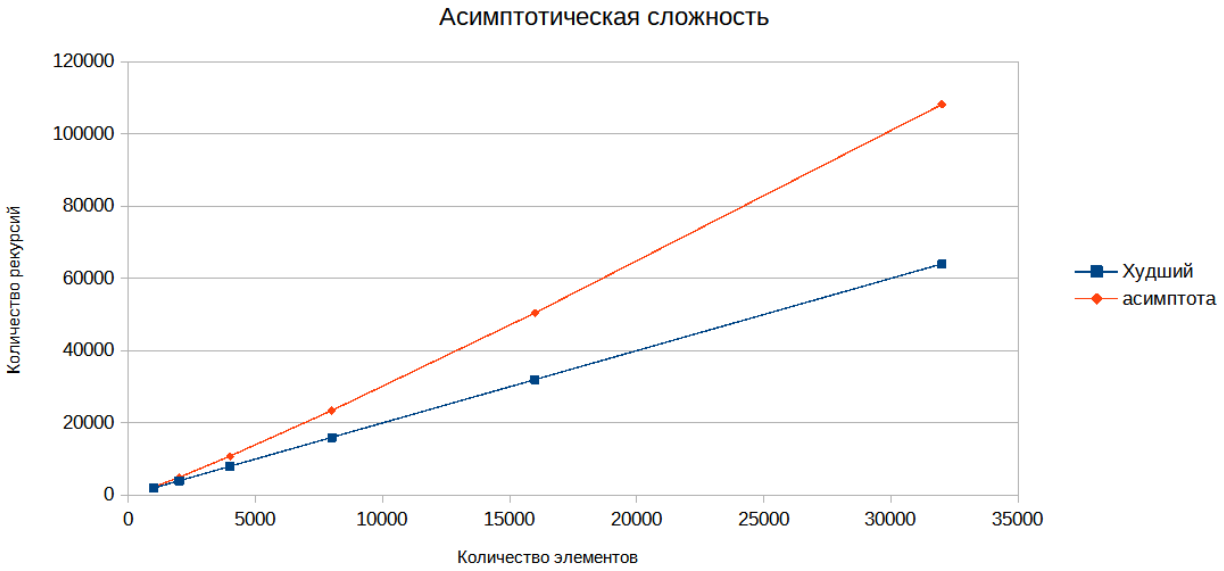
      Из полученных результатов можно сказать, что количество рекурсий в зависимости от длины вектора N можно получить по формуле 2*(N - 1). Тем самым можно сделать таблицу для более длинных векторов.

Количество рекурсий						
	1000	2000	4000	8000	16000	32000
1	1998	3998	7998	15998	31998	63998
Худший	1998	3998	7998	15998	31998	63998
Лучший	1998	3998	7998	15998	31998	63998
Средний	1998	3998	7998	15998	31998	63998
0,75	2250	4951,544993	10806,17997	23418,53992	50449,43979	108123,5995

- График худшего, среднего и лучшего случаев для количества рекурсий в зависимости от длины вектора. Очевидно, что они будут полностью совпадать.



- Был построен график асимптотической сложности с помощью формулы c*O(N*LOG(N)). Для данного худшего случая c = 0,75 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.



