МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное
учреждениевысшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №6

Определение эффективного алгоритма сортировки
по дисциплине
«СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ»

Выполнил студент Иолович Е.А.

группа ИНБО-03-22

Москва 2023

СОДЕРЖАНИЕ

1 ЗАДАНИЕ 1 4
1.1 Условие задания и требования 4
1.2 Постановка задачи 4
1.3 Алгоритм сортировки (Простой обмен с условием Айверсона) 4
1.4 Определение асимптотической сложности алгоритма (Простой обмен с условием Айверсона) 4
1.5 Сводная таблица результатов для среднего случая (Простой обмен с условием Айверсона) 5
1.6 График зависимости Сф+Мф (Простой обмен с условием Айверсона) 6
1.7 Алгоритм сортировки (Шейкерная сортировка) 6
1.8 Определение асимптотической сложности алгоритма (Шейкерная сортировка) 7
1.9 Сводная таблица результатов для среднего случая (Шейкерная сортировка) 9
1.10 График зависимости Сф+Мф (Шейкерная сортировка) 9
1.11 Анализ полученных результатов (Простой обмен и Шейкерная сортировка) 10
1.12 Алгоритм сортировки (Простое слияние) 11
1.13 Определение асимптотической сложности алгоритма (Простое слияние) 12
1.14 Сводная таблица результатов для среднего случая (Простое слияние) 13
1.15 График зависимости Сф+Мф (Простое слияние) 13
1.16 Анализ полученных результатов (Шейкерная сортировка и Простое слияние) 14
1.17 Определение емкостной сложности алгоритмов 14
2 ЗАДАНИЕ 2 16
2.1 Определение асимптотической сложности алгоритма (Простой обмен с условием Айверсона) 16
2.2 Сводная таблица результатов (Простой обмен с условием Айверсона) 16
2.3 Анализ зависимости (Простой обмен с условием Айверсона) 17
2.4 Определение асимптотической сложности алгоритма (Шейкерная

сортировка) 17
2.5 Сводная таблица результатов (Шейкерная сортировка) 18
2.6 Анализ зависимости (Шейкерная сортировка) 19
2.7 Определение асимптотической сложности алгоритма (Простое слияние) 20
2.8 Сводная таблица результатов (Простое слияние) 20
2.9 Анализ зависимости (Простое слияние) 21
2.10 Сравнение асимптотических сложностей алгоритмов 22
3 ВЫВОДЫ 23
4 СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 24

1 ЗАДАНИЕ 1

1.1 Условие задания и требования

Разработать три алгоритма сортировки, определенные вариантом. Провести анализ вычислительной и емкостной сложности алгоритма на массивах, заполненных случайно. Определить наиболее эффективный алгоритм.

Номер индивидуального варианта – 10%8+1 = 3

Вариант	Алгоритм простой сортировки	Алгоритм усовершенствованной сортировки	Алгоритм слияния
2	Простого обмена (пузырек) с условием Айверсона	Шейкерная с условием Айверсона	Простое слияние

1.2 Постановка задачи

Получить навыки по анализу вычислительной сложности нескольких алгоритмов сортировки и определение наиболее эффективного алгоритма

1.3 Алгоритм сортировки (Простой обмен с условием Айверсона)

```
2  long long* bubble_sort(int arr[], int n)
3  {
4      long long comp = 0, move = 0;
5
6      bool swapped;
7
8      for (int i = 0; i < n - 1; i++)
9      {
10         swapped = false;
11         for (int j = 0; j < n - i - 1; j++)
12         {
13             comp++;
14             if (arr[j] > arr[j + 1])
15             {
16                 move++;
17                 swap(arr[j], arr[j + 1]); //меняем элементы местами
18                 swapped = true;
```

```
19
20              }
21
22          }
23              // Если на данной иттерации элементы уже отсортированы, завершаем
    сортировку
24          if (swapped == false)
25          {
26              break;
27          }
28      }
29
30      long long* MC = new long long[2] { move, comp };
31
32      return MC;
33
34  }
```

1.4 Определение асимптотической сложности алгоритма (Простой обмен с условием Айверсона)

