**Министерство науки и высшего образования Российской Федерации**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Факультет безопасности информационных технологий**

**Дисциплина:**

«Алгоритмы и структуры данных»

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**

«Реализация алгоритма сортировки Radix,

используя дек»

**Выполнил:**

Арендаренко М.М, студент группы N3247

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

**Проверил:**

Ерофеев С. А.

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

(подпись)

Санкт-Петербург

2024 г.

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc164780957)

[1 Описание функционала программы 4](#_Toc164780958)

[2 Блок-схемы 6](#_Toc164780959)

[3 Описание алгоритма сортировки на примере 13](#_Toc164780960)

[4 Код программы 14](#_Toc164780961)

[5 Результаты тестирования 20](#_Toc164780962)

[6 Оценка сложности сортировки 22](#_Toc164780963)

[Заключение 23](#_Toc164780964)

Введение

Задача работы – разработать программу сортировки Radix, используя дек на базе массива. Оценить сложность сортировки посредством подсчёта количества элементарных операций в зависимости от количества чисел в файле.

Для реализации поставленной цели мне необходимо решить следующие

задачи:

• Реализовать структуру «дек» на базе массива;

• Адаптировать алгоритм сортировки Radix под написанную

структуру;

• Реализовать способ подсчета количества элементарных операций;

• Построить график зависимости количества операций от количества

сортируемых элементов и найти эту зависимость

В данной работе реализация дек необходима для временного хранения элементов в каждом разряде числа. Алгоритм сортировки Radix работает путем сортировки чисел по разрядам, начиная с младшего разряда и двигаясь к старшим.

В процессе сортировки Radix каждое число разбивается на отдельные разряды (обычно в десятичной системе это цифры). Затем числа распределяются по корзинам в соответствии с текущим разрядом. Дек используется для временного хранения чисел в каждой корзине. При этом числа добавляются в дек с одной стороны и извлекаются из него с другой стороны, чтобы сохранить правильный порядок.

После того, как все числа были размещены в соответствующих корзинах, они извлекаются из дека и объединяются в правильном порядке. Затем процесс повторяется для следующего разряда до тех пор, пока все разряды не будут обработаны.

Использование дека в алгоритме сортировки Radix позволяет эффективно управлять порядком обработки чисел и сохранять правильный порядок при объединении отсортированных разрядов.

Для реализации алгоритма был выбран язык программирования C++ стандарта C++17, для написания и отладки кода использовалась среда разработки Replit версии 2.66.9.

# Описание функционала программы

Дек (двусторонняя очередь) в сортировке radix используется для временного хранения элементов, имеющих одинаковое частное разряда. Это помогает поддерживать порядок элементов в каждом разряде, когда они обрабатываются и перераспределяются в последовательности с возрастающими значениями разряда.

В процессе сортировки radix числа разбиваются на разряды, и каждое число обрабатывается по каждому разряду с использованием сортировки подсчетом. Дек используется для того, чтобы:

Собрать элементы с одинаковым значением разряда:

Когда элементы обрабатываются по определенному разряду, те, которые имеют одинаковое значение, помещаются в дек.

Перераспределять элементы в порядке возрастания значений разряда: После того, как элементы с одинаковым значением разряда собраны в дек, они перераспределяются в исходный массив в порядке возрастания значений разряда.

Сохранять стабильность:

Дек обеспечивает стабильность сортировки, гарантируя, что элементы с одинаковыми значениями в старших разрядах сохраняют свой относительный порядок в итоговом отсортированном массиве.

Использование дека в Radix sort позволяет эффективно и стабильно сортировать числа по нескольким разрядам, разделяя и перераспределяя элементы на каждом этапе.

Для реализации алгоритма потребовалось реализовать следующие функции:  
counting\_sort - функция для сортировки массива на одном разряде с использованием метода подсчета.

radix\_sort - функция для вызова counting\_sort для каждого разряда чисел, начиная с младшего и заканчивая старшим.

main - основная функция, в которой происходит открытие и считывание входного файла, вызов radix\_sort для сортировки, запись отсортированных данных и количества операций в выходной файл.

