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 3823Б1-ПМ1

Бабадаев Расул Б.

**Проверил**:

преподаватель каф. ВВСП,

Волокитин В.Д.

Нижний Новгород

2023

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 6](#_Toc26962564)

[Описание программной реализации 7](#_Toc26962565)

[Подтверждение корректности 8](#_Toc26962566)

[Результаты экспериментов 9](#_Toc26962567)

[Заключение 12](#_Toc26962568)

Cписок используемой литературы…………………………………………………………………………………………………13

[Приложение 14](#_Toc26962569)

# Постановка задачи

1. Требовалось реализовать 4 вида различных видов сортировок массивов чисел на языке С тип данных double: сортировка пузырек, сортировка расческой, сортировка слиянием и поразрядная сортировка (LCD)

2. Проверить сортировки на корректность

3. Код программ загрузить в указанную репозиторию

4. Провести эксперименты с замером времени работы(в тактах) указанных сортировок и построить графики

# Метод решения

Для решения поставленной задачи необходимо реализовать 4 вида сортировок. Приведу описание их алгоритмов

**Пузырек**  
1. Начинаем со стартового индекса массива. Обозначим его как i и установим его значение в 0.  
2. Сравниваем элементы с индексами i и i+1. Если элемент с индексом i больше элемента с индексом i+1, меняем их местами.  
3. Переходим к следующей паре соседних элементов, увеличивая значение i на 1. Если достигнут конец массива, переходим к следующему шагу. В противном случае возвращаемся к шагу 2.  
4. Повторяем шаги 2 и 3 до тех пор, пока не произойдет полного просмотра массива без необходимости совершать обмены элементов.  
5. Если в процессе выполнения шагов 2 и 3 не было произведено ни одного обмена элементов, массив считается отсортированным, и алгоритм останавливается.  
Алгоритм сортировки пузырьком имеет временную сложность O(n^2), где n - количество элементов в массиве. Он является неэффективным для больших массивов, но может быть полезным для небольших наборов данных или в случаях, когда массив уже почти отсортирован.

**Расческа**  
1. Начинаем с начальной величины разрыва, равной размеру массива.  
2. Проводим проход по массиву, сравнивая элементы на расстоянии равном разрыву. Если элементы находятся в неправильном порядке, меняем их местами.  
3. Уменьшаем разрыв на некоторый коэффициент уменьшения.  
4. Повторяем шаги 2 и 3 до тех пор, пока разрыв не станет равным 1. Когда разрыв становится равным 1, проходим по массиву еще раз, используя обычный алгоритм сортировки пузырьком, чтобы завершить сортировку.  
5. Если в процессе выполнения шагов 2 и 3 не было произведено ни одного обмена элементов, массив считается отсортированным, и алгоритм останавливается.  
Алгоритм сортировки расческой имеет временную сложность в худшем случае O(n^2), но при правильном выборе коэффициента уменьшения разрыва может достигать временной сложности O(n \* log n). Этот алгоритм может быть эффективным для средних и больших массивов.

**Слиянием**  
1. Разбиваем исходный массив пополам, пока не останется массив из одного элемента или пустой массив. Это будет базовым случаем в рекурсивной функции.  
2. Рекурсивно сортируем левую и правую половины массива.  
3. Сливаем отсортированные половины массива в один отсортированный массив. Для этого создаем временный массив и сравниваем элементы из левой и правой половин массива, добавляя их в правильном порядке во временный массив.  
4. После завершения слияния, временный массив содержит упорядоченные элементы из исходного массива.  
5. Возвращаем отсортированный массив.  
Алгоритм сортировки слиянием имеет временную сложность O(n\*logn), где n - размер массива. Из-за своей эффективности и стабильности, он широко используется в практике для сортировки больших массивов данных. Однако, он требует дополнительной памяти для временного массива, что может быть недостатком при работе с большими объемами данных.