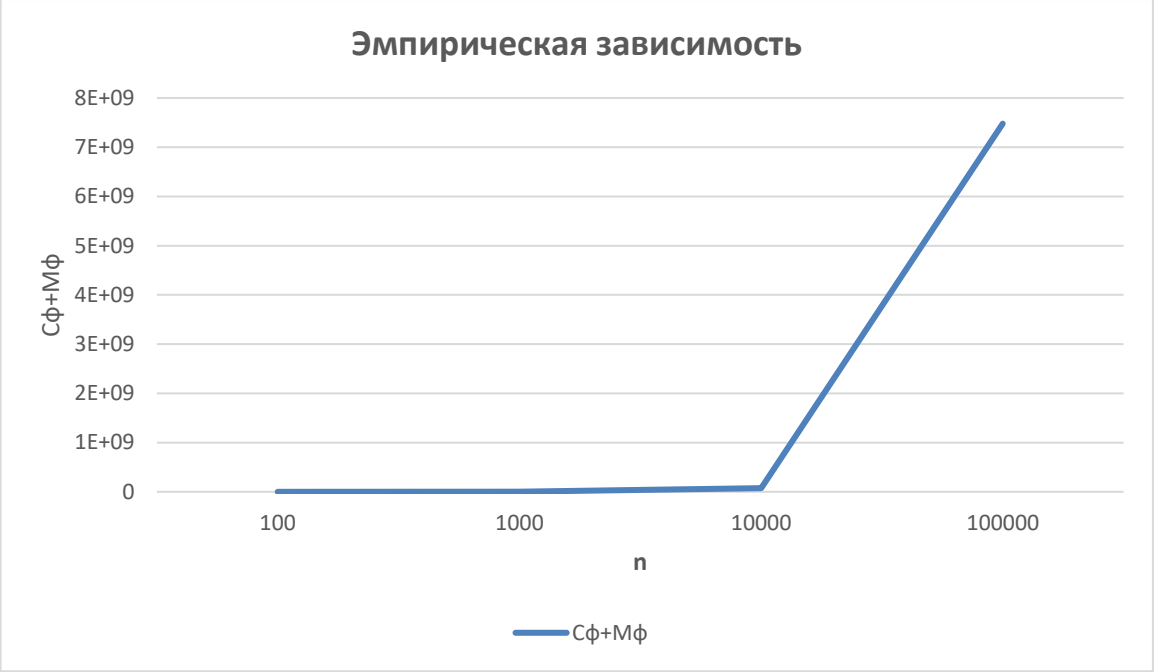
Алгоритм пузырьковой сортировки с условием Айверсона имеет такую же асимптотическую сложность, как и классическая версия. Его емкостная сложность можно оценить как $O(n^2)$, где n - количество элементов для сортировки. Даже с условием Айверсона, который позволяет оптимизировать алгоритм и не проверять уже отсортированные элементы, алгоритм все еще имеет квадратичную сложность.

1.5 Сводная таблица результатов для среднего случая (Простой обмен с условием Айверсона)

n	T, микросекунд	Тэп = f(C+M) - функция	Сф + Мф - количество
100	123	$O(n^2)$	4947 + 2508 = 7455
1000	21959	$O(n^2)$	498510 + 235084 = 733594

10000	1563505	$O(n^2)$	49984269 + 24911686 = 74895955
100000	88145219	$O(n^2)$	4999404510 + 2479297401 = 7478701911

1.6 График зависимости Сф+Мф (Простой обмен с условием Айверсона)



