Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

“Сортировка данных”

Отчет по лабораторной работе №8

По дисциплине

«Структуры и алгоритмы обработки данных в ЭВМ»

Студент гр. 431-3

_____ Д.П. Андреев

«___»_____ 2023 г.

Проверил: профессор кафедры АСУ, д.т.н.

_____ А.Н. Горитов

«___»_____ 2023 г.

Томск 2023

1.Задание на лабораторную работу

Пусть даны три файла вещественных чисел из интервала от 0 до 1, состоящих из 100, 1000 и 10000 чисел. Выполнить сортировку данных с помощью метода блочной сортировки. Подсчитать трудоемкость реализованного алгоритма для каждого из трех файлов.

2.Алгоритм решения задачи

Первый шаг — это открытие файлов и заполнение массивов. Далее передаём массивы в функцию сортировки засекая время работы функции. В функции вычисляем количество блоков, разбиваем массив на блоки, сортируем каждый блок, объединяем отсортированные блоки обратно в массив после чего выводим результат на экран.

3.Листинг программы

```
#include <iostream>
#include <fstream>
#include <algorithm>
#include <chrono>

using namespace std;

void printArray(double* arr)
{
   for (int i = 0; i< _msize(arr) / sizeof(double); i++)
   {
      cout<<arr[i]<<endl;
   }
   cout << endl;
}

// Функция блочной сортировки
void blockSort(double* arr, int blockSize)
{
   // Вычисляем количество блоков
   int numBlocks = (_msize(arr) / sizeof(double)) / block-
Size;
   if (_msize(arr) / sizeof(double) % blockSize != 0) num-
Blocks++;

   // Создаем вектор блоков
   double block[10][10];

   // Разбиваем массив на блоки
   for (int i = 0; i < _msize(arr) / sizeof(double); i++)
   {
      int blockIndex = i / blockSize;
      block[blockIndex][0] = arr[i];
   }
   double buf[10];
   for (int i = 0; i < numBlocks; i++)
   {
      buf[i] = block[i][0];
   }
    //Сортируем каждый блок
   for (int i = 1; i < numBlocks; i++)
            {
                for (int j = i; j > 0 && x[j - 1] > x[j]; j--) //
пока j>0 и элемент j-1 > j, x-массив int
                {
                        swap(buf[j - 1], x[j]);           // меняем
местами элементы j и j-1
                }
   }
   // Объединяем отсортированные блоки обратно в
массив
   int arrIndex = 0;
   for (int i = 0; i < numBlocks; i++)
   {
      for (int j = 0; j < _msize(block) / sizeof(double); j++)
      {
         arr[arrIndex] = block[i][j];
         arrIndex++;
      }
   }
   printArray(arr);
}


int main()
{
   setlocale(LC_ALL, "RU");
   // Открываем файлы
   ifstream file1("input100.txt");
   ifstream file2("input1000.txt");
   ifstream file3("input10000.txt");

   // Считываем данные из файлов
   double* arr1=new double[100];
   double* arr2= new double[1000];
   double* arr3= new double[10000];
   double val;

   for (int i=0; file1 >> val;i++)
   {
      arr1[i]=val;
   }
   for (int i=0;file2 >> val;i++)
   {
      arr2[i]=val;
   }
   for (int i=0;file3 >> val;i++)
   {
      arr3[i]=val;
   }


   // Запускаем таймер для первого файла
   auto start = chrono::high_resolution_clock::now();

   // Сортируем первый файл
   blockSort(arr1, 10);
   //printArray(arr1);
   // Останавливаем таймер
   auto finish = chrono::high_resolution_clock::now();
   chrono::duration<double> elapsed = finish - start;
   time[0] = elapsed.count();

   // Запускаем таймер для второго файла
   start = chrono::high_resolution_clock::now();

   // Сортируем второй файл
   blockSort(arr2, 100);

   // Останавливаем таймер
   finish = chrono::high_resolution_clock::now();
   elapsed = finish - start;
   time[1] = elapsed.count();

   // Запускаем таймер для третьего файла
   start = chrono::high_resolution_clock::now();

   // Сортируем третий файл
   blockSort(arr3, 1000);

   // Останавливаем таймер
   finish = chrono::high_resolution_clock::now();
   elapsed = finish - start;
   time[2] = elapsed.count();
   cout << "Файл input100 отсортирован за " << time[0]
<< " секунды\n";
   cout << "Файл input1000 отсортирован за " << time[1]
<< " секунды\n";
   cout << "Файл input10000 отсортирован за " <<
time[2] << " секунды\n";

   // Закрываем файлы
```

```
file1.close();
file2.close();                                  return 0;
file3.close();                                  }
```

4.Пример решения

Пример входных данных можно увидеть на рисунке 4.1.

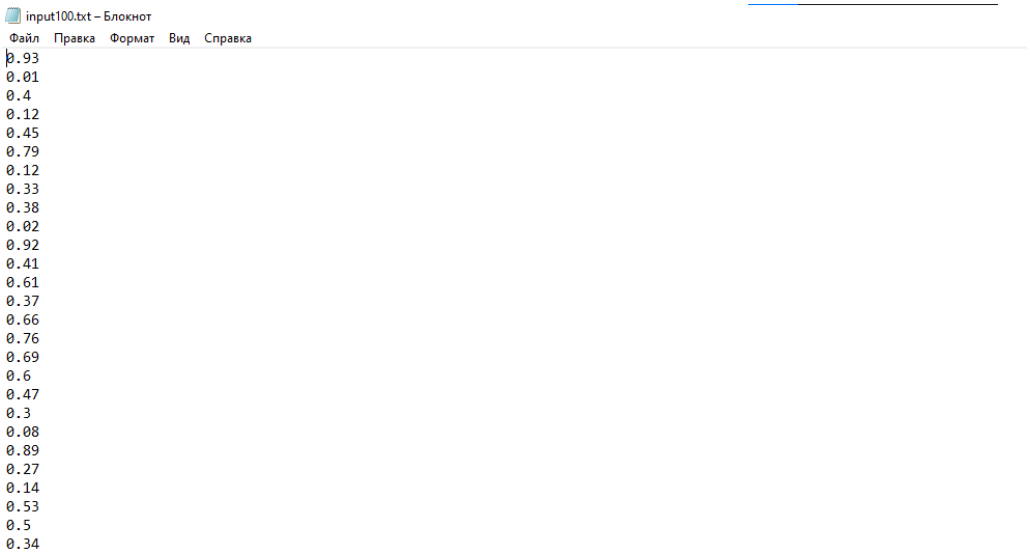

Рисунок 4.1— Входные данные из файла input100.txt

Результат программы можно увидеть на рисунке 4.2.

Рисунок 4.2— Выходные данные после запуска программы

5.Вывод

В результате лабораторной работы были изучен алгоритм блочной сортировки данных. Его преимущества в эффективности на больших объемах данных, быстрый алгоритм с линейным временем исполнения O(N). Недостатки блочной сортировки это неэффективность на малых объемах данных и неустойчивость.

Так же был изучен алгоритм сортировки вставками. Одним из главных преимуществ метода сортировки вставками является его простота. Он очень легко реализуется. Кроме того, он эффективен для сортировки небольших массивов. Также, если массив почти отсортирован, то метод сортировки вставками работает очень быстро. Недостатком метода сортировки вставками является то, что он не является эффективным для сортировки больших массивов данных. В таком случае он может работать очень медленно.