МИНОРБНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«МИРЭА - Российский технологический университет»

РТУ МИРЭА

Отчёт по выполнению практического задания № 3
Тема:
«Определение эффективного алгоритма сортировки на основе эмпирического и асимптотического методов анализа»
Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Фамилия И.О.
Фамилия И.О.

Группа: АААА-00-00
Номер группы

Москва 2023

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Цель работы

Получить навыки по анализу вычислительной сложности алгоритмов сортировки и определению наиболее эффективного алгоритма.

1.2 Задание 1

Эмпирическая оценка эффективности алгоритмов.

1. Разработать алгоритм ускоренной сортировки, определенной в варианте, реализовать код на языке C++. Сформировать таблицу результатов эмпирической оценки сложности сортировки по формату табл. 1 для массива, заполненного случайными числами 1.

Таблица 1 - Сводная таблица результатов

n	T(n), мс	$T_п(n) = C_ф + M_ф$
100		
1000		
10000		
100000		
1000000		

2. Определить ёмкостную сложность алгоритма ускоренной сортировки.

3. Разработать алгоритм быстрой сортировки, определенной в варианте, реализовать код на языке C++. Сформировать таблицу результатов эмпирической оценки сортировки по формату табл. 1 для массива, заполненного случайными числами.

4. Определить ёмкостную сложность алгоритма быстрой сортировки.

5. Добавить в отчёт данные по работе любого из алгоритмов простой сортировки в среднем случае, полученные в предыдущей практической работе.

6. Представить на общем сравнительном графике зависимости $T_п(n)=C_ф+M_ф$ для трёх анализируемых алгоритмов 2. График должен быть подписан, на нём – обозначены оси.

7. На основе сравнения полученных данных определите наиболее эффективный из алгоритмов в среднем случае (отдельно для небольших

массивов при n до 1000 и для больших массивов с n>1000).

8. Провести дополнительные прогоны программ ускоренной и быстрой сортировок на массивах, отсортированных строго в убывающем и строго в возрастающем порядке значений элементов. Заполнить по этим данным соответствующие таблицы для каждого алгоритма по формату табл. 1.

9. Сделать вывод о зависимости (или независимости) алгоритмов сортировок от исходной упорядоченности массива на основе результатов, представленных в таблицах.

1.2 Задание 2

Асимптотический анализ сложности алгоритмов

1. Из материалов предыдущей практической работы приведите в отчёте формулы $T_т(n)$ функций роста алгоритма простой сортировки в лучшем и худшем случае (того же алгоритма, что и в задании 1).

2. На основе определений соответствующих нотаций получите асимптотическую оценку вычислительной сложности простого алгоритма сортировки 3:

- в O-нотации (оценка сверху) для анализа худшего случая;

- в Ω-нотации (оценка снизу) для анализа лучшего случая.

3. Получить (если это возможно) асимптотически точную оценку вычислительной сложности алгоритма в нотации θ.

4. Реализовать графическое представление функции роста и полученных асимптотических оценок сверху и снизу.

5. Привести справочную информацию о вычислительной сложности усовершенствованного и быстрого алгоритмов сортировки, заданных в вашем варианте.

6. Общие результаты свести в табл. 2.

Таблица 2 - Сводная таблица результатов

Алгоритм	Асимптотическая сложность алгоритма			
	Наихудший случай (сверху)	Наилучший случай (снизу)	Средний случай (снизу)	Ёмкостная сложность
Простой				
Усовершен-ствованный				
Быстрый				

7. Сделать вывод о наиболее эффективном алгоритме из трёх.

1.3 Индивидуальный вариант

Таблица 3 - Индивидуальный вариант алгоритмов сортировки

Вариант	Усовершенствованный алгоритм	Быстрый алгоритм
1	Сортировка обменами с условием Айверсона	Простое слияние

2 ЗАДАНИЕ 1

2.1 Алогритм сортировки обменами с условием Айверсона

Сортировка простым обменом (Exchange Sort, пузырьковая).

В цикле, с начала массива до границы рассматриваемой области массива (в первом проходе до n-1, уменьшаясь с каждым разом на 1) соседние значения попарно сравниваются между собой и в случае, если предшествующий элемент больше последующего, меняются местами.