1.7 Алгоритм сортировки (Шейкерная сортировка с условием Айверсона)

```
2   long long* cocktail_sort(int arr[], int n)
3   {
4       long long numCompares = 0;
5       long long numSwaps = 0;
6
7       int left = 0;                   // Инициализация левой границы
8       int right = n - 1;              // Инициализация правой границы
9       bool swapped = true;              // Флаг, показывающий, были ли произведены обмены элементов
10
11      while (swapped) {                 // Цикл, продолжающийся до тех пор, пока не будет выполнено условие swapped = false
12          // Сортировка слева направо
13          swapped = false;                // Обнуление флага swapped перед каждой итерацией
```

```
14
15          for (int i = left; i < right; i++) // Цикл прохода по элементам массива
    слева направо
16          {
17              numCompares++;
18              if (arr[i] > arr[i + 1])
19              {
20                  numSwaps++;
21                  swap(arr[i], arr[i + 1]);     // Обмен элементов местами
22                  swapped = true;   // Установка флага swapped = true, так как был
    выполнен обмен
23
24              }
25          }
26          right--;                          // Уменьшение правой границы на 1
27
28
29
30          // Сортировка справа налево
31          swapped = false;               // Обнуление флага swapped перед каждой
    итерацией
32          for (int i = right; i > left; i--) // Цикл прохода по элементам массива
    справа налево
33          {
34              numCompares++;
35              if (arr[i - 1] > arr[i]) // Если текущий элемент меньше предыдущего
36              {
37                  numSwaps++;
38                  swap(arr[i - 1], arr[i]);     // Обмен элементов местами
39                  swapped = true;               // Установка флага swapped = true,
    так как был выполнен обмен
40              }
41          }
42          left++;                     // Увеличение левой границы на 1
43
44      }
45
46      long long* MC = new long long[2] {numSwaps, numCompares};
47      return MC;
48  }
```

1.8 Определение асимптотической сложности алгоритма (Шейкерная сортировка с условием Айверсона)

Асимптотическая сложность Шейкерной сортировки с условием Айверсона, как и обычной Шейкерной сортировки, составляет $O(n^2)$ в худшем случае и $O(n)$ в лучшем случае.

В худшем случае это происходит, когда массив уже отсортирован в обратном порядке.

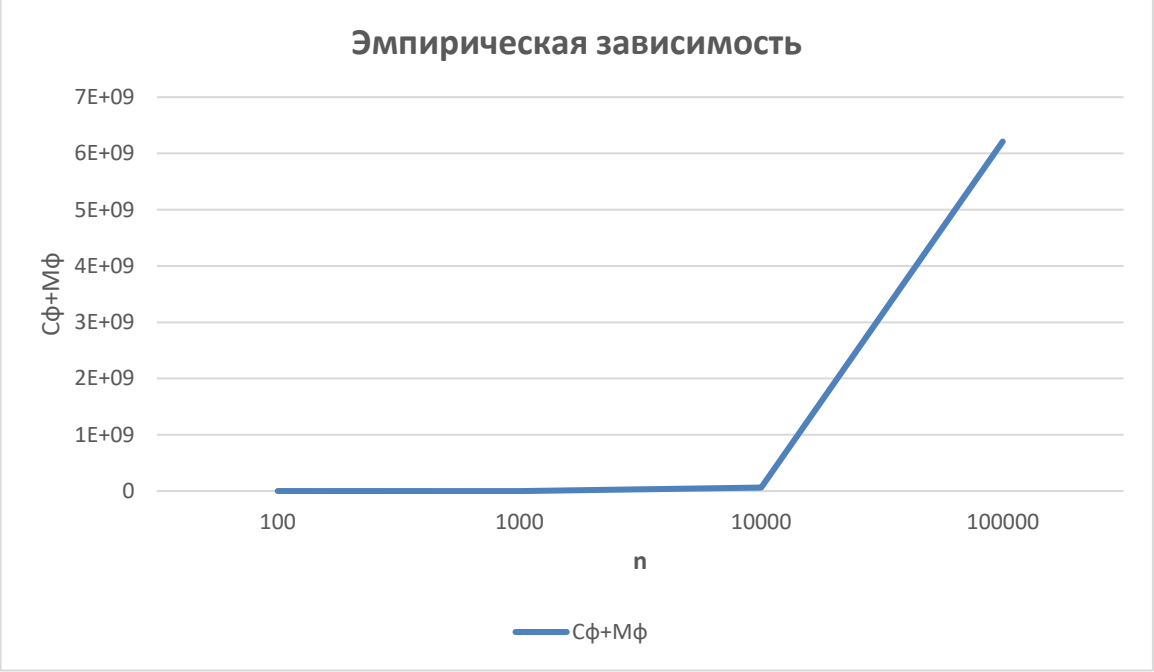
В лучшем случае, когда массив уже отсортирован, и при первом проходе внутреннего цикла не произойдет ни одного обмена, алгоритм завершится с линейной сложностью, т.е. выполнит всего n сравнений.