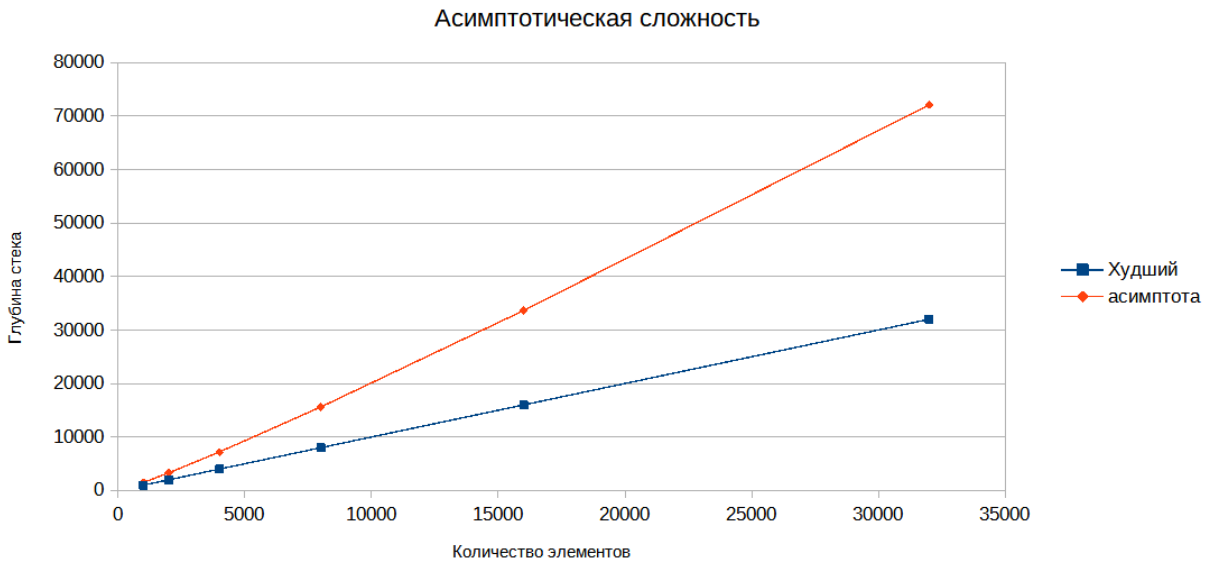
3. Глубина стека в зависимости от длины вектора.
    - Полученные результаты

      Из полученных результатов можно сказать, что глубину стека в зависимости от длины вектора N можно получить по формуле N - 1. Тем самым можно сделать таблицу для более длинных векторов.

Глубина стека						
	1000	2000	4000	8000	16000	32000
1	999	1999	3999	7999	15999	31999
Худший	999	1999	3999	7999	15999	31999
Лучший	999	1999	3999	7999	15999	31999
Средний	999	1999	3999	7999	15999	31999
0,5	1500	3301,029996	7204,119983	15612,35995	33632,95986	72082,39965

    - График худшего, среднего и лучшего случаев для глубины стека в зависимости от длины вектора. Очевидно, что они будут полностью совпадать.

    - Был построен график асимптотической сложности с помощью формулы c*O(N*LOG(N)). Для данного худшего случая c = 0,5 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.



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

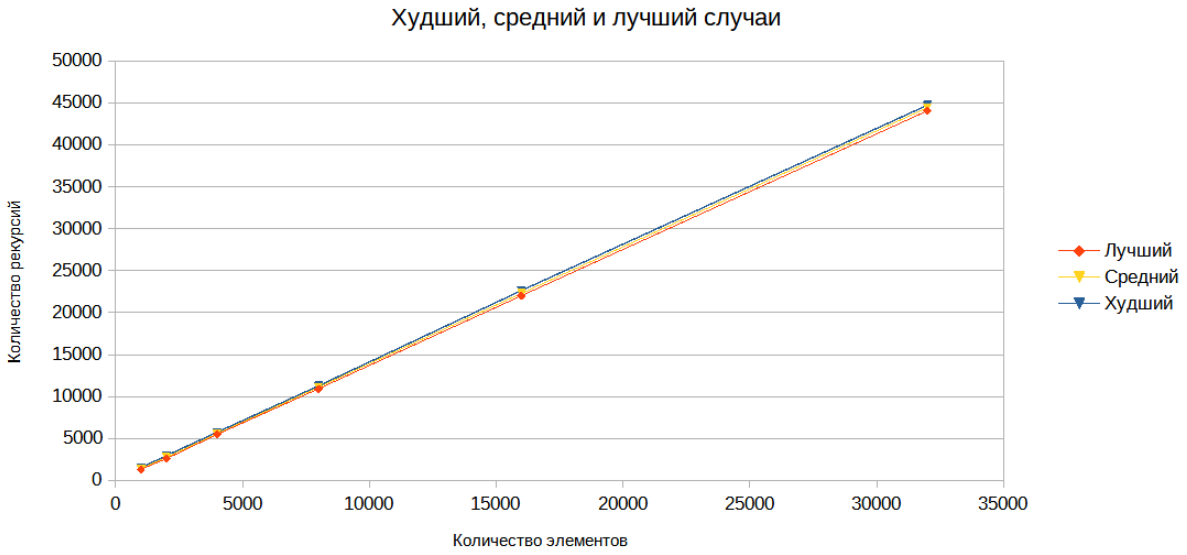
1. Для получения такого вектора было необходимо выполнить алгоритм, который преобразует наш ранее отсортированный вектор таким образом, чтобы при быстрой сортировке опорный элемент выбирался так, чтобы вектор делился на части по 1 и N-1 элементу.
2. Путем итерации элементы меняются местами так, чтобы на каждом шаге в качестве среднего будет выбираться самый крупный элемент.

```
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;
        swap(&: v[added_index], &: v[middle]);
    }
```