В программе с реализацией структуры и сортировки будут использоваться следующие переменные:

Элементы очереди (data) – тип int (диапазон от –2 147 483 648 до 2 147 483 647)

arr — это массив целых чисел, который требуется отсортировать.

exp — это переменная, представляющая разряд числа, по которому происходит сортировка в методе поразрядной сортировки.

action\_count — это переменная, которая отвечает за подсчет количества операций, которые выполняются в процессе сортировки.

n — это переменная, представляющая размер массива.

output — это массив, в который будет складываться отсортированный результат.

count — это вектор, содержащий очереди длиной 19 (от -9 до 9) для подсчета разрядов.

index — это переменная, используемая для хранения индекса разряда числа при подсчете.

i — это переменная-счетчик, используемая в цикле для обработки элементов массива после подсчета.

num — это переменная, в которую считывается каждое число из файла во время чтения входных данных.

fin — это объект для чтения данных из файла "input.txt".

fout — это объект для записи данных в файл "output.txt".

max\_num — это переменная, содержащая максимальное число в исходном массиве.

min\_num — это переменная, содержащая минимальное число в исходном массиве.

max\_digit — это переменная, которая содержит максимальное значение разряда числа, используется для определения количества разрядов.

# Блок-схемы

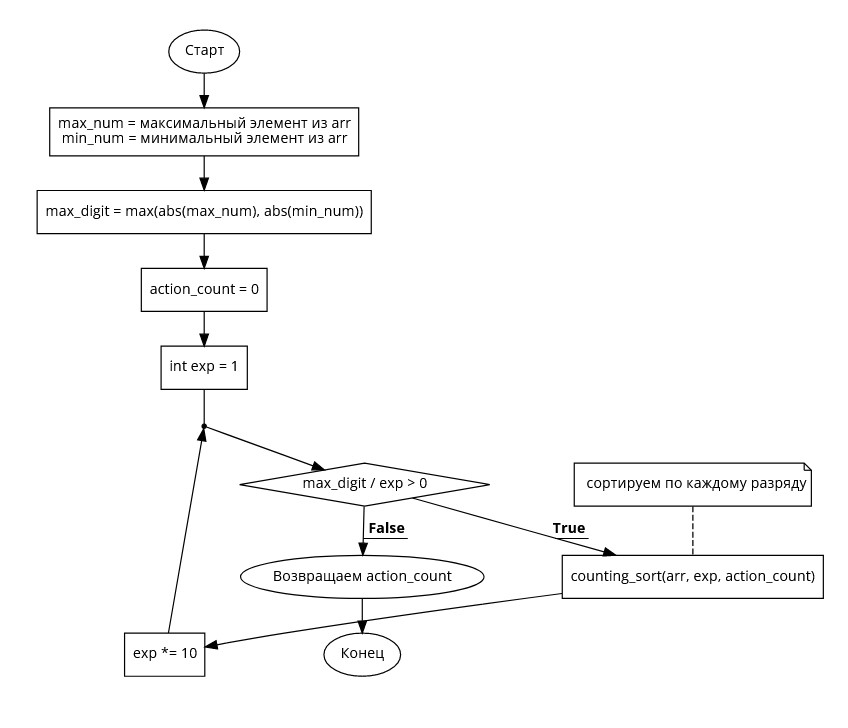


Рис. 1 Функция "radix\_sort"

