МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Сортировки

Студент гр. 9383 Ноздрин В.Я.

Преподаватель Попова Е.В.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с некоторыми алгоритмами сортировки, получить. Решить задачу реализации одного из таких алгоритмов на языке C++.

Задание 13.

Реализовать итеративный алгоритм сортировки массива простым слиянием.

Теория.

Алгоритм сортировки простым слиянием устроен следующим образом:

- Если есть два отсортированных массива, создается новый блок/массив, в который записываются данные двух отсортированных блоков следующим образом. Сначала из двух массивов выбирается минимальный элемент до тех пор, пока один из массивов не станет пустым. Далее то, что остается в одном из массивов добавляется в блок/массив.
- В случае рекурсивный сортировки одного массива, он разбивается на две примерно равные части, для которых в свою очередь рекурсивно вызывается такая же сортировка слиянием. После чего две отсортированных части сливаются, что дает в результате отсортированный изначальный массив.
- В случае итеративной сортировки одного массива, массив сначала разбивается на блоки длины один, затем эти блоки разбиваются на пары “соседних” и сливаются, порождая разбиение изначального массива на блоки длины два. Далее снова соседние блоки сливаются, порождая еще более крупное разбиение. Слияния продолжаются до тех пор, пока не получится один единственный блок, который и будет отсортированным изначальным массивом.

Выполнение работы.

В итеративной реализации алгоритма сортировки слиянием, итерация происходит по размеру блока ***BlockSizeIterator***. На каждом цикле размер блока увеличивается в два раза. Внутри цикла происходит итерация по блокам ***BlockIterator*****,** где каждый большой блок — два соседних блока размера ***BlockSizeIterator***. Когда определены размеры, в переменных ***LeftBorder*****,** ***MidBorder*****,** ***RightBorder*** соответственно хранятся индексы левой границы первого блока, правой границы первого блока, которая является так же левой границей второго блока, и правая граница второго блока. Также задаются два итератора — сдвиги для обоих блоков. Итераторы будут указывать на наименьший еще не использованный элемент блока. Когда определены границы и итераторы, выделяется память ***SortedBlock*** в которой будут сливаться два блока. То есть в этот массив будут записываться данные из двух блоков. Для этого мы сравниваем элементы, хранящиеся в ***LeftBorder*** и ***RightBorder***, и записываем меньший из них в ***SortedBlock***, при этом мы увеличиваем на 1 итератор соответствующего блока. Таким образом объединяем элементы упорядоченно в ***SortedBlock*** до тех пор, пока один из блоков не станет пустым, то есть его левая граница, сдвигаясь на итератор, оказывается на его правой границе. Далее элементы из оставшегося блока переносятся в ***SortedBlock***.

После этого содержимое из ***SortedBlock*** копируется в изначальный массив поверх двух блоков. Таким образом два блока теперь слиты в один упорядоченный блок.

Итерация прекращается, когда два последних блока, сливаясь, дают весь массив.

Пример работы программы.

Входные данные (в файле input.txt):

628 746 687 619 866 167 162 534 171 772 543 128 449 703 950 649 319 234 696 858 830 211 761 418 401

Выходные данные:

array = [628 746 687 619 866 167 162 534 171 772 543 128 449 703 950 649 319 234 696 858 830 211 761 418 401]

sorted = [128 162 167 171 211 234 319 401 418 449 534 543 619 628 649 687 696 703 746 761 772 830 858 866 950]

std   = [128 162 167 171 211 234 319 401 418 449 534 543 619 628 649 687 696 703 746 761 772 830 858 866 950]

Разработанный программный код см. в приложении А.

Вывод.

Был реализован итеративный алгоритм сортировки простым слиянием на языке программирования C++.