**Поразрядная**  
1. Начинаем с наименее значимого разряда чисел (самого правого разряда) и перебираем все разряды, двигаясь слева направо.  
2. Для каждого разряда выполняем сортировку стабильной сортировкой, например, сортировкой подсчетом или сортировкой с использованием очередей.  
3. После сортировки по текущему разряду, элементы сгруппированы в соответствии с их значениями в текущем разряде.  
4. Повторяем шаги 2 и 3 для следующего разряда, двигаясь от младших разрядов к более старшим.  
5. После завершения сортировки всех разрядов, получаем отсортированный массив.  
Важным моментом при поразрядной сортировке LSD является использование стабильной сортировки на каждом разряде, чтобы сохранить относительный порядок элементов с одинаковыми значениями в текущем разряде. Это гарантирует правильность сортировки на каждом этапе и окончательный корректный результат.  
Поразрядная

сортировка LSD обычно применяется к целым числам или строкам, где каждый символ представляет разряд. Этот алгоритм является стабильным и имеет временную сложность O(d\*n), где d - количество разрядов, а n - количество элементов в массиве.

# Руководство пользователя

Каждая сортировка представлена отдельным файлом. В консоле можно выбрать количество элементов массива и диапазон чисел. Массив будет заполняться случайными числами. Далее в зависимости от выбранной программы произойдет сортировка массива по возрастанию и укажется количество тактов процессора.

# Описание программной реализации

Каталог содержит 4 подкаталога с разными сортировками:

1. BubbleSort – содержит сортировку пузырьком
2. CombSort – содержит сортировку расческой
3. MergeSort – содержит сортировку слиянием
4. RadixSort – содержит поразрядную сортировку

Каждый подкаталог содержит файл с кодом \*.cpp.

В папке также доступен файл Otchet\_Babadaev\_Rasul.docx – личный файл с текстом отчета.

# Подтверждение корректности

Для подтверждения корректности в программе была использована сортировка qsort() из [#include](https://vk.com/im?sel=329777122&st=%23include) <stdlib.h>. Проверку на корректность прошли все сортировки с диапазоном чисел до 10000.

# Результаты экспериментов

Построим графики, используя формулу y = , где t – количество тактов, f(n) – предполагаемая сложность сортировки. Эксперименты проводились на устройствах с установленными на них Unix – системах (MacOS conoma 14.0, процессор Apple M1).

**cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC; // Вычисляем затраченное процессорное время  
printf("Количество тактов: %ld\n", end - start);** - замер тиков

Предполагаемые сложности сортировок: пузырьком – n^2, расческой – n\*logn,

Слиянием n\*logn, поразрядная – n.