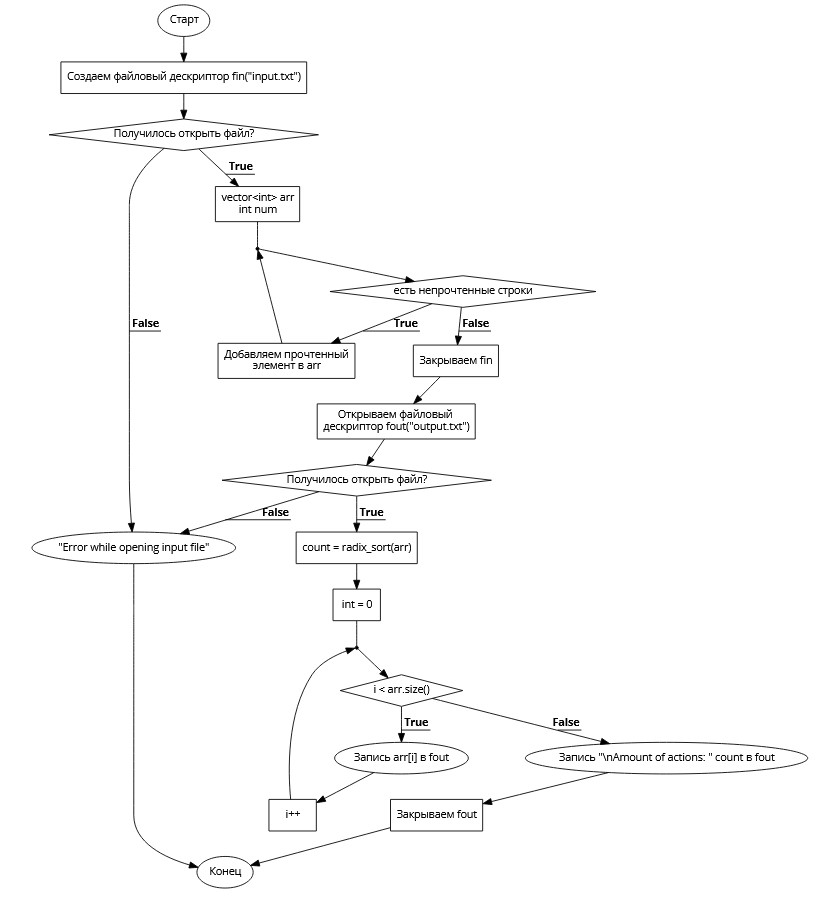


Рис. 2 Функция "main"