После первого прохода всего массива элемент с максимальным значением займет место последнего элемента – свою окончательную позицию, потому исключается из дальнейшего рассмотрения.

Алгоритм завершится, когда: а) рассматриваемая область станет пустой, или б) когда на очередном проходе по массиву не будет произведено ни одной перестановки (условие Айверсона).

Алгоритм представлен на рис. 1. Его блок-схема на рис. 2.

```cpp
void exchangeSort(int *x, int n)
{
    long long int move_count = 0;
    long long int comp_count = 0;
    auto start = high_resolution_clock::now();

    // Внешний цикл
    for (int i = 0; i < n - 1; i++)
    {
        bool iver = true;
        // Внутренний цикл
        for (int j = i + 1; j < n; j++)
        {
            comp_count += 1;

            // Сравнение
            if (x[i] > x[j])
            {
                iver = false;
                move_count += 1;
                // Перемещение
                int temp = x[i];
                x[i] = x[j];
                x[j] = temp;
            }
        }
        // Завершение сортировки по условию Айверсона
        if(iver) break;
    }

    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start).count();

    cout << "n = " << n << ": "
        << "Сравнений - " << comp_count
        << ", Перемещений - " << move_count
        << ", Время: " << duration << " мс."
        << endl;
}
```

Рисунок 1 - Алгоритм обмена с условием Айверсона

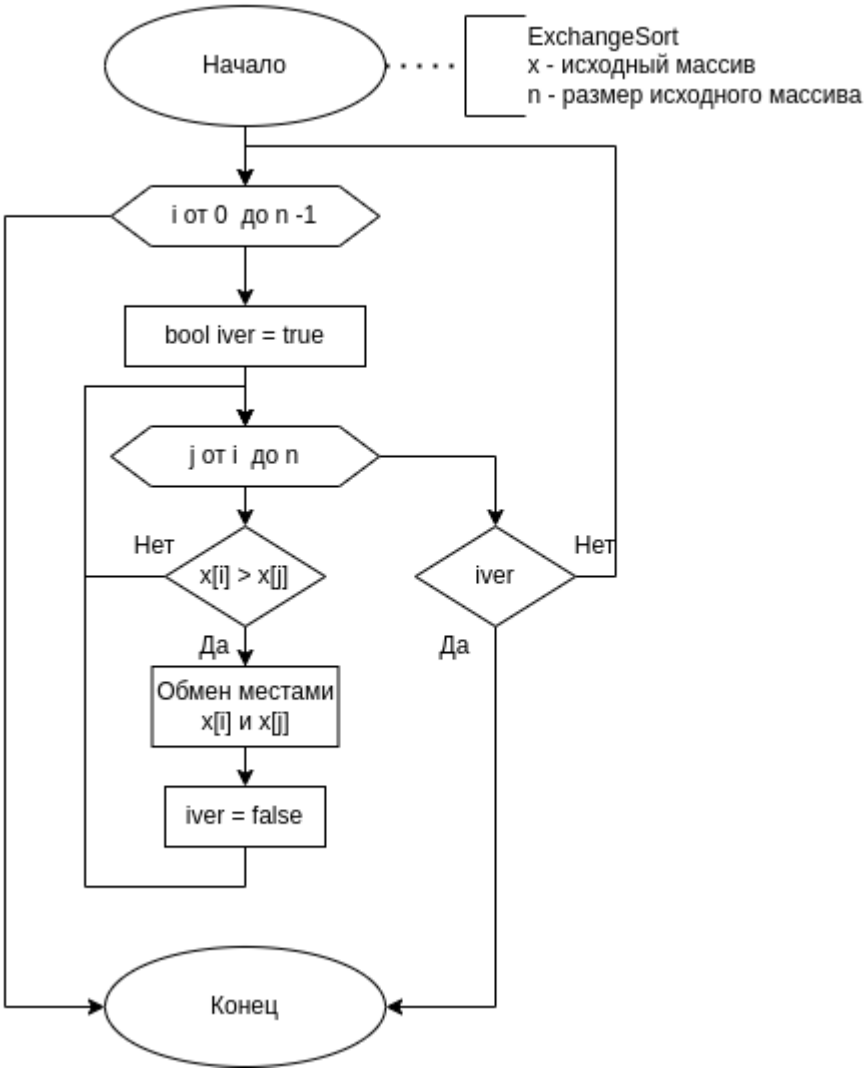

Рисунок 2 - Блок-схема сортировки обменом с условием Айверсона

2.2 Эмпирическая оценка сложности сортировки обменом с условием Айверсона

Результаты тестирования приведены на рис. 3

Сводная таблица результатов тестирования алгоритма для эмпирической оценки сложности представлена в табл. 4.

```
Сортировка вставки с условием Айверсона (Случайный массив)
n = 100: Сравнений - 4845, Перемещений - 346, Время: 0 мс., Всего операций: 5191

n = 1000: Сравнений - 493829, Перемещений - 3986, Время: 0 мс., Всего операций: 497815

n = 10000: Сравнений - 49311135, Перемещений - 39895, Время: 81 мс., Всего операций: 49351030

n = 100000: Сравнений - 4938139479, Перемещений - 400450, Время: 7921 мс., Всего операций: 4938539929
```

Рисунок 3 - Результаты тестирования

Таблица 4 - Сводная таблица результатов

