# Московский авиационный институт (национальный исследовательский университет)

## Факультет информационных технологий и прикладной математики

## Кафедра вычислительной математики и программирования


**Лабораторная работа №1 по курсу «Дискретный анализ»**


Студент: А. М. Голев
Преподаватель: С. А. Михайлова
Группа: М8О-201Б

Дата: 23.03.25
Оценка:
Подпись:

**Москва, 2025**

# Лабораторная работа №1

**Задача:** Требуется разработать программу, осуществляющую ввод пар «ключ-значение»их упорядочивание по возрастанию ключа указанным алгоритмом сортировки за линейное время и вывод отсортированной последовательности. Вариант задания определяется типом ключа (и соответствующим ему методом сортировки) и типом значения:

Сортировка подсчётом.

**Тип ключа:** `Числа от` $0$ `до` $65535$.

**Тип значения:** `числа от` $0$ `до` $2^{64} - 1$.

# 1 Описание

Требуется написать реализацию алгоритма сортировки подсчётом.

Основная идея сортировки подсчетом заключается в том, чтобы для каждого входного элемента $x$ определить количество элементов, которые меньше $x$. С помощью этой информации элемент $x$ можно разместить в той позиции выходного массива, где он должен находиться. Например, если всего имеется 17 элементов, которые меньше $x$, то в выходной последовательности элемент $x$ должен занимать 18-ю позицию. [1]

Если входной массив содержит одинаковые элементы, их количество также тербуется учесть. В стабильном варианте сортировки, который реализуется в лабораторной работе, сначала подсчитывается количество каждого элемента, затем к этому количеству добавляется количество элементов меньше данного. На завершающем этапе алгоритма производится перебор элементов входного массива с конца. При этом каждый элемент записывается в ячейку результирующего массива с индексом, равным количеству элементов, меньших или равных данному. После чего это количество уменьшается на 1.

Ниже представлен псевдокод алгоритма стабильной сортировки подсчетом [2] :

```
1 for i = 0 to k - 1
2   C[i] = 0;
3 for i = 0 to n - 1
4   C[A[i]] = C[A[i]] + 1;
5 for j = 1 to k - 1
6   C[j] = C[j] + C[j - 1];
7 for i = n - 1 to 0
8   C[A[i]] = C[A[i]] - 1;
9   B[C[A[i]]] = A[i];
```

$A$ - входной массив

$B$ - результирующий

$C$ - массив счетчиков элементов, меньших или равных данному

В реализации в лабораторной работе также учитывается, что минимальное значение во входном массиве может отличаться от 0.

# 2 Исходный код

На каждой непустой строке входного файла располагается пара «ключ-значение», поэтому создадим новую структуру $Pair$, в которой будем хранить ключ и значение. Объявим функцию $countingSort$ для реализации сортировки подсчетом. Т.к. результат сортировки требуется только вывести в стандартный поток вывода, шаг с копированием в исходный массив функция выполнять не будет.

Выполним проверку, что длина переданного массива больше 0. В противном случае завершим сортировку, вернув пустой массив.

Первый шаг алгоритма - поиск границ диапазона чисел. Создадим переменные $min$ и $max$, в качестве начального значения которых установим ключ первого элемента. В цикле будем перебирать оставшиеся ключи и обновлять значения переменных при необходимости.

Таким образом количество уникальных ключей в массиве не превосходит $max - min + 1$. Создадим массив счетчиков соответствующего размера и инициализируем его элементы нулями. Переберем все ключи. Ключу $key$ соответствует счетчик с индексом $key - min$. На каждой итерации будем инкрементировать счетчик с этим индексом. Тем самым найдем количества всех уникальных ключей.

Значения каждого счетчика увеличим на сумму значений счетчиков, соответствующих меньшим ключам. Тогда значение каждого счетчика будет соответствовать количеству элементов с ключами меньшими или равными ключу, соответствующему данному счетчику. Тогда последний элемент с таким ключем будет занимать индекс в результирующем массиве, равный значению счетчика.

Будем перебирать элементы исходного массива в обратном порядке (для обеспечения естественности сортировки) и записывать в результирующий массив по индексу равному значению соответствующего счетчика. После чего значение будем уменьшать на 1.

В конце функции освободим память, выделенную под счетчики, и вернем результирующий массив.