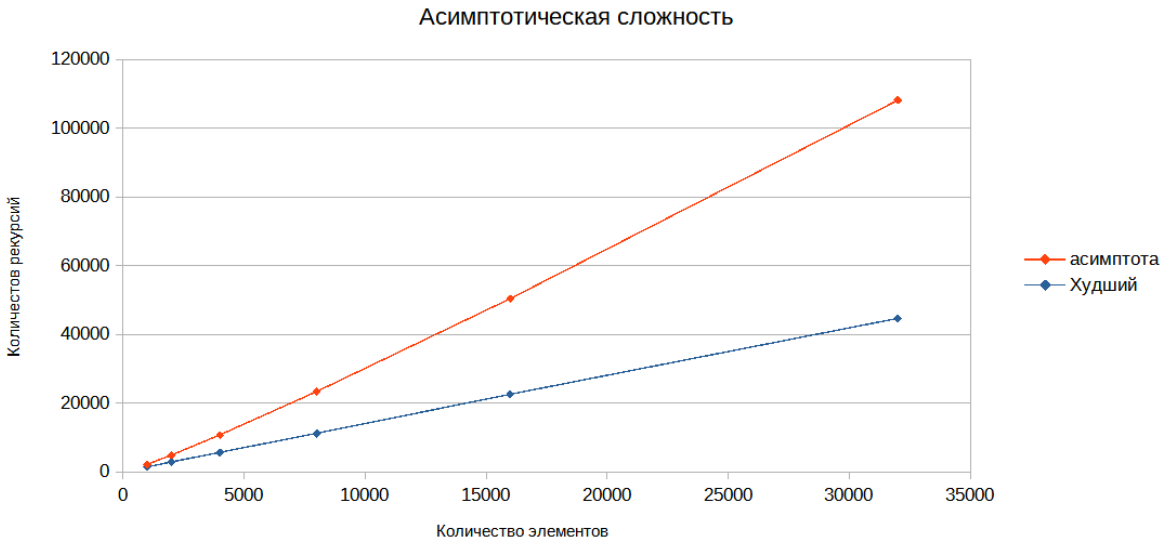
3. Далее также было проведено 20 тестов для различных размеров вектора (а именно, 1000, 2000, 4000, 8000, 16000, 32000).
4. Количество рекурсий в зависимости от длины вектора.
    - Полученные результаты

	Количество рекурсий					
	1000	2000	4000	8000	16000	32000
1	1410	2722	5564	11118	22298	44318
2	1378	2720	5522	11138	22266	44040
3	1326	2812	5564	11084	22422	44394
4	1416	2588	5732	11128	22386	44554
5	1412	2800	5486	11270	22228	44702
6	1470	2758	5472	11178	22006	44304
7	1430	2774	5494	10924	22198	44476
8	1402	2754	5540	11182	22476	44192
9	1394	2856	5554	11200	22364	44476
10	1426	2730	5526	11232	22188	44406
11	1404	2742	5524	10944	22318	44400
12	1418	2950	5554	11056	22628	44452
13	1394	2746	5528	11112	22162	44236
14	1392	2758	5516	11062	22138	44498
15	1396	2734	5534	11076	22392	44666
16	1384	2744	5512	11140	22560	44434
17	1366	2788	5640	10986	22162	44190
18	1320	2732	5548	11138	22076	44468
19	1372	2892	5738	11084	22202	44480
20	1508	2788	5610	11160	22380	44596
Худший	1508	2950	5738	11270	22628	44702
Лучший	1320	2588	5472	10924	22006	44040
Средний	1400,9	2769,4	5557,9	11110,6	22292,5	44414,1
0,75	2250	4951,544993	10806,17997	23418,53992	50449,43979	108123,5995

- График худшего, среднего и лучшего случаев для количества рекурсий в зависимости от длины вектора.