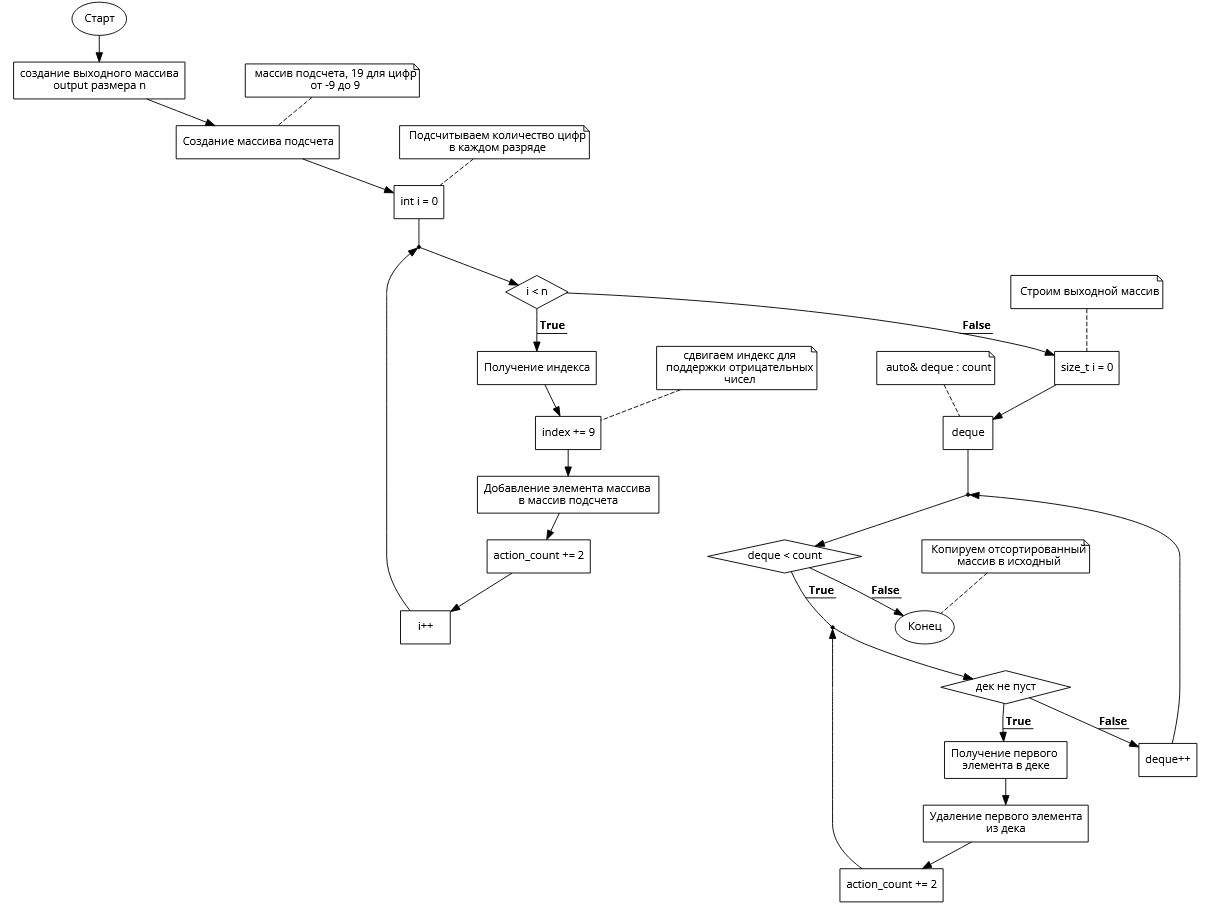


Рис. 3 Функция "counting\_sort"

# Описание алгоритма сортировки на примере

Предположим, у нас есть массив чисел [170, 45, 75, 90, 802, 24, 2, 66]. Мы хотим отсортировать их при помощи поразрядной сортировки.

1. Поиск максимального числа в массиве и определение количества разрядов. В данном примере максимальное число 802, которое имеет три разряда, поэтому нам понадобятся три прохода для сортировки.

2. Начинаем с разряда единиц: сортируем числа по этому разряду и помещаем их в соответствующие корзины (0-9).

3. После этого собираем числа обратно в массив, сохраняя порядок в корзинах.

4. Повторяем процесс для разряда десятков, а затем для разряда сотен (если есть).

5. После третьего прохода весь массив будет отсортирован по всем разрядам.

В итоге, собрав все числа вместе после трех проходов, мы получим отсортированный массив: [2, 24, 45, 66, 75, 90, 170, 802].

