Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Сортировки»**

**Выполнил**:

студент группы 3824Б1ПМ1

Мееров М.И.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2024

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

В рамках лабораторной работы требовалось реализовать на языке C 4 вида сортировок: пузырьковую, расчёской, слиянием и поразрядную (LSD) для случайных массивов типа данных float. Оценить зависимость времени работы программы и количества операций, производимых ей, от размера массива.

# Метод решения

Первый алгоритм – сортировка пузырьком. На i-ом шаге рассматриваются все пары последовательных элементов от первого, до (n-i)-го. В случае, если элемент с наименьшим индексом больше следующего, то элементы пары меняют местами. Таким образом, на i-ом шаге, на (n-i)-ую позицию становится максимальным из всех предыдущих элементов, а, значит, к тому моменту, когда не будет проведено ни одного обмена или к концу (n-1)-го шага, массив станет отсортирован.

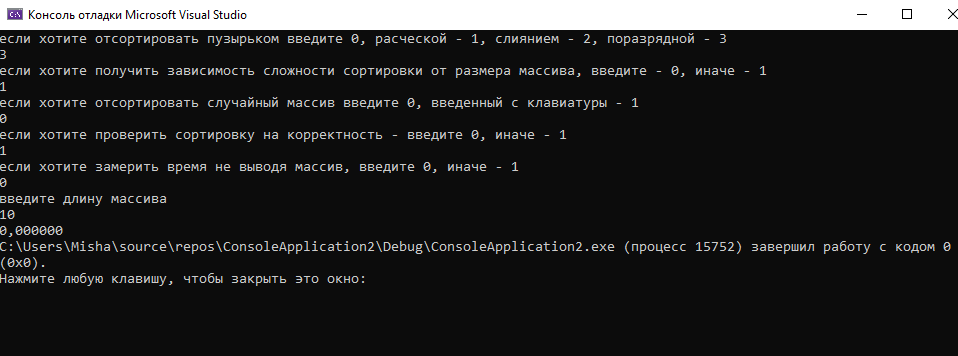
Второй алгоритм – сортировка расчёской. На i-ом шаге рассматриваются все пары элементов с шагом s от первого, до (n-s)-го. В случае, если элемент с наименьшим индексом больше второго элемента пары, то элементы пары меняют местами. Дальше шаг s уменьшается в 1.27 раз. Алгоритм выполняется до тех пор, пока . Дальше запускается сортировка пузырьком

Третий алгоритм – сортировка слиянием. На i-ом шаге происходит слияние пар подряд идущих подмассивов размера (если размер последнего подмассива меньше чем , то все-равно происходит слияние). Алгоритм слияния двух отсортированных подмассивов в третий происходит так. На k-ом шаге сравниваются i-ый элемент первого массива и j-ый второго. На k-ую позицию нового массива ставится максимальных из них. Индекс в этом массиве увеличивается. Таким образом, на каждом шаге массив состоит из отсортированных подмассивов, размером в 2 раза больше, чем предыдущие.

Четвертый алгоритм – поразрядная сортировка. На i-ом шаге происходит сортировка подсчетом по i-му с конца байту элементов. Подсчитывается количество элементов с i-ым байтом от 0 до 256 и в новый массив элементы ставятся в порядке возрастания i-го байта так, чтобы при одинаковом i-ом байте, порядок элементов не меняется. Таким образом, получается, что сначала идут положительные элементы в порядке возрастания, потом отрицательные в порядке убывания. Дальше происходит перепаковка элементов, уже в нужном порядке.

# Руководство пользователя

1. Если требуется отсортировать пузырьком - введите 0, расческой - 1, слиянием - 2, поразрядной – 3
2. Если необходимо получить зависимость сложности сортировки от размера массива, введите - 0, иначе – 1
3. Если ввели 0, то:
   1. Введите кол-во точек, для которых исследуется зависимость
   2. Если хотите ввести точки с клавиатуры, введите 0, иначе – 1
      1. Если ввели 0, то введите точки
      2. Если ввели 1, то введи 2 числа – диапазон, в котором эти точки будут располагаться.
4. Если ввели 1, то:
   1. Если хотите отсортировать случайный массив, введите 0, введённый с клавиатуры – 1.
   2. Если хотите проверить сортировку на корректность - введите 0, иначе – 1.
   3. Если хотите замерить время, не выводя массив, введите 0, иначе – 1.
   4. Введите длину массива n.
      1. Если на шаге 4.1. ввели 1, то введите массив из n элементов.



# Описание программной реализации

Функция bubble\_sort() получает на вход число элементов и указатель на массив и сортирует его пузырьком.