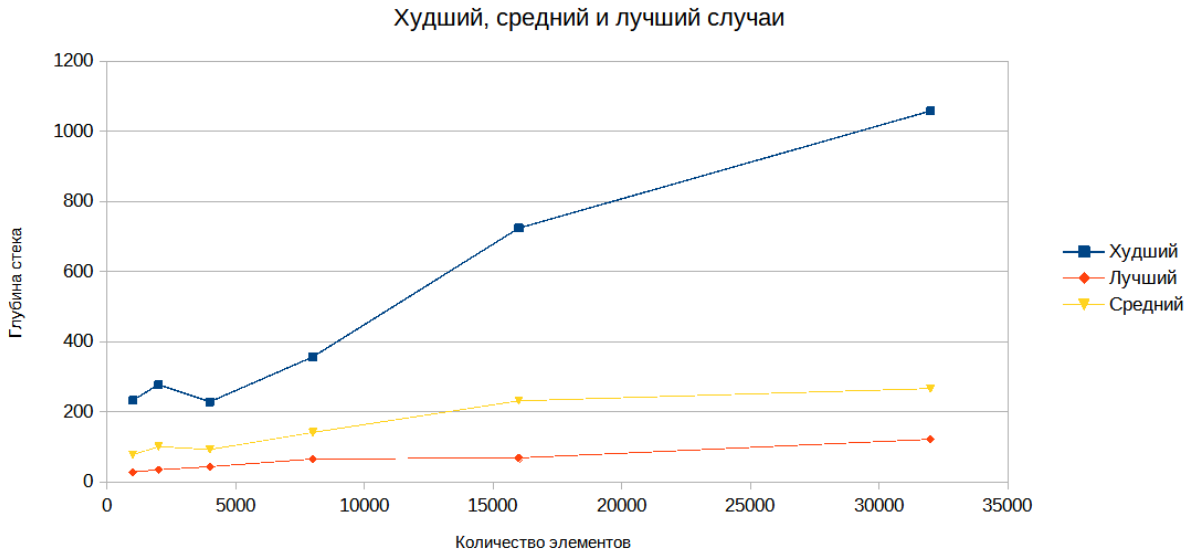
- Был построен график асимптотической сложности с помощью формулы c*O(N*LOG(N)). Для данного худшего случая c = 0,75 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.



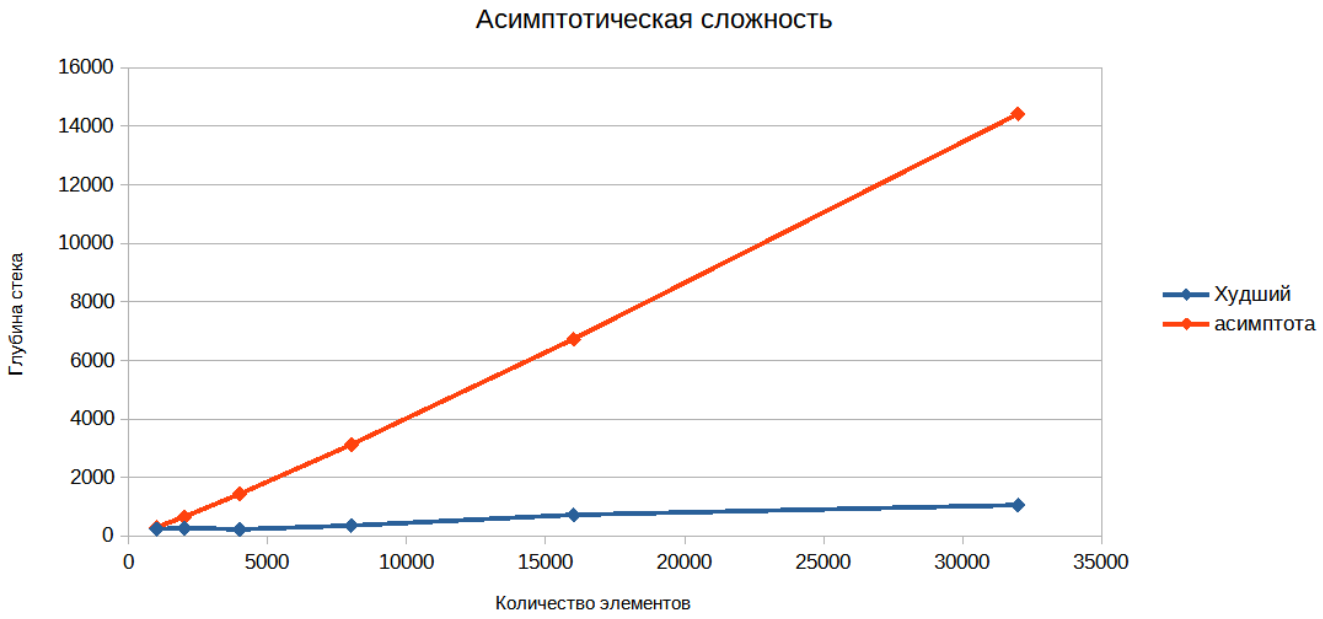
5. Глубина стека в зависимости от длины вектора.
   - Полученные результаты