n	T(n), мс	$T_n(n) = C_ф + M_ф$
100	0	5 191
1000	0	497 815
10000	81	49 351 030
100000	7921	4 938 539 929

2.3 Ёмкостная сложность сортировки обмена с условием Айверсона

Количество переменных, создаваемых в функции, не зависит от входных данных. Поэтому V(n) – это константа.

2.4 Алгоритм сортировки слиянием

Сортируемый массив разбивается на две части примерно одинакового размера. Каждая из получившихся частей сортируется отдельно, например, тем же самым алгоритмом. Два упорядоченных подмассива половинного размера соединяются в один. Т.о. задача разбивается на несколько более простых, подобных исходной, а сами простые задачи решаются рекурсивно. Полученные решения комбинируются для получения окончательного решения. Этапы решения:

А. Разделение:

Сортируемая последовательность, состоящая из n элементов, разбивается на две меньшие последовательности, каждая из которых содержит n/2 элементов.

Б. Рекурсивное решение:

Сортировка обеих вспомогательных последовательностей методом слияния.

В. Комбинирование (слияние):

Слияние двух отсортированных последовательностей для получения окончательного результата.

Рекурсия достигает своего нижнего предела, когда длина сортируемой подпоследовательности становится равной 1.

Реализация алгоритма слияния представлена на рис. 4, сама сортировка на рис. 5, блок-схемы алгоритмов представлены на рис. 6, 7.

```
void merge(int *x, int left, int mid, int right)
{
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int *L = new int[n1];
    int *R = new int[n2];
    for (int i = 0; i < n1; i++)
    {
        L[i] = x[left + i];
    }
    for (int j = 0; j < n2; j++)
    {
        R[j] = x[mid + j + 1];
    }

    int i = 0, j = 0;
    L[n1] = numeric_limits<int>::max();
    R[n2] = numeric_limits<int>::max();
    for (int k = left; k <= right; k++)
    {
        if (L[i] <= R[j])
        {
            x[k] = L[i];
            i++;
        }
        else
        {
            x[k] = R[j];
            j++;
        }
    }

    delete[] L;
    delete[] R;
}
```

Рисунок 4 - Алгоритм слияния

```
void mergeSort(int *x, int begin, int end)
{
    if (begin < end)
    {
        int mid = (int)floor((begin + end) / 2);
        mergeSort(x, begin, mid);
        mergeSort(x, mid + 1, end);
        merge(x, begin, mid, end);
    }
}
```