# Код программы

|  |
| --- |
| #include <cstdlib>  #include <string>  #include <limits>  #include <cstring>  #include <iostream>  #include <vector>  #include <deque>  #include <fstream>  #include <algorithm> // for max\_element  // ОБЩАЯ ОЦЕНКА: O(3 + 3 + 1 + n + 19 + 1 + 23n + 5 + 2 + 13n + 77) = O(37n + 111)  // O(3) -- создание arr, exp, action\_count  void counting\_sort(std::vector<int>& arr, int exp, size\_t& action\_count)  {  // Количество операций(3) -- создание n, вычисление arr.size(), инициализация n  int n = arr.size();  // Количество операций(1) -- создание output  std::vector<int> output; // выходной массив  // Количество операций(n) -- выделение памяти размера n  output.reserve(n);  // Количество операций(19 + 1) -- создание вектора count размера 19  std::vector<std::deque<int>> count(19); // массив подсчета, 19 для цифр от -9 до 9  // Подсчитываем количество цифр в каждом разряде  // оценка цикла for Количество операций(23n + 5)  // первая строка for:  // Количество операций(2) -- создание i и инициализация  // Количество операций(3) -- чтение i, чтение n и сравнение i < n  // Количество операций(23\*n) -- количество операций для n итераций цикла  for (int i = 0; i < n; i++) {  // Количество операций(7):  // создание переменной index, инициализация переменной index  // чтение i, чтение exp, чтение arr[i], деление arr[i] / exp, взятие %10  int index = (arr[i] / exp) % 10;  // Количество операций(5)  // чтение index, сранение с 0, чтение index, увеличение на 9, запись в index  index = index < 0 ? index + 9 : index + 9; // сдвигаем индекс для поддержки отрицательных чисел  // Количество операций(4)  // чтение index, чтение i, чтение arr[i], удаление последнего  count[index].push\_back(arr[i]);  // Количество операций(3)  // чтение action\_count, добавление 2, сохранение в action\_count  action\_count += 2;  // Количество операций(3) -- чтение i, чтение n и сравнение i < n  // Количество операций(3) -- i++ : чтение i, увеличение на 1, запись в i    // Количество операций(23) -- общее количество операций для одной итерации цикла  }  // Строим выходной массив  // Количество операций(2) -- создание и инициализация i  size\_t i = 0;  // Общая оценка цикла: Количество операций(1 + 19\*4 + 13n) = Количество операций(13n + 77)  // Количество операций(1) -- создание deque  // Количество операций(19 \* 4) -- оценка для 19 значений итераций for вне while  // Количество операций(13 \* n) -- оценка для n итераций внутри while (их n, поскольку нужно расставить n элементов)  for (auto& deque : count) {  // первая строка while:  // Количество операций(3) -- получение deque.empty(), взятие обратного !, проверка  while (!deque.empty()) {  // Количество операций(6)  // чтение i, прибавление 1 к i, запись в i, получение deque.front(), чтение arr[i++], запись arr[i++]  arr[i++] = deque.front();  // Количество операций(1) -- удаление элемента  deque.pop\_front();  // Количество операций(3) -- чтение action\_count, добавление 2, запись в action\_count  action\_count += 2;  // Количество операций(3) -- получение deque.empty(), взятие обратного !, проверка    // Количество операций(13) -- общая оценка для одной итерации  }  // Количество операций(1) -- изменение deque    // Количество операций(4) -- оценка вне while для одной итерации  }      }  // ОБЩАЯ ОЦЕНКА: O(1) + O(n + 5) + O(n+5) + O(7) + O(2) + O((37n + 122) \* k + 6) + O(1) =  // = O((37n + 122) \* k + n + 27), где k -- максимальное количество разрядов  // Количество операций(1) -- передача arr  size\_t radix\_sort(std::vector<int>& arr) {  // Оценка первой строки: Количество операций(n + 5)  // Количество операций(1) -- создание переменой max\_num  // Количество операций(2) -- получение arr.begin() и arr.end()  // Количество операций(n) -- поиск ссылки на максимум std::max\_element(arr.begin(), arr.end())  // Количество операций(1) -- получение значение максимума по ссылке  // Количество операций(1) -- присвоение значения переменной max\_num  int max\_num = \*std::max\_element(arr.begin(), arr.end());  // Оценка второй строки: O(n + 5) -- по аналогии  int min\_num = \*std::min\_element(arr.begin(), arr.end());  // Оценка третьей строки: Количество операций(7)  // Количество операций(1) -- создание переменной max\_digit  // Количество операций(1) -- чтение значения max\_num  // Количество операций(1) -- чтение значения min\_num  // Количество операций(1) -- нахождение модуля максимума  // Количество операций(1) -- нахождение модуля минимума  // Количество операций(1) -- нахождение максимума модулей  // Количество операций(1) -- присвоение значения переменной max\_digit (инициализация)  int max\_digit = std::max(std::abs(max\_num), std::abs(min\_num));  // Оценка четвертой строки: Количество операций(2) -- создание и присвоение переменной  size\_t action\_count = 0;  // Оценка блока с for: O((37n + 122) \* k + 6)  // оцениваем первую строчку for:  // Количество операций(2) -- создание и инициализация exp  // Количество операций(4) -- чтение max\_digit и exp, деление max\_digit / exp, сравнение с 0  for (int exp = 1; max\_digit / exp > 0; exp \*= 10) {  // Количество операций(3) -- копирование значений в вызов функции, О(1) + O(37n + 111) -- вызов функции  counting\_sort(arr, exp, action\_count); // сортируем по каждому разряду  // опять оцениваем первую строчку for  // Количество операций(3) -- чтение exp, умножение exp на 10, запись значения в exp  // Количество операций(4) -- чтение max\_digit и exp, деление max\_digit / exp, сравнение с 0  // общая оценка итерации: O(37n + 122)  }  return action\_count; // Количество операций(1)  }  int main()  {  std::ifstream fin("input.txt");  if (!fin) {  std::cout << "Error while opening input file";  return -1;  }  std::vector<int> arr;  int num;  while (fin >> num) {  arr.push\_back(num);  }  if(!fin.eof()) {  fin.close();  std::cout << "Error while reading input file";  return -2;  }  fin.close();  std::ofstream fout("output.txt");  if (!fout) {  std::cout << "Error while opening output file";  return -3;  }  size\_t count = radix\_sort(arr);  for (const auto& number : arr) {  fout << number << " ";  }  fout << "\nAmount of actions: " << count;  fout.close();  } |