Функция bubble\_sort\_o() получает на вход число элементов и массив и начальное кол-во операций, сортирует массив пузырьком и выводит количество проведенных операций.

Функция comb\_sort() получает на вход число элементов и указатель на массив и сортирует его расчёской.

Функция comb\_sort\_o() получает на вход число элементов и указатель на массив и начальное кол-во операций, сортирует массив расчёской и выводит количество проведенных операций.

Функция merge\_sort() получает на вход число элементов и указатель на массив и сортирует его слиянием.

Функция merge\_sort() получает на вход число элементов и указатель на массив и сортирует его слиянием.

Функция sl() получает на вход два указателя на массивы, индексы начала подмассивов первого массива, для которых надо произвести слияние и размер последнего подмассива. Функция производит слияние этих подмассивов во второй массив.

Функция merge\_sort\_o() получает на вход число элементов, указатель на массив и начальное кол-во операций, сортирует массив слиянием и выводит количество проведенных операций.

Функция sl\_o() получает на вход два указателя на массивы, индексы начала подмассивов первого массива, для которых надо произвести слияние, размер последнего подмассива и начальное кол-во операций. Функция производит слияние этих подмассивов во второй массив. И возвращает количество операций.

Функция radix\_sort() получает на вход число элементов и указатель на массив и сортирует его поразрядно.

Функция fcount() получает на вход указатель на массив, размер одного элемента, смещение, массив для подсчета и размер массив и производит сортировку подсчетом.

Функция radix\_sort\_o() получает на вход число элементов, указатель на массив и кол-во операций, сортирует его поразрядно и выводит кол-во операций.

Функция fcount\_o() получает на вход указатель на массив, размер одного элемента, смещение, массив для подсчета, размер массив и кол-во операций, производит сортировку подсчетом и возвращает кол-во операций.

Функция start() получает на вход информацию о том какую сортировку надо запустить, нужна ли проверка, указатель на массив, его длину и возвращает время работы сортировки.

Функция proverka() получает на вход указатель на массив и его длину и проверяет корректность работы сортировки. В случае ошибки печатает “ошибка сортировки”

Функция randmas() получает на вход указатель на массив и его длину и генерирует случайный массив

Функция randpoint() получает на вход указатель на массив, его длину и диапазон, в котором генерировать числа и генерирует случайный массив элементов типа int в данном диапазоне.

Функция main() запускает функции выше, (порядок согласно руководству пользователю) и возвращает 0.

# Подтверждение корректности

Для подтверждения корректности в программе существует функция проверка. Она сравнивает все пары соседних элементов, и если первый элемент больше второго, то выводит “ошибка сортировки”.

Была проведена проверка сортировки случайных массивов для 200 случайных длин.

# Результаты экспериментов

Программа была запущенна 200-250 раз для случайных массивов. Построены графики зависимости отношения времени работы, число операций к предполагаемой сложности.

Для сортировки пузырьком, начиная с некоторого размера массива, и число операций и время пропорциональны квадрату размера массива (рис 1).

Рисунок 1. Зависимость времени и числа операций от размера массива для сортировки пузырьком

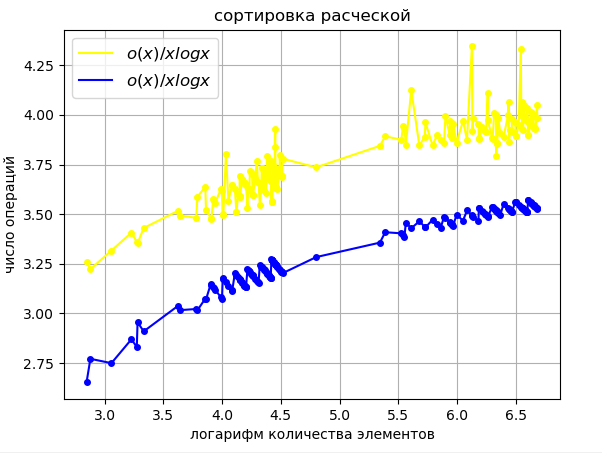
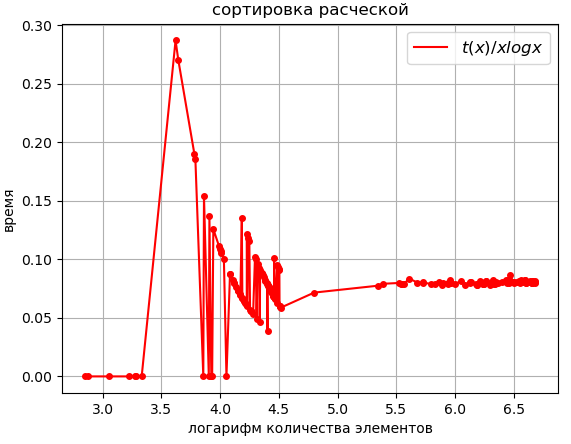
Для сортировки расческой, начиная с некоторого небольшого n, и число операций и время пропорциональны , где n - размера массива (рис 2). При этом число операций, если не учитывать сортировку пузырьком, более стабильно, поэтому, если в конце досортирововать массив с помощью сортировки вставками, то можно существенно ускорить сортировку расческой (рис 2).