Средняя сложность шейкерной сортировки с условием Айверсона составляет $O(n^2)$.

1.9 Сводная таблица результатов для среднего случая (Шейкерная сортировка с условием Айверсона)

n	T, микросекунд	Тэп = f(C+M) - функция	Сф + Мф - количество
100	164	$O(n^2)$	3822 + 2462 = 6284
1000	19692	$O(n^2)$	374750 + 242550 = 617300
10000	1935273	$O(n^2)$	37726419 + 24752256 = 62478675
100000	177705539	$O(n^2)$	3740958890 + 2471348011 = 6212306901

1.10 График зависимости Сф+Мф (Шейкерная сортировка с условием Айверсона)



1.11 Анализ полученных результатов (Простой обмен и Шейкерная сортировка с условие Айверсона)

Из графика выше видно, что Шейкерная сортировка с условием Айверсона имеет асимптотическую сложность $O(n^2)$, но на практике она работает немного быстрее пузырьковой сортировки с условием Айверсона благодаря двусторонней обработке массива.

1.12 Алгоритм сортировки (Простое слияние)

```
2   void merge_sort(int* m, int low, int high) //указатель на массив, индекс начала
    сортировки, индекс конца сортировки
3   {
4       if (low >= high)
5           return;
6
7       int mid = (low + high) / 2; //вычисление индекса середины массива.
8
9       merge_sort(m, low, mid); //запуск сортировки для левой части массива.
10
11      merge_sort(m, mid + 1, high); //запуск сортировки для правой части массива.
12
13      int* aux = new int[high - low + 1]; //создание временного массива, который
    будет использован для слияния двух.
14
15      //инициализация трех переменных: указатели на текущие элементы массива
16      // - левой, правой части и k - указатель на текущий элемент временного массива.
17      int i = low;
18      int j = mid + 1;
19      int k = 0;
20
21      while (i <= mid && j <= high) //если значение элемента левой половины меньше
    или равно значению
22          //элемента правой половины, то это значение помещается в aux[k] и увели-
    чивается i.
23      {
24          if (m[i] <= m[j])
25          {
26              aux[k] = m[i];
27              i++;
28          }
29          else //в противном случае значение элемента
30              //правой половины помещается в aux[k] и увеличивается j.
31          {
32              aux[k] = m[j];
33              j++;
34          }
35          k++;
36      }
37      while (i <= mid) //Если в левой половине массива остались
38          //непроверенные элементы, то они добавляются в конец временного массива
    aux.
39      {
40          aux[k] = m[i];
41          i++;
42          k++;
43      }
44      while (j <= high) //Если в правой половине массива остались
45          //непроверенные элементы, то они также добавляются в конец временного
    массива aux.
```

```
46      {
47          aux[k] = m[j];
48          j++;
49          k++;
50      }
51      for (int i = low, k = 0; i <= high; i++, k++) //Значения элементов временного
    массива aux
52          //копируются обратно в исходный массив m от low до high.
53      {
54          m[i] = aux[k];
55      }
56
57      delete[] aux;
58  }
```

1.13 Определение асимптотической сложности алгоритма (Простое слияние)

Асимптотическая сложность алгоритма простого слияния - O(n*log(n)), где n - длина массива.

В лучшем случае алгоритм простого слияния имеет сложность O(n*log(n)).