Приложение А
Исходный код программы

Название файла: main.c

```
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <functional>
#include <cstdlib>

#define DEBUG 0

template<typename T>
void MergeSort(T arr[], size_t length) {
#if DEBUG
  std::cerr << "   0:   ";
  for(int k = 0; k < length; k++)
    std::cerr << " | " << arr[k];
  std::cerr << "\n";
#endif // DEBUG
  size_t BlockSizeIterator;
  size_t BlockIterator;
  size_t LeftBlockIterator;
  size_t RightBlockIterator;
  size_t MergeIterator;
  size_t LeftBorder;
  size_t MidBorder;
  size_t RightBorder;
   for (BlockSizeIterator = 1; BlockSizeIterator < length; BlockSizeIterator *=
2) {
       for (BlockIterator = 0; BlockIterator < length - BlockSizeIterator;
BlockIterator += 2 * BlockSizeIterator) {
       // Производим слияние с сортировкой пары блоков начинающуюся с элемента
BlockIterator
      // левый размером BlockSizeIterator, правый размером BlockSizeIterator или
меньше
      LeftBlockIterator = 0;
      RightBlockIterator = 0;
      LeftBorder = BlockIterator;
      MidBorder = BlockIterator + BlockSizeIterator;
      RightBorder = BlockIterator + 2 * BlockSizeIterator;
      RightBorder = (RightBorder < length) ? RightBorder : length;
      int* SortedBlock = new int[RightBorder - LeftBorder];
       //Пока в обоих массивах есть элементы выбираем меньший из них и заносим в
отсортированный блок
          while (LeftBorder + LeftBlockIterator < MidBorder && MidBorder +
RightBlockIterator < RightBorder) {
                if (arr[LeftBorder + LeftBlockIterator] < arr[MidBorder +
RightBlockIterator]) { // cmp
          SortedBlock[LeftBlockIterator + RightBlockIterator] = arr[LeftBorder +
LeftBlockIterator];
          LeftBlockIterator += 1;
        } else {
           SortedBlock[LeftBlockIterator + RightBlockIterator] = arr[MidBorder +
RightBlockIterator];
```

```
          RightBlockIterator += 1;
        }
      }

      //После этого заносим оставшиеся элементы из левого или правого блока
      while (LeftBorder + LeftBlockIterator < MidBorder) {
         SortedBlock[LeftBlockIterator + RightBlockIterator] = arr[LeftBorder +
LeftBlockIterator];
        LeftBlockIterator += 1;
      }
      while (MidBorder + RightBlockIterator < RightBorder) {
         SortedBlock[LeftBlockIterator + RightBlockIterator] = arr[MidBorder +
RightBlockIterator];
        RightBlockIterator += 1;
      }

             for (MergeIterator = 0; MergeIterator < LeftBlockIterator +
RightBlockIterator; MergeIterator++) {
        arr[LeftBorder + MergeIterator] = SortedBlock[MergeIterator];
      }
      delete SortedBlock;
    }
#if DEBUG
    char buf[1024]; sprintf(buf, "%4d:   ", BlockSizeIterator);
    std::cerr << std::string(buf);
//     std::cerr << BlockSizeIterator << ":  ";
    for(int k = 0; k < length; k++) {
      if (!(k % (BlockSizeIterator*2)))
        std::cerr << " | " << arr[k];
      else
        std::cerr << "   " << arr[k];
    }
    std::cerr << "\n";
#endif // DEBUG
  }
}

int main() {
  std::ifstream input("input.txt");
  for (std::string line; getline(input, line); ) {
    int arr[25] = {};
    int n = 0;
    int i = 0;
    std::cout << "array  = [";
    for (int j = i; j < line.length(); j++) {
      if (line[j] == ' ' || j+1 == line.length()) {
        arr[n++] = stoi(line.substr(i,j));
        std::cout << arr[n-1] << " ";
        i = j+1;
      }
    }
    std::cout << "]\n";
    // MergeSort and print
    MergeSort<int>(arr, n);
    std::cout << "sorted = [";
    for(auto x: arr)
      std::cout << x << " ";
    std::cout << "]\n";
```

```
    // std::sort and print
    std::sort(arr, arr+n);
    std::cout << "std    = [";
    for(auto x: arr)
      std::cout << x << " ";
    std::cout << "]\n";
  }
  return 0;
}
```