# Результаты тестирования

Было проведено тестирование программы с различными входными данными. Сортировка входных данных производится успешно, в случае некорректных данных программа выводит соответствующие сообщения на экран. Тестирование производилось с помощью среды разработки Replit версии 2.66.9. Ниже представлены скриншоты входного и выходного файла для кольцевой очереди, состоящей из 100 элементов, а также случай некорректных входных данных.

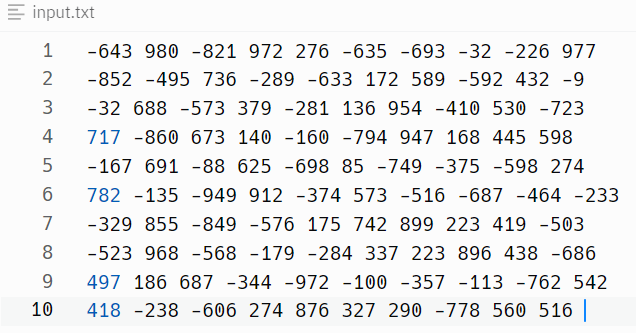


Рис. 4 Файл "input.txt"

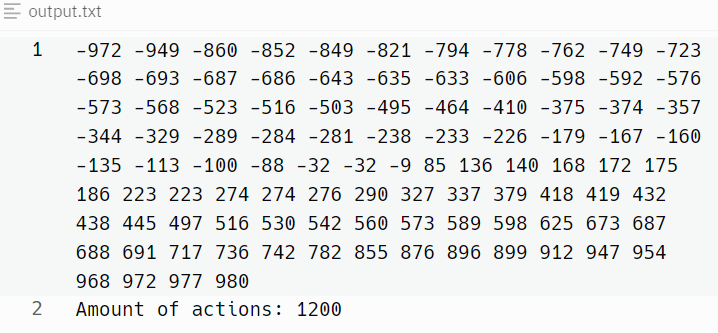
**

Рис. 5 Файл "output.txt"