Рисунок 5 - Алгоритм сортировки слиянием

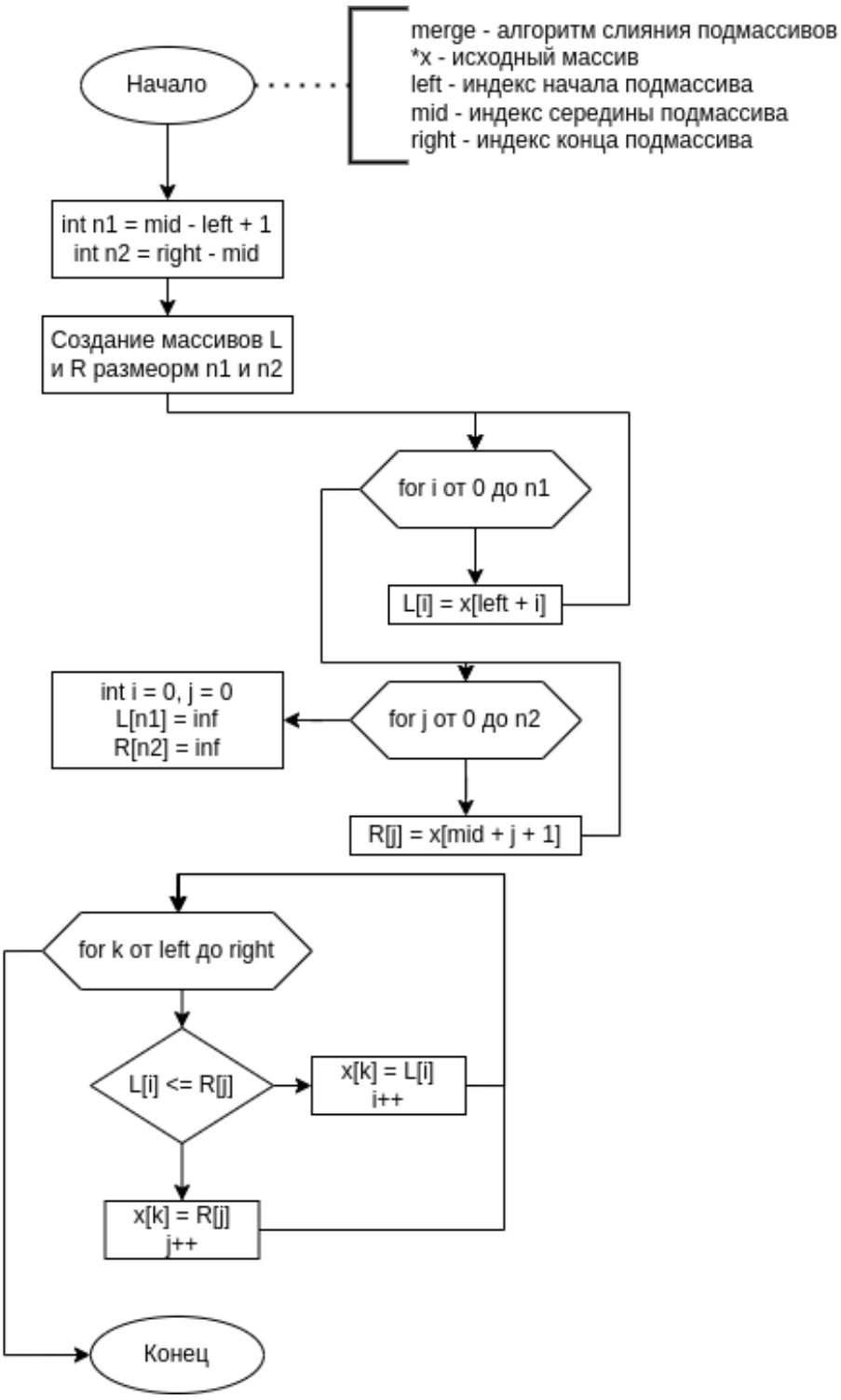

Рисунок 6 - Блок-схема алгоритма слияния

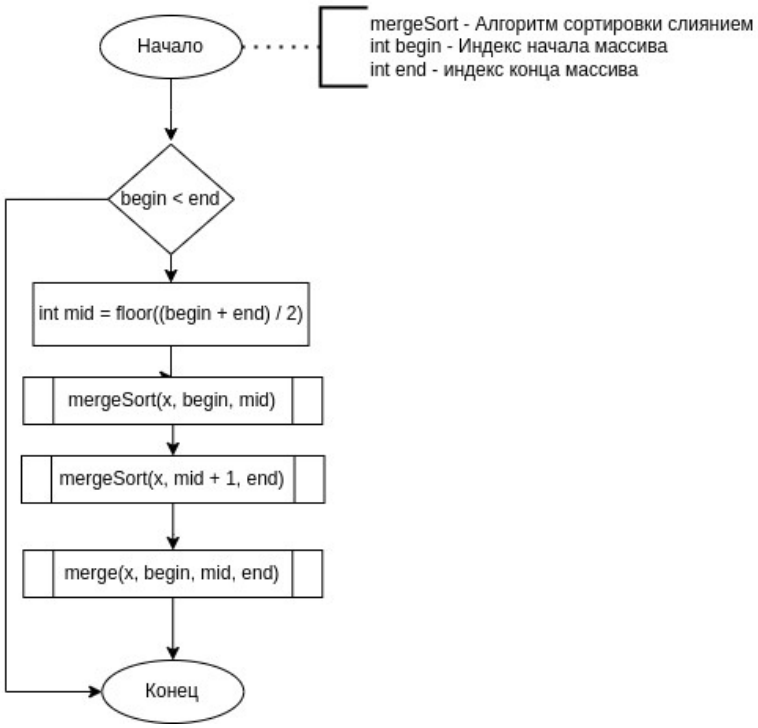

Рисунок 7 - Блок-схема алгоритма сортировки слиянием

2.5 Эмпирическая оценка сложности сортировки обменом с условием Айверсона

Результаты тестирования приведены на рис. 8

Сводная таблица результатов тестирования алгоритма для эмпирической оценки сложности представлена в табл. 5.

```
Сортировка простым слиянием (Случайный массив)
n = 100: Сравнений - 1186, Перемещений - 672, Время: 0 мс., Всего операций: 1858

n = 1000: Сравнений - 16906, Перемещений - 9976, Время: 0 мс., Всего операций: 26882

n = 10000: Сравнений - 218222, Перемещений - 133616, Время: 1 мс., Всего операций: 351838

n = 100000: Сравнений - 2683950, Перемещений - 1668928, Время: 12 мс., Всего операций: 4352878

n = 1000000: Сравнений - 31836414, Перемещений - 19951424, Время: 147 мс., Всего операций: 51787838
```

Рисунок 8 - Результаты тестирования сортировки слиянием

Таблица 5 - Сводная таблица результатов

n	T(n), мс	$T_n(n) = C_ф + M_ф$
100	0	1 858
1000	0	26 882
10000	1	351 838
100000	12	4 352 878
1000000	146	51 787 838

2.6 Ёмкостная сложность сортировки слиянием

В алгоритме создаются подмассивы, которые в сумме дают список длиной n, поэтому алгоритму требуется дополнительная память. И поэтому V(n) = n.

2.7 Данные о сортировке простой вставки

Сводная таблица сортировки простой вставки представлена в табл. 6.
Таблица 6 - Сводная таблица для случайного массива

n	T(n), мс	$T_N = C_N + M_N$
100	7	5091
1000	536	425 377
10000	50611	44 078 245
100000	4872996	4 464 575 959

2.8 Сравнительный график трёх алгоритмов сортировок

График сравнения представлен на рис. 9. Анализируя график, можно сказать, что алгоритм сортировки слиянием значительно показывает себя лучше во всех случаях, и очень эффективен для большинства набора данных.