1. Сортировка пузырьком

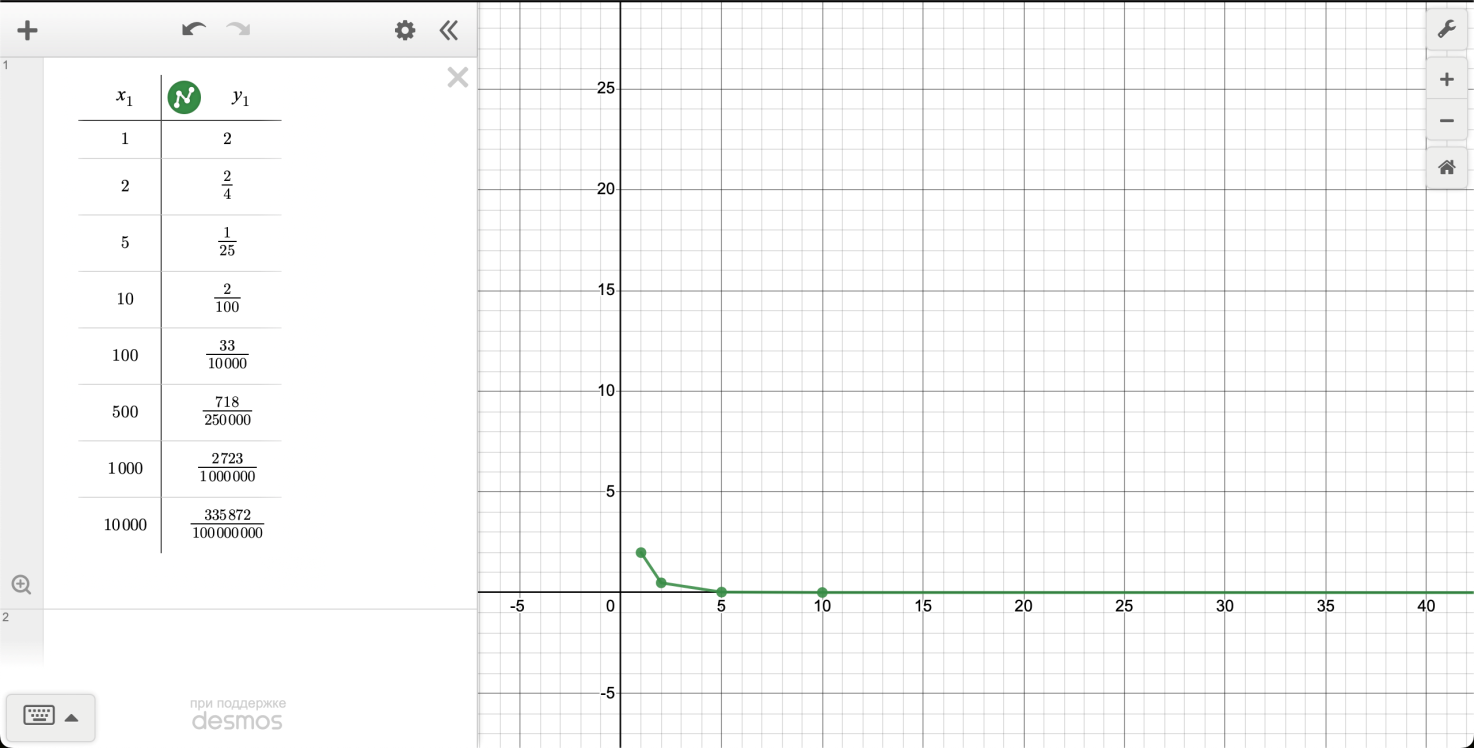


Рисунок 1

Из графика видно, что функция, начиная с некоторого n стала принимать константу, значит, предположение оказалось верно и сложность алгоритма O(n^2). [рисунок 1]

1. Сортировка расческой



Рисунок 2

Видно, что начиная с некоторого n график принимает константу, предположение верно, сложность O(nlog(n)). [рисунок 2]

1. Сортировка слиянием



Рисунок 3

Видно, что начиная с некоторого n график принимает константу, предположение верно, сложность O(nlog(n)). [рисунок 3]