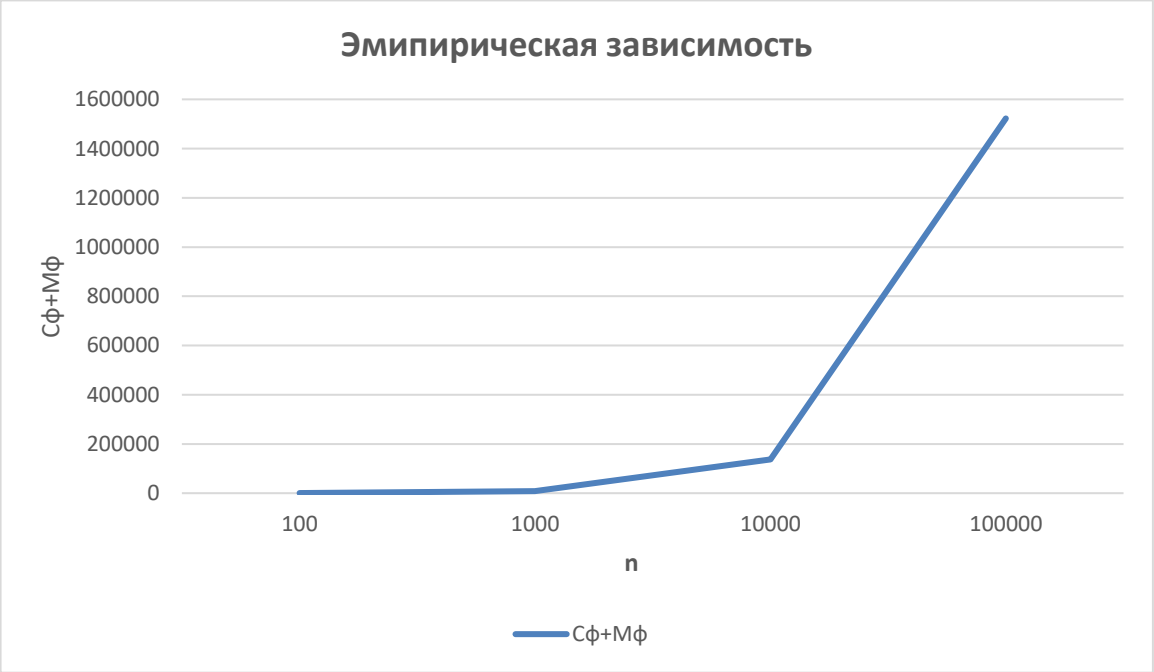
В среднем и в худшем случае алгоритм имеет ту же асимптотику O(n*log(n)).

Сортировка простым слиянием делит массив пополам, затем каждую половину рекурсивно сортирует и соединяет результат. На каждом уровне рекурсии выполняется O(n) операций для объединения двух уже отсортированных половин. Значит, общая сложность алгоритма на каждом уровне рекурсии - O(n). Деление массива происходит до тех пор, пока длина сортируемого массива не достигнет 1. Всего уровней рекурсии будет log(n). Таким образом, общая сложность алгоритма - O(n*log(n)).

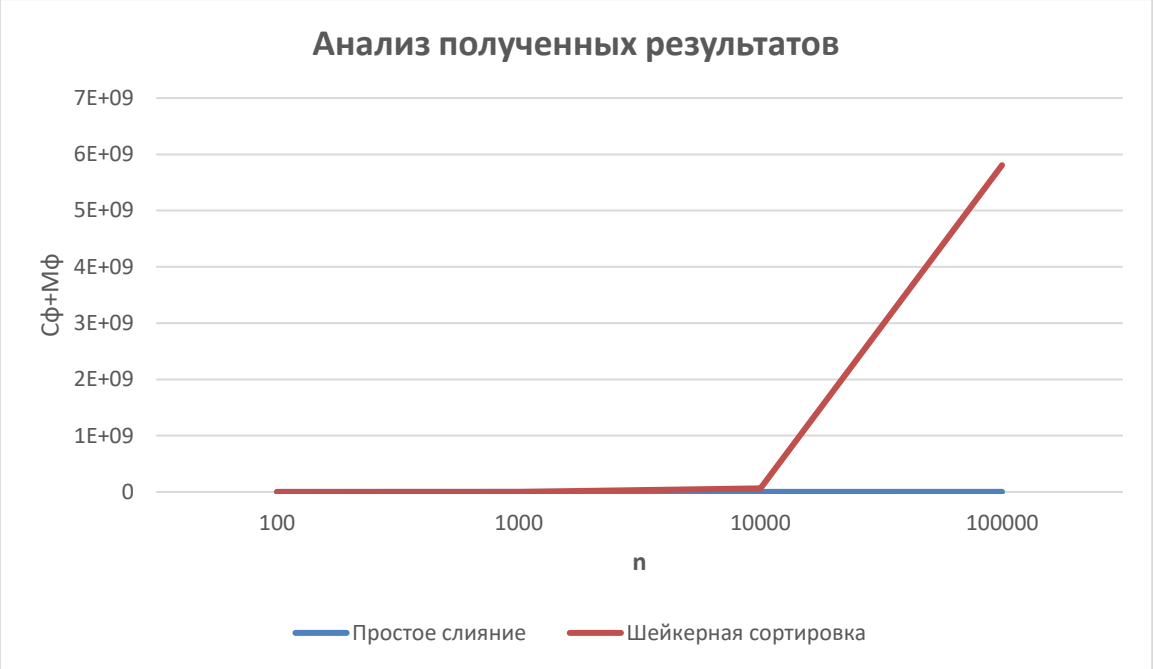
1.14 Сводная таблица результатов для среднего случая (Простое слияние)