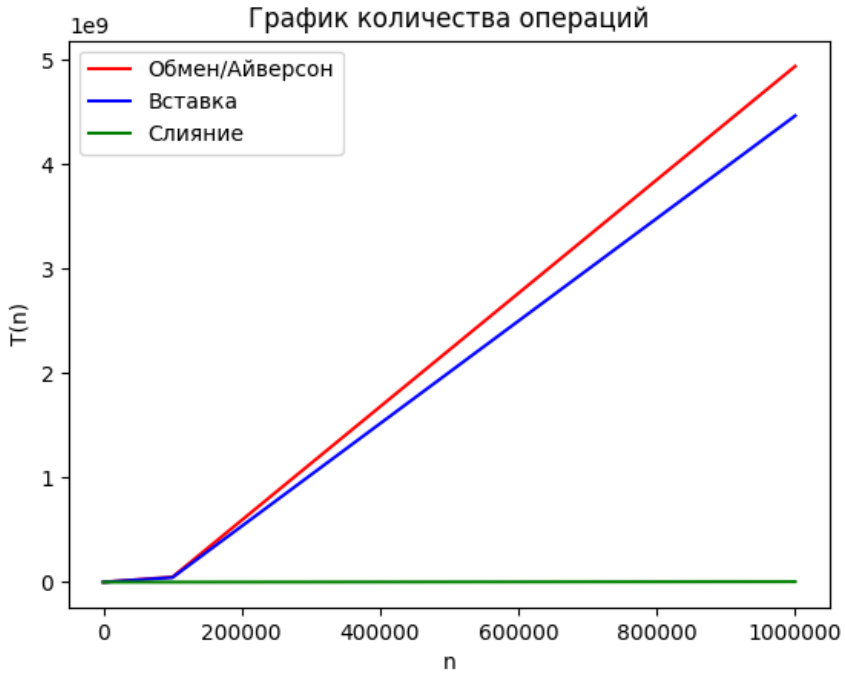

Рисунок 9 - Сравнительный график трёх сортировок

2.9 Дополнительные тесты на отсортированных массивах

Тесты на отсортированном массиве представлены в табл. 7 и 8 для ускоренной и быстрой сортировок соответственно. На убывающем массиве в табл. 9 и 10.

Таблица 7 - Возрастающий ускоренный

n	T(n), мс	$T_n(n) = C_ф + M_ф$
100	0	99
1000	0	999
10000	0	9999
100000	0	99999

Таблица 8 - Возрастающий быстрый

n	T(n), мс	$T_n(n) = C_ф + M_ф$
100	0	1 898
1000	0	26 994
10000	1	351 238
100000	19	4 352 123
1000000	131	51 944 232

Таблица 9 - Убывающий ускоренный

n	T(n), мс	$T_n(n) = C_ф + M_ф$
100	0	9900
1000	1	999000
10000	164	99990000
100000	16338	9999900000

Таблица 10 - Убывающий быстрый

n	T(n), мс	$T_n(n) = C_ф + M_ф$
100	0	1 858
1000	0	26 882
10000	1	351 838
100000	12	4 352 878
1000000	146	51 787 838

2.10 Выводы по эффективности алгоритмов

Исходя из тестов, можно убедиться, что у быстрого алгоритма сортировки слияния одинаковая временная сложность на любых массивах, что делает его не эффективным на практически отсортированных массивах. Ускоренная сортировка обменом с условием Айверсона показывает линейную скорость на отсортированном массиве, поскольку дополнительное условие автоматически завершают сортировку на отсортированном массиве, так что данный алгоритм имеет смысл использовать на практически отсортированных масссивах.

3 ЗАДАНИЕ 2

3.1 Асимптотическая оценка простой сортировки вставками

Для оценки приведём функции роста для данной сортировки.

В лучшем случае $T(n) = 4n — 2 \Rightarrow n$ — линейная зависимость.

В худшем случае $T(n) = 2{,}5n^2 + 3.5n - 4 \Rightarrow n^2$ — квадратичная зависимость

В лучшем случае:

Для записи $T(n)=\Omega(n)$, докажем, что существует константа c такая, что для бесконечного числа значений $n>n0$ выполняется неравенство $T(n) = 4n - 2 \geq c*n$. Разделим обе части на n, тогда $4 — 2/n \geq c$, неравенство выполняется для всех $n \geq 1$, если $c \leq 2$. А значит константа c существует, следовательно $T(n)=\Omega(n)$.

В худшем случае:

Для записи $T(n)=O(n^2)$, докажем, что существует константа c такая, что для бесконечного числа значений $n \geq n0$ выполняется неравенство $T(n) = 2{,}5n^2 + 3.5n - 4 \leq c*n^2$. Разделим обе части на n^2, тогда $2{,}5 + 3.5/n - 4/n^2 \leq c$. Вычислим производную: $D=8/(n^3)-7/(2*n^2)$, для положительных n, точкой максимума будет $n = 16/7$, а значит существуют такие константы c, при которых данное неравенство выполняется, следовательно $T(n)=O(n^2)$.

Точную оценку получить достаточно сложно, поскольку функции роста отличаются на степень, поэтому средняя оценка не будет являться объективной.