Объявим функцию $main$, в которой создадим указатель под динамический массив, а также переменные для хранения его размера ($capacity$) и количества элементов ($size$) в нем. Будем считывать строки из стандартного потока ввода пока не встретим EOF и записывать их по индексу $size$ (в конец массива), после чего будем увеличивать значение $size$. При превышении количеством элементов размера - будем выполнять реалокацию на участок памяти большего размера и увеличивать значение $capacity$. Увеличение на несколько ячеек позволит сократить количество реалокаций.

После завершения считывания выполним сортировку при помощи $countingSort$ и напечатаем результирующий массив. В конце освободим память, выделенную под исходный и результирующий массивы.

```c
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>

#define MAX_INPUT_LENGTH 30

typedef struct Pair {
  uint16_t key;
  uint64_t value;
} Pair;

Pair* countingSort (Pair *arr, int n) {
  if (n < 1) return NULL;
  uint16_t min = arr[0].key, max = arr[0].key;

  for (int i = 1; i < n; i++) {
    if (arr[i].key < min) min = arr[i].key;
    if (arr[i].key > max) max = arr[i].key;
  }

  int range = max - min + 1;

  int *counts = malloc(sizeof(unsigned int) * range);
  for (int i = 0; i < range; i++) counts[i] = 0;

  for (int i = 0; i < n; i++) counts[arr[i].key - min]++;

  for (int i = 1; i < range; i++) counts[i] += counts[i - 1];

  Pair *res = malloc(sizeof(Pair) * n);

  for (int i = n - 1; i >= 0; i--) {
    res[--counts[arr[i].key - min]] = arr[i];
  }

  free(counts);

  return res;
}

int main () {
  Pair *arr = NULL;
  int capacity = 0;
  int size = 0;

  char str[MAX_INPUT_LENGTH];

  while (fgets(str, MAX_INPUT_LENGTH, stdin)) {
    if (str[0] == '\n' || str[0] == '\0') continue;
```

```
50
51      if (size >= capacity) {
52        capacity += 10;
53        arr = realloc(arr, sizeof(Pair) * capacity);
54      }
55
56      int scanRes = sscanf(str, "%hu %lu", &(arr[size].key), &(arr[size].value));
57      if (scanRes == 2) size++;
58    }
59
60    Pair *sorted = countingSort(arr, size);
61
62    for (int i = 0; i < size; i++) {
63      printf("%hu\t%lu\n", sorted[i].key, sorted[i].value);
64    }
65
66    free(arr);
67    free(sorted);
68
69    return 0;
70  }
```

# 3 Консоль

```
gcc -o app.out main.c
./app.out
0       13207862122685464576
65535   7670388314707853312
0       4588010303972900864
65535   12992997081104908288
0       13207862122685464576
0       4588010303972900864
65535   7670388314707853312
65535   12992997081104908288
```

# 4 Тест производительности

Тест производительности представляет из себя следующее: сравнивается время сортировки последовательности из $4 * 10^6$ пар ключ и значение методами быстрой сортировки (из стандартной библиотеки языка C) и сортировки подсчетом. Время на ввод и вывод не учитывается. В сортировке подсчетом также не учитывается время на копирование из вспомогательного массива в результирующий (т.е. сортировка подсчетом, в отличие от быстрой, выполняется not in place).

```
smoking_elk@DESKTOP-PJPQAEE:~/discran-labs/lab1/task/benchmark$ make
gcc countingSort.c -o countingSort.out
gcc qsort.c -o qsort.out
smoking_elk@DESKTOP-PJPQAEE:~/discran-labs/lab1/task/benchmark$ make test_counting
./countingSort.out <./in.txt | grep "time"
time: 117.616000ms
smoking_elk@DESKTOP-PJPQAEE:~/discran-labs/lab1/task/benchmark$ make test_quick
./qsort.out <./in.txt | grep "time"
time: 734.725000ms
```

Как видно, реализованная сортировка подсчетом выигрывает у быстрой сортировки из стандартной библиотеки с большим отрывом. Эти результаты согласуются со сложностными оценками алгоритмов: $O(n)$ для сортировки подсчетом против $O(n * log(n))$ для быстрой сортировки.

# 5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научился реализовывать алгоритм сортировки подсчетом на языке программирования C, использовать функцию быстрой сортировки из стандартной библиотеки, производить измерение времени работы программы, используя функции заголовочного файла *time.h* из стандартной библиотеки. Я узнал об особенностях реализации сортировки подсчетом для объектов, отличных от чисел (в данном случае, производилась сортировка пар : по ключу).

# Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Сортировка подсчётом — Википедия.*
URL: `http://ru.wikipedia.org/wiki/Сортировка_подсчётом` (дата обращения: 22.03.2025).