Глубина стека						
	1000	2000	4000	8000	16000	32000
1	171	42	97	141	110	215
2	37	153	66	147	189	1058
3	226	53	53	74	351	220
4	43	278	229	96	271	217
5	60	61	60	145	302	250
6	60	119	61	215	109	201
7	57	91	65	157	131	140
8	45	54	100	116	275	167
9	29	154	135	102	184	143
10	108	64	52	357	69	150
11	64	65	67	101	125	326
12	35	145	104	168	724	325
13	53	46	83	103	130	198
14	43	61	133	236	94	218
15	38	65	57	80	116	451
16	45	260	95	122	469	202
17	33	94	63	256	383	122
18	233	35	44	71	263	428
19	40	125	201	66	140	147
20	132	61	90	77	198	153
Худший	233	278	229	357	724	1058
Лучший	29	35	44	66	69	122
Средний	77,6	101,3	92,75	141,5	231,65	266,55
0,1	300	660,2059991	1440,823997	3122,47199	6726,591972	14416,47993

   - График худшего, среднего и лучшего случаев для глубины стека в зависимости от длины вектора.



- Был построен график асимптотической сложности с помощью формулы c*O(N*LOG(N)). Для данного худшего случая c = 0,1 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.



Вектор с максимальным количеством сравнений при детерминированном выборе опорного элемента

1. Для получения такого вектора было необходимо выполнить алгоритм, который преобразует наш ранее отсортированный вектор таким образом, чтобы при быстрой сортировке опорный элемент выбирался так, чтобы вектор делился на части по 1 и N-1 элементу.
2. Путем итерации элементы меняются местами так, чтобы на каждом шаге в качестве среднего будет выбираться самый крупный элемент.