Приведём графики данных функций роста (рис. 10)

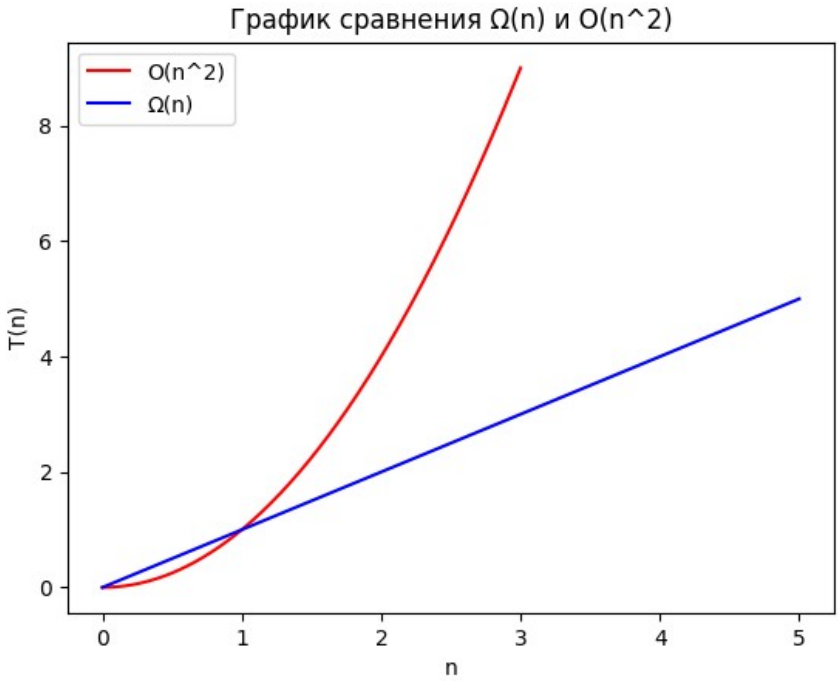

Рисунок 10 - График сравнений функций роста

3.2 Асимптотическая оценка ускоренной и быстрой сортировок

Приведём справочные материалы по данным сортировкам.

Сортировка обменом с условием Айверсона:

Сложность алгоритма $O(n^2)$ операций сравнения и $O(n^2)$ операций перемещения, т.о. в целом сложность алгоритма $O(n^2)$, что не является хорошим результатом. На больших n алгоритму потребуется очень много времени (квадратичная сложность).

Сортировка слиянием:

Для алгоритма сортировки слиянием справедливо рекуррентное

соотношение: T (n) = 2T(n/2) + cn, где cn – время выполнения на одном уровне рекурсии. Отсюда получим log2(n) + 1 уровня раскрытия рекурсии (далее lg(n)).

Общее время работы алгоритма на всех уровнях рекурсии: cn·(lg(n) + 1) = cn·lg(n) + cn. Пренебрегая членами более низких порядков, получаем cn·lg(n).

Порядок роста времени сортировки в среднем случае определяется как $\theta(n \cdot \lg(n))$.

На нижнем k-ом шаге размер задачи $T(n)=\theta(1)=1$, т.е. $n/2k = 1$ или $n= 2k$, значит $k=\lg(n)$. Общее время $k \cdot cn = cn \cdot \lg(n)$, значит $O(n \cdot \lg(n))$.

3.3 Вывод

Для формирования вывода сведём данные об эффективности алгоритмов в табл. 11.

Таблица 11 - Сводная таблица результатов

Алгоритм	Асимптотическая сложность алгоритма			
	Наихудший случай (сверху)	Наилучший случай (снизу)	Средний случай (снизу)	Ёмкостная сложность
Вставками	n^2	n	-	1
Обмена/ Айверсон	n^2	n^2	n^2	1
Слиянием	nlog(n)	nlog(n)	nlog(n)	n

Исходя из содержимого таблицы, можно сделать вывод, что сортировка слиянием будет являться лучшим выбором для любых наборов данных, однако сортировку вставками удобно использовать на малых наборах данных на почти отсортированных массивах.

4 ВЫВОД

Была проделана работа, в ходе которые были вычислены асимптотические сложности алгоритмов, благодаря которым удалось определить наиболее эффективный алгоритм для различных случаев.

4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Структуры и алгоритмы обработки данных : Лекционные материалы / Рысин М. Л. МИРЭА — Российский технологический университет, 2022/23. – 77 с.

2. Merge Sort – Data Structure and Algorithms Tutorialss [Электронный ресурс]. URL: https://www.geeksforgeeks.org/merge-sort/ (дата обращения 11.03.2024).

3. Сортировка вставками - Материал из Википедии — свободной энциклопедии [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Сортировка_вставками (дата обращения 11.03.2024)