1. Поразрядная сортировка

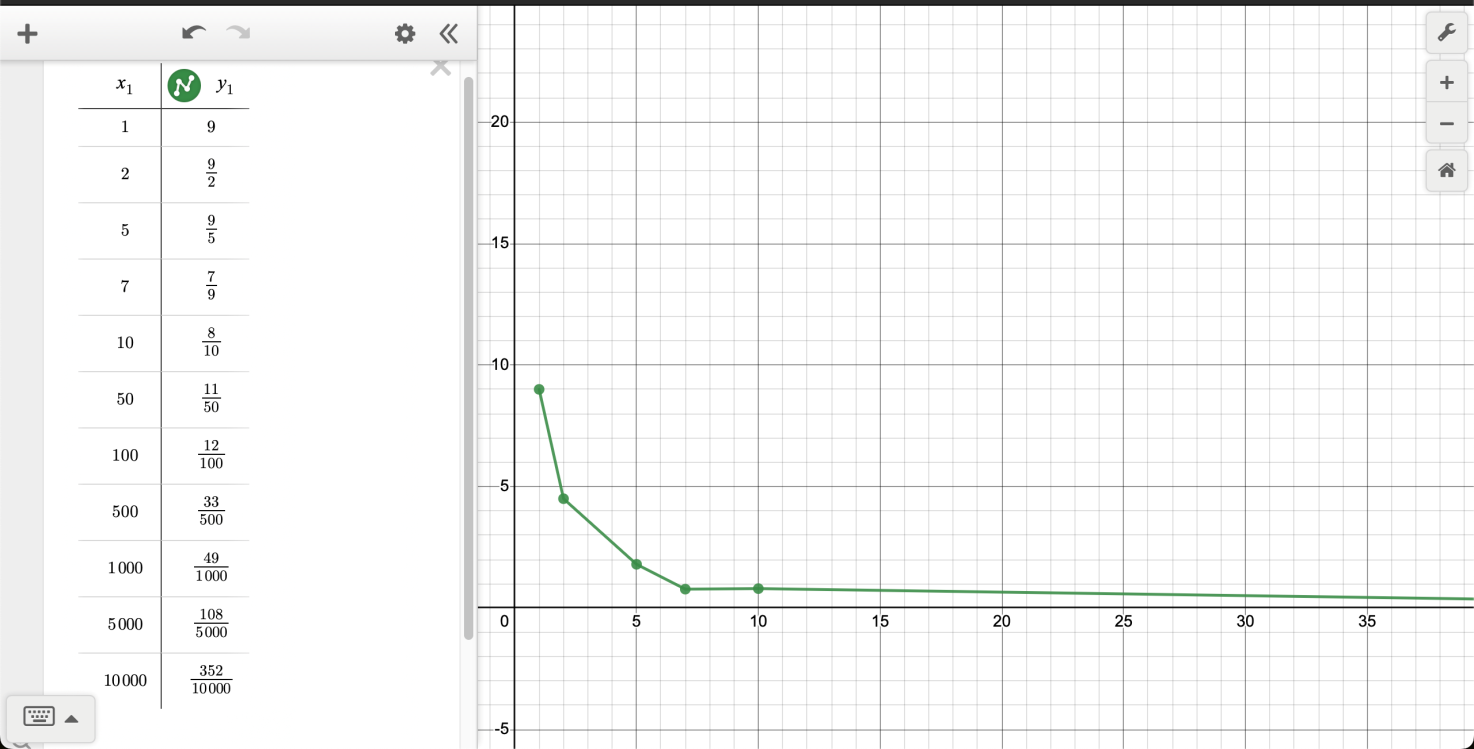


Рисунок 4

Видно, что начиная с некоторого n графика принимает константу, предположение верно, сложность O(n). [рисунок 4]

# Заключение

Реализованные алгоритмы были проверены на корректность и на временную сложность. В ходе исследований было выявлено:

* Самая быстрая сортировка для достаточно больших n – поразрядная
* Для небольших n выгоднее всего использовать простые алгоритмы(пузырек)
* Сортировка слиянием выгодна для больших массивов, но занимает много памяти.

**Список используемой литературы**

1. <https://ru.wikibooks.org/wiki/Язык_Си_в_примерах/Сортировка>
2. <https://www.geeksforgeeks.org/c-program-to-sort-an-array-in-ascending-order/>
3. <https://www.helloworld.ru/texts/comp/lang/c/c/h22.htm>
4. Язык программирования Си Брайан Керниган, Деннис Ритчи 3-е издание

# Приложение

[#include](https://vk.com/im?sel=329777122&st=%23include) <stdio.h>  
[#include](https://vk.com/im?sel=329777122&st=%23include) <string.h>  
[#include](https://vk.com/im?sel=329777122&st=%23include) <stdlib.h>  
[#include](https://vk.com/im?sel=329777122&st=%23include) <time.h>  
[#include](https://vk.com/im?sel=329777122&st=%23include) <stdint.h>  
[#define](https://vk.com/im?sel=329777122&st=%23define) SIZE 1000  
[#define](https://vk.com/im?sel=329777122&st=%23define) RANGE 100  
[#define](https://vk.com/im?sel=329777122&st=%23define) UCHAR\_MAX 255  
  