Рисунок 2. Зависимость времени и числа операций от размера массива для сортировки расческой

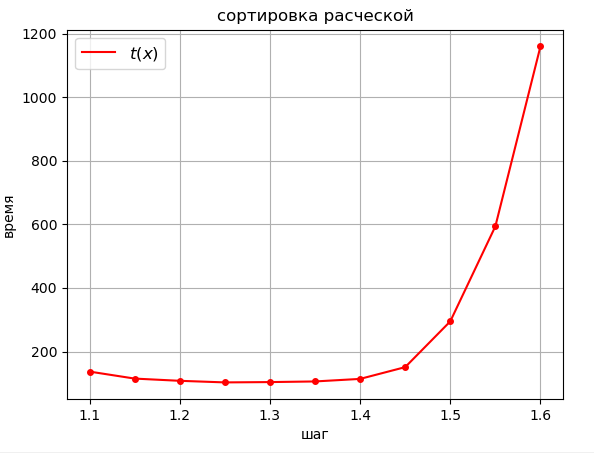
При этом сортировка расческой зависит от шага – того, с какой скоростью уменьшается расстояние, через которое сравнивают элементы. Из графиков (рис 3) видно, что оптимальный шаг – 1.27.

Рисунок 3. Зависимость времени работы сортировки расческой, от шага.

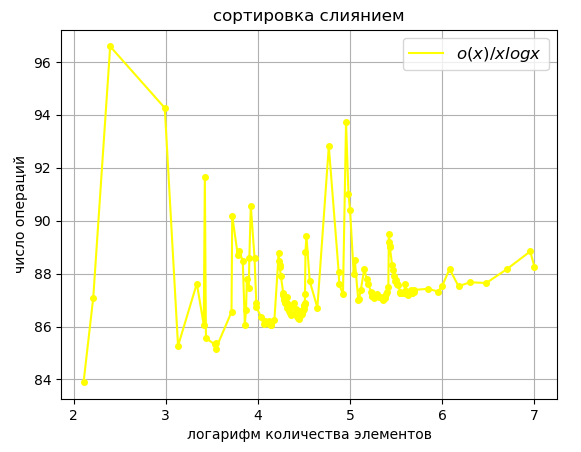
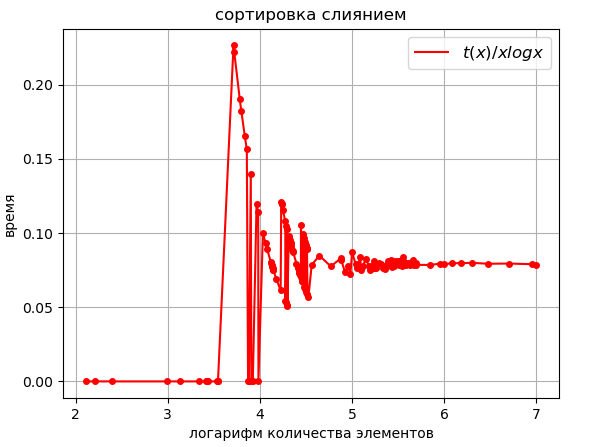
Для сортировки слияниями, начиная с некоторого небольшого n, и число операций и время пропорциональны , где n - размера массива (рис 4).

Рисунок 4. Зависимость времени и числа операций от размера массива для сортировки слиянием