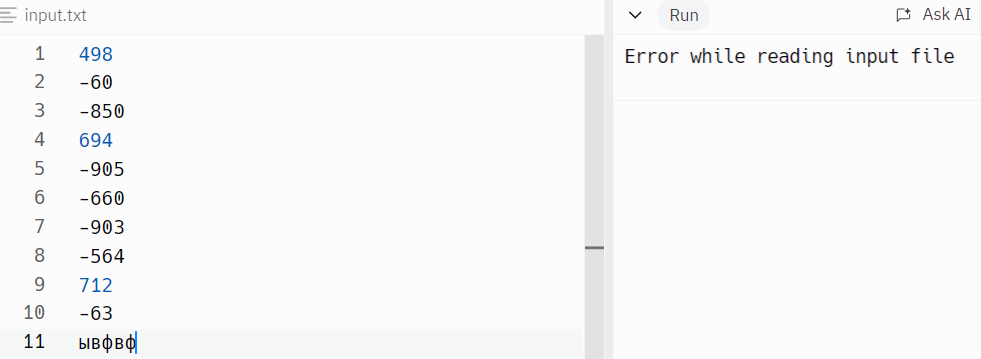


Рис. 6 Некорректные для сортировки данные

# ОЦЕНКА СЛОЖНОСТИ СОРТИРОВКИ

Одной из целей лабораторной работы являлась оценка сложности сортировки Radix. Для подсчёта количества элементарных операций в функции «counting\_sort» была добавлена новая переменная «action\_count».

Чтобы выполнить данное задание, был модифицирован предыдущий код

Далее представлены некоторые данные полученные на выходе программы, различные значения переменной «action\_count», а также графически показана зависимость количества элементарных операций от количества чисел в файле:

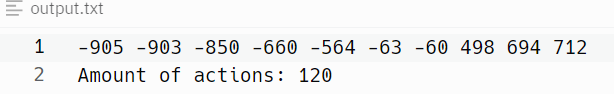


Рис. 7 Файл «output.txt»

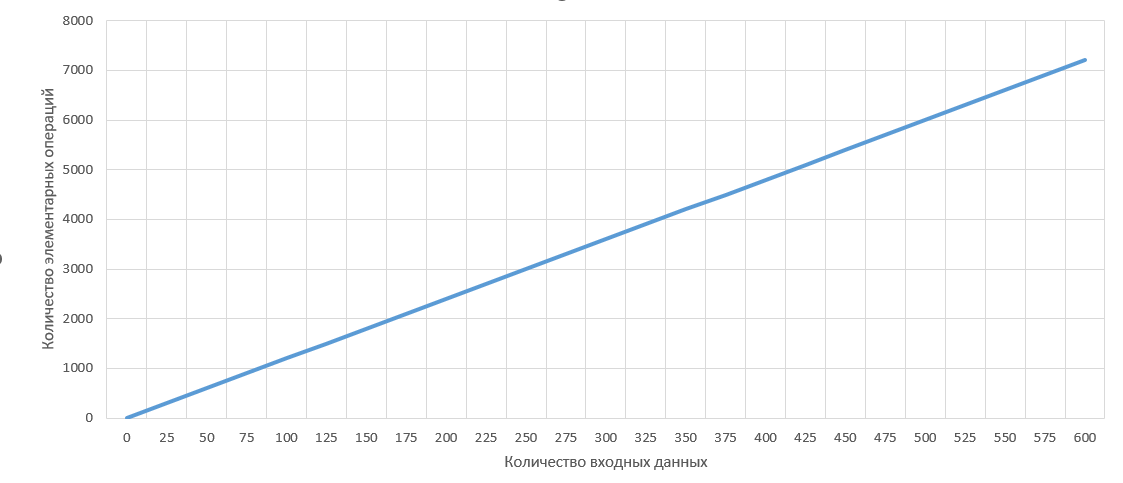


Рис. 8 Оценка сложности

 Путем подсчета количества элементарных операций в худшем случае мы получим сложность O(37n + 122)

# Заключение

В ходе выполнения лабораторной работы были достигнуты все поставленные задачи. В рамках работы была разработана программа сортировки Radix, использующая дек на основе массива. Кроме того, проведена оценка сложности сортировки путем подсчета количества элементарных операций в зависимости от количества чисел в файле.

Тестирование программы было проведено с использованием среды разработки Replit версии 2.66.9 и языка программирования C++17. В процессе тестирования были использованы различные входные данные, и программа продемонстрировала корректную работу, а сортировка была выполнена верно. Были обнаружены и обработаны некорректные случаи ввода.