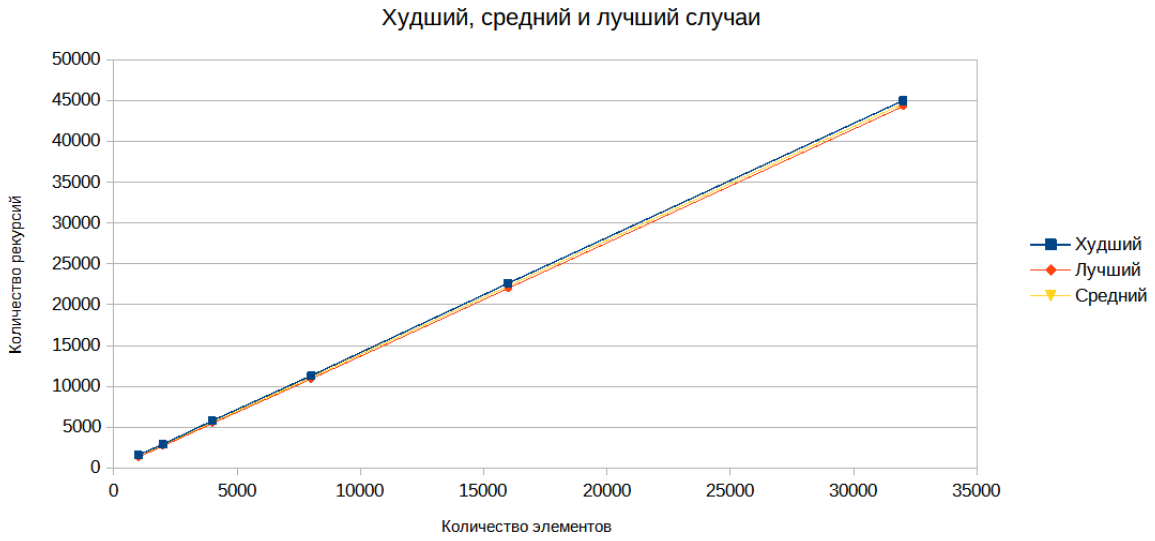
```
void determQSort(vector <double> v, int N, int i) {
    double temp_array;
    int k = 0, n = N - 1;

    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);
}
```

3. Далее также было проведено 20 тестов для различных размеров вектора (а именно, 1000, 2000, 4000, 8000, 16000, 32000).

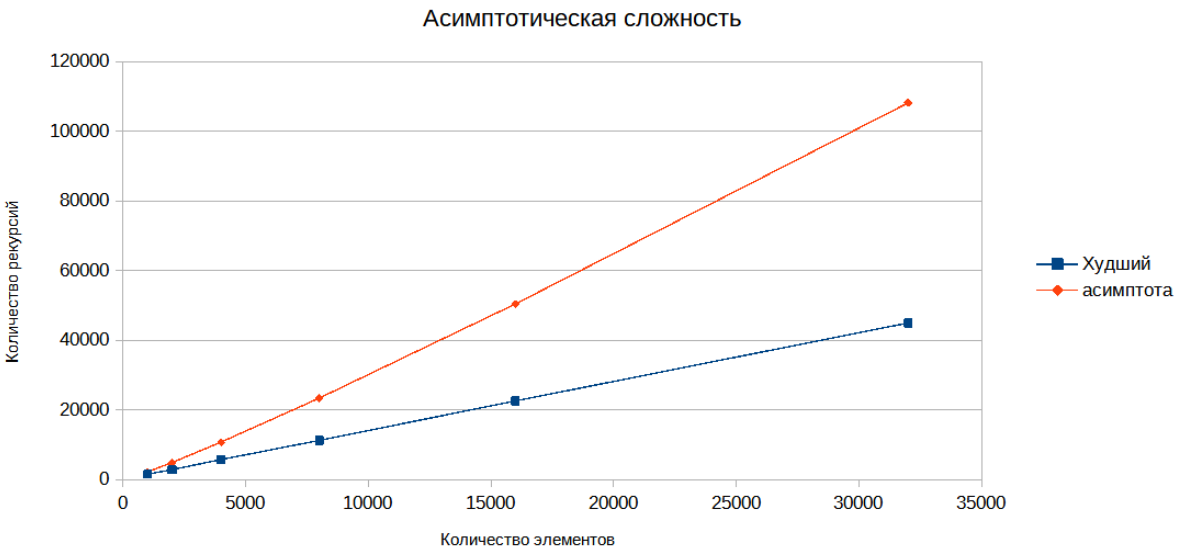
4. Количество рекурсий в зависимости от длины вектора.
    - Полученные результаты