n	T, микросекунд	Тэп = f(C+M) -функция	Сф + Мф - количество
100	0	O(n*log(n))	374+215= 589
1000	0	O(n*log(n))	5893+3096= 8 989
10000	7	O(n*log(n))	94499+42171= 136 670
100000	65	O(n*log(n))	1105210+ 417186= 1 522 396

1.15 График зависимости Сф+Мф (Простое слияние)



1.16 Анализ полученных результатов (Шейкерная сортировка с условием Айверсона и Простое слияние)



Шейкерная сортировка с условием Айверсона и сортировка Простым слиянием имеют различные подходы к сортировке массивов и имеют свои преимущества и недостатки.

Шейкерная сортировка с условием Айверсона более эффективна на коротких массивах, так как имеет лучший производительность в лучшем случае. Сортировка простым слиянием более эффективна на более длинных массивах, так как ее скорость не зависит от исходного порядка данных.

1.17 Определение емкостной сложности алгоритмов

Простой обмен с условием Айверсона:

Емкостная сложность сортировки Простого обмена зависит от размера сортируемого массива и может быть оценена как O(1), так как алгоритм не использует дополнительную память для хранения элементов массива, кроме той, которая необходима для самого массива.

Шейкерная сортировка с условием Айверсона:

Емкостная сложность шейкерной сортировки зависит от размера сортируемого массива и может быть оценена как O(1), так как алгоритм не использует дополнительную память для хранения элементов массива, кроме той, которая необходима для самого массива.

Простое слияние:

В алгоритме простого слияния требуется дополнительная память для создания временных массивов при слиянии подмассивов. Общий объем дополнительной памяти, необходимый для алгоритма, зависит от размера сортируемого массива и может быть оценен как O(n), где n — это количество элементов в массиве.

2 ЗАДАНИЕ 2

2.1 Определение асимптотической сложности алгоритма (Простой обмен с условием Айверсона)

Лучший случай:

В наилучшем случае, когда массив уже отсортирован, число инверсий равно 0, и пузырьковая сортировка производит только один проход по массиву для проверки, что элементы уже упорядочены и ни одного обмена не происходит. Следовательно, асимптотическая сложность в лучшем случае будет O(n).

Худший случай:

В наихудшем случае, когда массив полностью обратно упорядочен и каждая пара элементов находится в обратном порядке, количество инверсий равно n(n-1)/2. В таком случае пузырьковая сортировка производит n-1 проходов по массиву, и на каждом проходе происходит обмен соседних элементов. Асимптотическая сложность в наихудшем случае будет $O(n^2)$.

2.2 Сводная таблица результатов (Простой обмен с условием Айверсона)

Лучший случай:

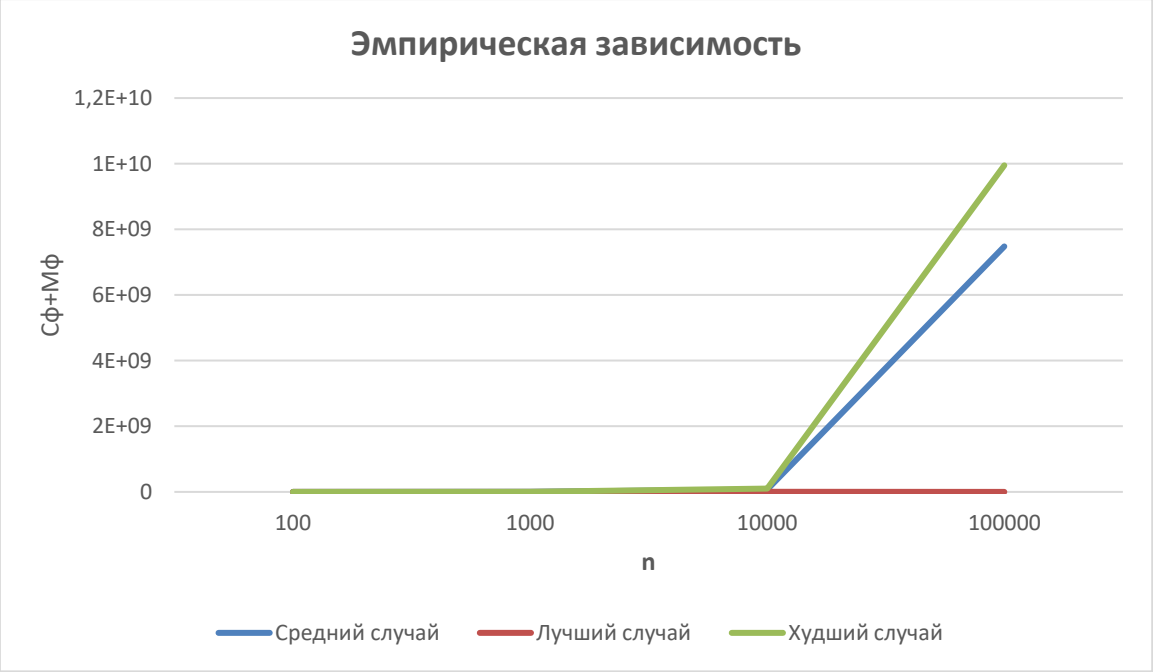
n	T, микросекунд	Тэп = f(C+M) -функция	Сф + Мф - количество
100	3	O(n)	99 + 0 = 99
1000	7	O(n)	999 + 0 + 999
10000	44	O(n)	9999 + 0 = 9999
100000	734	O(n)	99999 + 0 = 99999

Худший случай:

n	T, микросекунд	Тэп = f(C+M) -функция	Сф + Мф - количество
100	339	$O(n^2)$	4950 + 4905 = 9855
1000	19277	$O(n^2)$	499472 + 494625 = 994097

10000	2212137	$O(n^2)$	49990995 + 49495799 = 99486794
100000	132896218	$O(n^2)$	4999473224 + 4949954576 = 9949427800

2.3 Анализ зависимости (Простой обмен с условием Айверсона)



Сортировка Простого обмена с добавлением условия Айверсона уменьшает количество сравнений и перемещений при сортировке массива. Это связано с тем, что в условии Айверсона пропускаются элементы, которые уже находятся на своем месте, что позволяет уменьшить количество операций сравнения.

Из представленных графиков видно, что алгоритм Простого обмена с условием Айверсона наиболее эффективен в случае с уже отсортированным массивом данных.

2.4 Определение асимптотической сложности алгоритма (Шейкерная сортировка с условием Айверсона)

Лучший случай:

Для лучшего случая, когда входной массив уже отсортирован, сложность шейкерной сортировки с условием Айверсона будет составлять O(n), так как в этом случае алгоритм не будет проходить по массиву в

обратном направлении и все элементы будут перемещаться только в одну сторону на каждой итерации внешнего цикла.

Худший случай:

Однако, в худшем случае, когда массив расположен в обратном порядке, шейкерная сортировка будет иметь $O(n^2)$ асимптотическую сложность, так как каждый элемент необходимо переместить до своей окончательной позиции.

2.5 Сводная таблица результатов (Шейкерная сортировка с условием Айверсона)

Лучший случай:

n	T, микросекунд	Тэп = f(C+M) -функция	Сф + Мф - количество
100	6	O(n)	197+0= 197
1000	24	O(n)	1997+0= 1997
10000	129	O(n)	19997+0= 19997
100000	1175	O(n)	199997+0= 199997

Худший случай:

n	T, микросекунд	Тэп = f(C+M) -функция	Сф + Мф - количество
100	360	$O(n^2)$	4950+4903= 9853
1000	29302	$O(n^2)$	499485+494364= 993849
10000	3187994	$O(n^2)$	49994985 + 49494456 = 99489441
100000	306197728	$O(n^2)$	4999782090 + 4949942846 = 9949724936

2.6 Анализ зависимости (Шейкерная сортировка с условием Айверсона)

Из представленных графиков видно, что Шейкерная сортировка с условием Айверсона наиболее эффективна в случае с уже отсортированным массивом данных.

