# Лабораторная работа № 1 по курсу Дискретный Анализ. Сортировка за линейное время

Выполнил студент группы 08-207 МАИ *Павлов Иван.*

## Условие

Кратко описывается задача:

1. 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.

2. Вариант задания: 2-1. Тип ключа: почтовые индексы. Тип значения: Строки фиксированной длины 64.

## Метод решения

Я ознакомился с алгоритмом сортировки подсчетом, используя материалы лекций и книгу Томаса Кормена. Алгоритм позволяет за счет дополнительного объема памяти и некоторых ограничений на входные данные (целые неотрицательные числа на небольшом интервале) отсортировать массив за $O(n)$. Выигрыш по времени достигается за счет того, что операция сравнения элементов в нем не применяется. Вместо этого вся сортировка происходит с помощью проходов по исходному массиву и заполнения дополнительной памяти. Дополнительная память представлена в виде результирующего массива индексов размера $n$ и дополнительного массива размера $k$ (здесь и везде в отчете $n$ - длина входного массива ключей; $k$ - длина дополнительного массива, в моем случае равная $max - min + 1$, где $max$ и $min$ соответственно максимальное и минимальное значения ключей в исходном массиве). Для начала результирующий массив заполнен числами от 0 до $n$, а дополнительный массив заполнен нулями. Первым проходом ищем максимум и минимум в исходном массиве. Вторым проходом увеличиваем те элементы промежуточного массива, индексы которых соответствуют значениям исходного массива (уменьшенным на $min$). Третьим проходом обеспечиваем устойчивость сортировки, путем аккумулирования (каждый следующий элемент массива равен сумме двух предыдущих). Четвертым проходом (обратным!) заполняем результирующий массив (элементу результирующего массива, индекс которого равен элементу дополнительного массива за вычетом минимума присваиваем индекс во входном массиве, затем вычитаем из того же элемента дополнительного массива 1, чтобы одинаковые значения вставлялись согласно исходному порядку и соблюдалась устойчивость).

## Описание программы

Из-за ограничений, связанных с платформой Яндекс Контест, пришлось писать весь код в одном файле. Программа состоит из:

1. Вектора типа $int$, хранящего ключи и реализованного по причине особенностей ввода данных $while(scanf(\ldots))$. Устройство вектора:

   1.1 структура $VectorInt$ - хранит указатель на выделенную память для элементов, длину массива и емкость (фактическое кол-во памяти).

   1.2 функция $VectorIntCreate$ - инициализирует структуру, выделяет память под 1 элемент, емкость присваивает 1, длину присваивает 0.

   1.3 функция $VectorIntPushBack$ - вставляет элемент в конец памяти, в случае переполнения перевыделяет память с емкостью в 2 раза больше, размер увеличивает на 1.

   1.4 функция $VectorIntDestroy$ - освобождает память, присваивает указателю $NULL$ для возможной повторной инициализации вектора.

2. Вектора типа $char*$, хранящего значения (строки длиной 64). Устройство отличается от вектора ключей функциями $VectorString64PushBack$ (сразу выделяет 64 бита и считывает из $stdin$ строку) и $VectorString64Destroy$ (перед вызовом $free()$ в цикле очищает всю выделенную под строки память).

3. Самой функции сортировки $countingSort$, принимающей $VectorInt$ ключей и массив (выделенный с помощью $malloc$) результирующих индексов. Заполняет результирующий массив индексами, соответствующими элементам исходного массива ключей и значений для вывода их в отсортированном порядке).

4. Функции $main$, которая состоит из ввода данных (реализованного с помощью $while(scanf("\%d\,\&key) != EOF)$), заполнения результирующего массива, вызова функции сортировки и вывода данных в консоль, с сохранением оформления (с помощью возможностей $printf$).

Я решил не придумывать структуры, а хранить ключи и значения отдельно; также в результирующем массиве хранить именно индексы исходного, а не элементы. Таким образом обеспечен быстрый вывод как ключей, так и значений, без применения дополнительных типов данных.

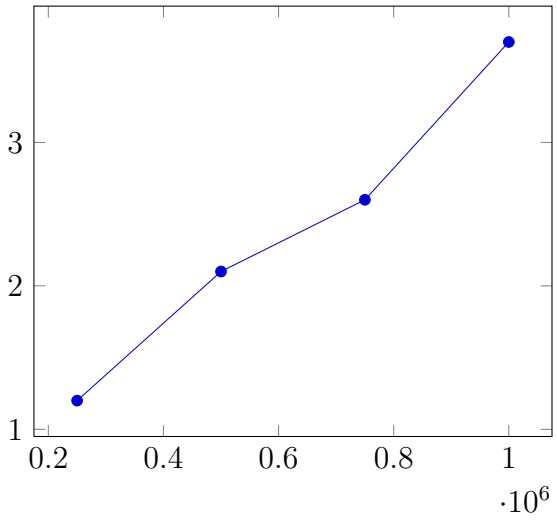
## Дневник отладки

CE1 Случайно запустил код не на том компиляторе.

WA1 Во время считывания значений я не заменил того, что они разделяются знаком $\backslash t$, и выводил его вместе с ответом. Поправил, вызвав $getc(stdin)$ между вводом ключа и значения.

TL13 Программа не проходила по времени, выполняясь почти впритык (за 1.1 секунды). Поэтому были приняты решения по оптимизации кода (убрал *templates*, выделял большой кусок памяти, вместо вектора, ускорял ввод и вывод путем $putc\_unlocked(s[i], stdout)$ и ему подобным, вызывал оптимизации $std :: cin.sync\_with\_stdio(0)$ и $std :: cin.tie(0)$). Проблема решилась, когда я полностью переписал код на Си, реализацию которого я полностью описал выше.

## Тест производительности




Здесь по оси X количество входных данных в $10^6$, по оси Y время в секундах. Данный график можно приблизительно аппроксимировать линейной функцией, из чего следует что алгоритм работает за $O(n)$

## Выводы

Иногда, когда ключи из входного массива являются целыми числами, желательно находящиеся на небольшом промежутке, и объем дополнительной памяти не критичен, можно для ускорения времени работы программы применить сортировку подсчетом. В данном варианте в качестве ключей выступали почтовые индексы, находящиеся в промежутке от 0 до 999999, что делает использование подобной сортировки целесообразным. Сам алгоритм сортировки не показался мне каким-то сложным, трудности возникали только из-за единицы в индексации, однако методом проб и ошибок был получен желаемый результат.