	Количество рекурсий					
	1000	2000	4000	8000	16000	32000
1	1364	2906	5648	10892	22616	44464
2	1346	2820	5548	10978	22014	44276
3	1402	2780	5522	11108	22060	44474
4	1438	2856	5512	10932	22510	44462
5	1372	2700	5526	11234	22116	44778
6	1428	2752	5454	11024	22314	44520
7	1396	2828	5608	11134	22532	44260
8	1388	2778	5490	11294	22534	44808
9	1366	2750	5548	10990	22348	44406
10	1310	2732	5560	11164	21996	44632
11	1384	2752	5550	11046	22578	44672
12	1400	2798	5486	11154	22002	44462
13	1462	2686	5608	11054	22240	44760
14	1372	2818	5540	11074	22056	44600
15	1350	2850	5534	11008	22328	44492
16	1416	2690	5638	11004	22260	44526
17	1342	2762	5600	11208	22572	44974
18	1378	2818	5592	11170	22032	44718
19	1402	2752	5502	11148	22150	44514
20	1600	2898	5790	11114	22402	44822
Худший	1600	2906	5790	11294	22616	44974
Лучший	1310	2686	5454	10892	21996	44260
Средний	1395,8	2786,3	5562,8	11086,5	22283	44581
0,75	2250	4951,544993	10806,17997	23418,53992	50449,43979	108123,5995

- График худшего, среднего и лучшего случаев для количества рекурсий в зависимости от длины вектора.



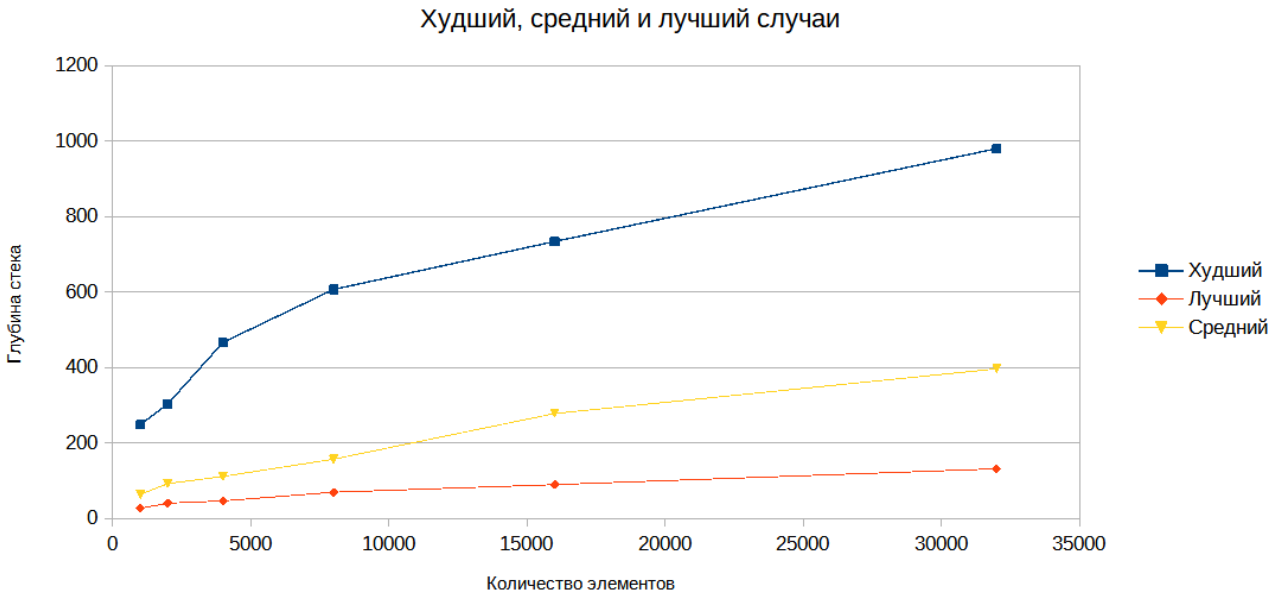
- Был построен график асимптотической сложности с помощью формулы c*O(N*LOG(N)). Для данного худшего случая c = 0,75 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.