2.7 Определение асимптотической сложности алгоритма (Простое слияние)

Лучший случай:

В лучшем случае, когда массив уже отсортирован, алгоритм Простого слияния также работает за время O(n*log n), поскольку каждая пара элементов уже отсортирована и может быть соединена за одну операцию.

Худший случай:

Худший случай возникает, когда массив для сортировки уже отсортирован в обратном порядке или содержит множество повторяющихся элементов. В этом случае в функции сортировки при каждом вызове массив будет делиться на две половины до тех пор, пока каждый элемент не станет отдельной подмассивом. Затем при каждом вызове функции для слияния двух отсортированных подмассивов потребуется O(n) операций, так как оба подмассива будут полностью заполнены. Таким образом, общее время выполнения в худшем случае будет пропорционально n*log(n).

2.8 Сводная таблица результатов (Простое слияние)

Лучший случай:

n	T, микросекунд	Тэп = f(C+M) -функция	Сф + Мф - количество
100	0	O(n*log(n))	369+ 180= 549
1000	0	O(n*log(n))	6025+ 3193= 9 218
10000	5	O(n*log(n))	94539+ 42261= 136 800
100000	33	O(n*log(n))	1102679+ 414716= 1517395

Худший случай:

n	T, микросекунд	Тэп = f(C+M) -функция	Сф + Мф - количество
100	0	O(n*log(n))	383+ 213= 596
1000	1	O(n*log(n))	6004+ 3216= 9 220
10000	8	O(n*log(n))	94377+ 41757= 136 134
100000	44	O(n*log(n))	1101411+ 412858= 1 514 269

2.9 Анализ зависимости (Простое слияние)

Из представленных графиков видно, что алгоритм сортировки Простым слиянием показывает себя одинаково эффективным во всех трех случаях.

2.10 Сравнение асимптотических сложностей алгоритмов

Название алгоритма	Асимптотическая сложность алгоритма			
	Наихудший случай	Наилучший случай	Средний случай	Емкостная сложность
Простого обмена с условием Айверсона	$O(n^2)$	O(n)	$O(n^2)$	O(1)
Шейкерная с условием Айверсона	$O(n^2)$	O(n)	$O(n^2)$	O(1)
Простого слияния	O(n*log(n))	O(n*log(n))	O(n*log(n))	O(n)

3 ВЫВОДЫ

В ходе выполнения данной практической работы были изучены и реализованы три алгоритма сортировки: Простой обмен с условием Айверсона, Шейкерная с условием Айверсона, Простое слияние.

Алгоритм Простого обмена с условием Айверсона улучшает обычный алгоритм Простого обмена, тем самым ускоряя сортировку. Основная идея заключается в том, чтобы при каждом проходе по массиву менять местами элементы только в тех парах, где нарушено условие упорядоченности.

Шейкерная сортировка с условием Айверсона представляет собой улучшенную версию сортировки пузырьком, где происходит проход по массиву справа налево, а затем слева направо, что уменьшает количество итераций и улучшает время работы алгоритма.

Сортировка слиянием, с другой стороны, имеет гарантированную асимптотическую сложность O(nlog(n)) для всех случаев, включая лучший, худший и средний. Она превосходит сортировку пузырьком и шейкерную сортировку в эффективности на больших наборах данных, но может потребовать дополнительную память для хранения временных массивов. Является более эффективным для работы с большим количеством элементов.

В результате проведенной работы было выяснено, что все три алгоритма дают правильный результат сортировки, но каждый имеет свои преимущества и недостатки в зависимости от особенностей массива и количества элементов. Поэтому выбор алгоритма для сортировки данных должен быть сделан с учетом конкретных условий и задач.

4 СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Материалы по дисциплине (Сорокин А.В.).

2. Скворцова Л.А., Гусев К.В., Трушин С.М., Филатов А.С. Учебно-методическое пособие СиАОД.