int compare(const void \*a, const void \*b){  
return (\*(double\*)a - \*(double\*)b);  
}  
  
void \_swap\_i(uint64\_t\* a, uint64\_t\* b) {  
uint64\_t tmp = \*a;  
\*a = \*b;  
\*b = tmp;  
}  
  
void foffset(uint8\_t\* mas, size\_t size, size\_t offset, uint64\_t count[255 + 1]) { //вычисляет отступы для для поразрядной сортировки  
size\_t i;  
uint64\_t tmp;  
memset(count, 0, (255 + 1)[uint64\_t](https://vk.com/club40310224)); //заполняет массив нулями  
for (i = 0; i < size \* sizeof(double); i += sizeof(double))  
count[mas[i + offset]]++;  
tmp = count[0];  
count[0] = 0;  
for (i = 0; i < 255; i++) {  
\_swap\_i(&tmp, &count[i + 1]);  
count[i + 1] += count[i];  
}  
}  
  
void \_swap\_p(double\*\* a, double\*\* b) {  
double\* tmp = \*a;  
\*a = \*b;  
\*b = tmp;  
}  
  
void radix\_sort\_LSD(double\* arr, size\_t size) { //Далее следует цикл, который выполняет поразрядную сортировку по каждому байту в числах типа double. В цикле вызывается функция foffset, которая вычисляет смещение для каждого значения и сохраняет его в массиве count. Затем происходит перестановка элементов массива arr во временный массив mas2 на основе вычисленных смещений. После каждой итерации цикла массивы arr и mas2 меняются местами с помощью функции \_swap\_p. После завершения цикла указатель pm устанавливается на начало массива arr.  
uint8\_t\* pm = (uint8\_t\*)arr;  
uint64\_t count[UCHAR\_MAX + 1];  
double\* mas2 = (double\*)malloc(size \* sizeof(double));  
size\_t i, j, k;  
  
for (i = 0; i < sizeof(double); i++) {  
foffset(pm, size, i, count);  
for (j = 0; j < size; j++)  
mas2[count[pm[j \* sizeof(double) + i]]++] = arr[j];  
\_swap\_p(&arr, &mas2);  
pm = (uint8\_t\*)arr;  
}  
//Далее в коде проверяется, есть ли отрицательные числа в массиве. Если есть, то числа размещаются в начале временного массива mas2, а положительные числа - в конце. Затем элементы временного массива mas2 копируются обратно в исходный массив arr, используя цикл. В конце функции освобождается память, занятая временным массивом mas2.  
k = 0;  
if (arr[size - 1] < 0) {  
for (i = size - 1; i >= 0; i--) {  
if (arr[i] > 0) break;  
mas2[k++] = arr[i];  
}  
  
for (i = 0; i < size; i++) {  
if (arr[i] < 0) break;  
mas2[k++] = arr[i];  
}  
  
for (i = 0; i < size; i++) arr[i] = mas2[i];  
}  
  
free(mas2);  
}  
  
int main() {  
clock\_t start, end;  
double cpu\_time\_used;  
double arr[SIZE];  
int i;  
srand(time(0));  
for (i = 0; i < SIZE; i++) {  
arr[i] = (double)(rand() % RANGE);  
}  
printf("массив \n ");  
for (i = 0; i < SIZE; i++)  
printf("%.2lf ", arr[i]) ;  
start = clock();  
radix\_sort\_LSD(arr, SIZE) ;  
end = clock();  
printf("\nОтсортированный массив\n ");  
for (i = 0; i < SIZE; i++)  
printf("%.2lf ", arr[i]);  
qsort(arr, SIZE, sizeof(double), compare);  
printf("\nготовый массив \n");  
for (int i = 0; i < SIZE; i++){  
printf ("%.2lf ", arr[i]);  
}  
cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC; // Вычисляем затраченное процессорное время  
printf("Количество тактов: %ld\n", end - start);  
return 0;  
}