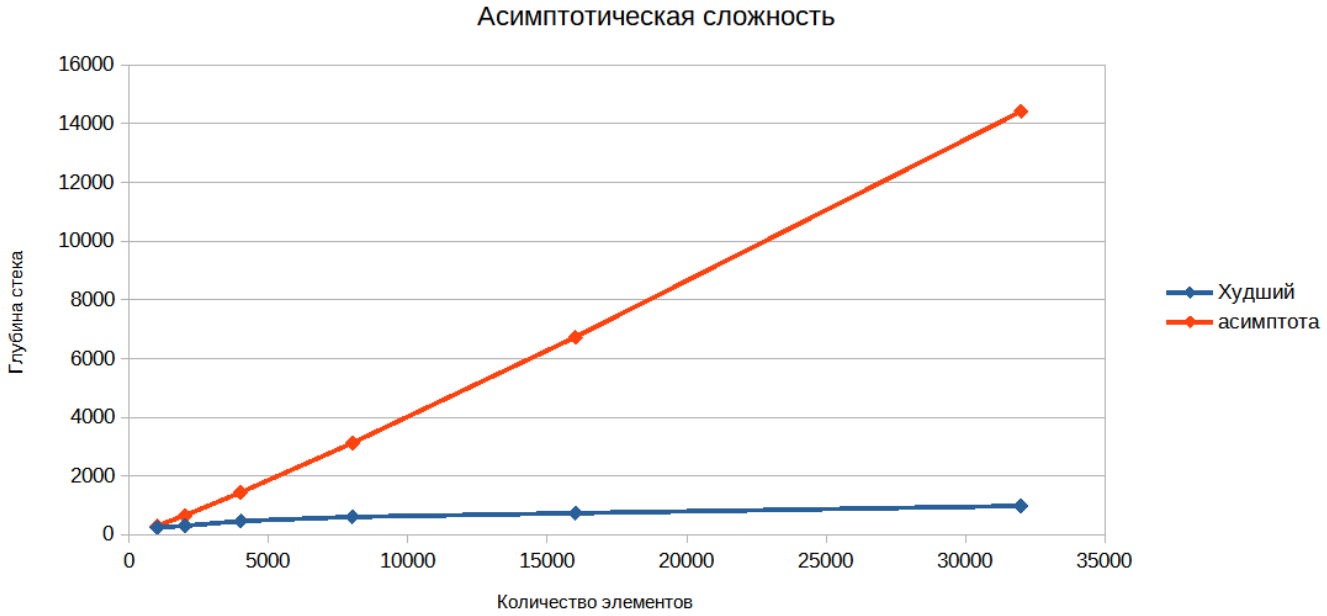
5. Глубина стека в зависимости от длины вектора.
   - Полученные результаты

	Глубина стека					
	1000	2000	4000	8000	16000	32000
1	63	88	90	97	734	221
2	84	46	54	106	261	216
3	27	58	63	109	361	292
4	65	141	47	98	219	134
5	70	56	101	175	159	704
6	71	75	79	91	368	504
7	31	83	93	607	634	250
8	83	65	125	163	128	920
9	49	50	75	110	144	281
10	37	113	72	142	269	353
11	30	75	89	99	187	377
12	46	96	111	130	90	386
13	136	41	98	108	226	726
14	57	51	110	218	154	160
15	37	106	198	82	200	228
16	30	83	80	276	181	305
17	35	304	102	196	468	980
18	33	49	82	154	192	432
19	51	62	100	118	144	132
20	249	212	467	70	448	333
Худший	249	304	467	607	734	980
Лучший	27	41	47	70	90	132
Средний	64,2	92,7	111,8	157,45	278,35	396,7
0,1	300	660,2059991	1440,823997	3122,47199	6726,591972	14416,47993

- График худшего, среднего и лучшего случаев для глубины стека в зависимости от длины вектора.



- Был построен график асимптотической сложности с помощью формулы c*O(N*LOG(N)). Для данного худшего случая c = 0,1 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.



Заключение.

В заключении можно заметить, что метод быстрой сортировки действительно занимает меньше времени на сортировку массива. Также алгоритм самой сортировке достаточно просто в реализации. Однако 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.

Помимо этого, проблема данной сортировки в том, что из-за того, что функция сортировки вызывает сама себя, тем самым происходит заполнение стека рекурсий. При достаточно большом векторе может произойти такое, что стек рекурсий будет переполнен, что приведет к остановке программы. Именно поэтому в лабораторной работе не были выполнены тесты на 64000 и 128000 элементов.

Также из недостатков можно выделить то, что данная сортировка сильно деградирует по скорости до $O(N^2)$. На это сильно влияет сгенерированный изначально массив элементов.

Для того, чтобы избежать переполнение стека рекурсий стоит использовать другой метод сортировки, например, более медленных.