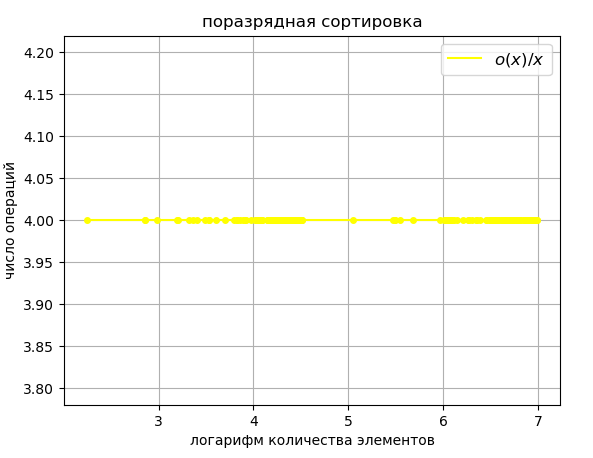
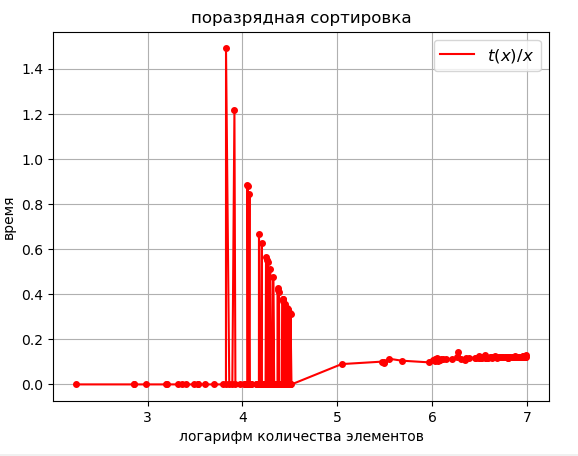
Для поразрядной сортировки, начиная с некоторого небольшого n, и число операций и время пропорциональны , где n - размера массива (рис 5).

Рисунок 5. Зависимость времени и числа операций от размера массива для поразрядной сортировки

# Заключение

В рамках лабораторной работы были реализованы 4 алгоритма сортировки: сортировка пузырьком, расческой, слиянием и поразрядная. Результатом является программа, которая позволяет запустить любой из этих алгоритмов, для массива, случайного или введенного с клавиатуры, проверить корректность полученного результата и изучить зависимость времени работы или количества проведенных операция сортировкой, от длины массива. Исследования этих алгоритмов позволяют прийти к выводу, что, в данной реализации, сортировка пузырьком работает за расческой и слияниями за причем, сортировка слияниями работает быстрее, а поразрядная - за

# Приложение

Поразрядная сортировка:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | int radix\_sort(int n, float\* a) {  ...  int q = 0, f = 0;  b = (float\*)malloc(n \* sizeof(float));  memset(b, 0.0, n\*sizeof(float));  count = (int\*)malloc(256 \* sizeof(int));  for (int i = 0; i < sizeof(unsigned int); i++) {  fcount((unsigned char\*)a, sizeof(unsigned int), i, count, n);  for (int j = 0; j < n; j++) {  b[count[\*(((unsigned char\*)a)+j\*sizeof(float)+i)]++] =a[j];  }  c = a;  a = b;  b = c;  }  for (int i = 0; i < n; i++) {  if (a[i] < 0 && f == 0) {  q = i;  f = 1;  }  b[i] = a[i];  }  for (int i = 0; i < n - q; i++) {  a[i] = b[n - i - 1];  }  for (int i = n - q; i < n; i++) {  a[i] = b[i - (n - q)];  }  free(b);  free(count);  return 0;  } |

Сортировка слиянием:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **int** **merge\_sort**(**int** n, **float**\* a) {  ...  **for** (r = **1**; r < n; r \*= **2**) {  **for** (i = **0**; i < n / r - **1**; i = i + **2**) {  sl(a, b, i \* r, (i + **1**) \* r, r);  }  **if** ((n - i \* r - **1**) / r == **1**) {  sl(a, b, n - n % r - r, n - n % r, n % r);  }  **else** {  **for** (**int** j = **0**; j < n - i \* r; j++) {  b[j + i \* r] = a[j + i \* r];  }  }  c = a;  a = b;  b = c;  k += **1**;  }  **if** ((k % **2**) == **1**) {  **for** (i = **0**; i < n; i++) {b[i] = a[i];}  free(a);  }  **else** { free(b); }  **return** **0**;  } |

Сортировка расческой:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | **int** **comb\_sort**(**int** n, **float**\* a) {  **float** s;  **for** (**int** step = (**int**) (n / **1.27**); step > **1**; step = (**int**) (step/**1.27**)){  **for** (**int** j = **0**; j < n - step; j++) {  **if** (a[j] > a[j + step]) {  s = a[j + step];  a[j + step] = a[j];  a[j] = s;  }  }  }  bubble\_sort(n, a);  **return** **0**;  } |

Сортировка пузырьком:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **int** **bubble\_sort**(**int** n, **float**\* a) {  **int** flag;  **float** s;  **for** (**int** i = **1**; i < n; i++) {  flag = **0**;  **for** (**int** j = **0**; j < (n - i); j++) {  **if** (a[j] > a[j + **1**]) {  s = a[j];  a[j] = a[j + **1**];  a[j + **1**] = s;  flag = **1**;  }  }  **if** (flag == **0**) {**return** **0**;}  }  **return** **0**;  } |