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. 9384 _____ Соседков К.С.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Получить практические навыки написания алгоритмов сортировки данных.

Задание. (Вариант № 16)

Сортировка массивов слиянием – естественное слияние.

Выполнение работы.

Для выполнения работы был разработан шаблонный класс *custom_vector* со следующими методами:

*push*_back — добавляет элемент в вектор

resize — увеличивает размер вектора

print — выводит вектор на экран

begin, *end* — возвращает *iterator* на начало и конец вектора

sort,*split*,*merge*,*find_end_of_sequence* — методы для сортировки вектора

Алгоритм сортировки естественным слиянием:

1) Из исходного массива выделяются две подряд идущие возрастающие подпоследовательности

2) Эти подпоследовательности сливаются в одну упорядоченную подпоследовательность.

3) Шаги 1 и 2 повторяются до тех пор, пока не будет достигнут конец массива

4) Шаги 1 и 3 применяются к новому полученному массиву если он еще не отсортирован

Анализ алгоритма:

В лучшем случае сложность алгоритма равна O(n), это достигается когда массив уже упорядочен.

В среднем и худшем случае сложность равна O(n*logn), так же как у стандартной сортировки слиянием.

В плане реализации алгоритм прост и понятен.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	1 2 3 4 5	1 2 3 4 5
2.	-2 3 2 90 -12	-12 -2 2 3 90
3.	321 21 21 21 9 8 7 6 5	5 6 7 8 9 21 21 21 321

Выводы.

В ходе выполнения лабораторной работы были изучены алгоритмы сортировки, а так же была реализована сортировка естественным слиянием.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#pragma once
#include <iostream>
#include <algorithm>

template <class T>
class custom_vector
{
private:
   T* vector=nullptr;
   int allocated_size = 10;
   int current_size = 0;

public:
   class iterator {
   public:
      typedef iterator self_type;
      typedef T value_type;
      typedef T& reference;
      typedef T* pointer;
      typedef std::forward_iterator_tag iterator_category;
      typedef int difference_type;
      iterator(T* ptr) : ptr_(ptr) { }
      bool operator<(const self_type& rhs) { return ptr_ < rhs.ptr_; }
      self_type operator+(int junk) { self_type i = *this; return ptr_+junk; }
      self_type operator-(int junk) { self_type i = *this;  return ptr_-junk; }
      self_type operator++() { self_type i = *this; ptr_++; return i; }
      self_type operator++(int junk) { ptr_++; return *this; }
      reference operator*() { return *ptr_; }
      pointer operator->() { return ptr_; }
      bool operator==(const self_type& rhs) { return ptr_ == rhs.ptr_; }
      bool operator!=(const self_type& rhs) { return ptr_ != rhs.ptr_; }
   private:
      pointer ptr_;
   };
```

```
custom_vector() {
   this->vector = new T[allocated_size];
};

custom_vector(std::initializer_list<T> c) {
   this->vector = new T[c.size()];
   allocated_size = c.size();
   for(auto it = c.begin(); it!=c.end(); it++) {
      this->push_back((*it));
   }
};

// move
      custom_vector<T>(custom_vector&& other) {
   vector = other.vector;
               other.vector = nullptr;
   current_size = other.size();
   allocated_size = other.allocated_size;
      }

      // move
      custom_vector<T>& operator=(custom_vector<T>&& other) {
               if (&other == this) return *this;
   if(vector) {
                  delete []vector;
   }
   vector = other.vector;
               other.vector = nullptr;
   current_size = other.size();
   allocated_size = other.allocated_size;
               return *this;
      }

~custom_vector() {
   if(this->vector) {
      delete []this->vector;
      this->vector = nullptr;
   }
};
```

```
int size() {
   return this->current_size;
}

void push_back(T data) {
   if(this->current_size == this->allocated_size) {
      this->resize(allocated_size*2);
   }
   this->vector[this->current_size++] = data;
};

void resize(int n) {
   this->allocated_size = n;
   T* buffer = new T[n];
   if(this->vector) {
      std::copy_n(this->vector, this->size(), buffer);
      delete []this->vector;
   }
   this->vector = buffer;
}

iterator begin() {
   return iterator(this->vector);
}

iterator end() {
   return iterator(this->vector+this->current_size);
}

T operator[](int index) const {
   return this->vector[index];
}

void print() {
   for(iterator it = this->begin(); it!=this->end(); it++) {
      std::cout << (*it) << " ";
   }
   std::cout << "\n";
}
```

```
        void print(iterator first, iterator last) {
            for(iterator it = first; it!=last; it++) {
                std::cout << (*it) << " ";
            }
            std::cout << "\n";
        }



        //NaturalMergeSort
        void sort() {
            std::cout << "NaturalMergeSort\n";
            std::cout << "List: ";
            this->print();
            *this = this->split(this->begin(), this->end());
        }

        custom_vector<T> split(iterator begin, iterator end) {
            if(begin == end) return {};
            iterator first_sequence_end = find_end_of_sequence(begin, end);
            iterator second_sequence_end = find_end_of_sequence(first_sequence_end, end);
              custom_vector<T> first_part = merge(begin, first_sequence_end, first_sequence_end,
second_sequence_end);

            std::cout << "AFTER MERGE: ";
            first_part.print();

            custom_vector<T> second_part = split(second_sequence_end, end);
                        return merge(first_part.begin(), first_part.end(), second_part.begin(),
second_part.end());
        }

        iterator find_end_of_sequence(iterator begin, iterator end) {
            if(begin == end) return end;

            std::cout << "COMPARE: \n";

            for(begin; begin!=(end-1); begin++) {
                if(*begin > *(begin+1)) {
```

```
                std::cout << (*begin) << " > " << *(begin+1) << std::endl;
                std::cout << "RETURN " << *(begin+1) << std::endl;

                return begin+1;
            }
            std::cout << (*begin) << " <= " << *(begin+1) << std::endl;
        }
        return end;
    }

        custom_vector<T> merge(iterator first_start, iterator first_end, iterator second_start, iterator second_end) {
        std::cout << "MERGE\n";
        std::cout << "FIRST SEQUENCE: ";
        this->print(first_start, first_end);
        std::cout << "SECOND SEQUENCE: ";
        this->print(second_start, second_end);

        custom_vector<T> merged_seq;
        while(first_start < (first_end) && second_start < second_end) {
            iterator& tmp = *first_start <= *second_start? first_start : second_start;
            merged_seq.push_back(*tmp);
            tmp++;
        }

        iterator it_start = first_start == first_end? second_start: first_start;
        iterator it_end = first_start == first_end? second_end: first_end;

        for(it_start; it_start!=it_end; it_start++) {
            merged_seq.push_back(*it_start);
        }
        return merged_seq;
    }
};
```

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
```

```
#include <vector>
#include "custom_vector.h"
#include <sstream>

custom_vector<int> input();
custom_vector<int>  readListFromFile();
custom_vector<int>  readListFromTerminal();


int main() {
   // custom_vector<char> char_vector = {'c','b','a','w','g','a','x','f', 'y', 'a', 'z'};
   // char_vector.sort();
   // std::cout << "RESULT: ";
   // char_vector.print();

   // custom_vector<std::string> str_vector = {"a","qwe", "asd", "zxc", "fffff", "rqwe"};
   // str_vector.sort();
   // std::cout << "RESULT: ";
   // str_vector.print();

   custom_vector<int> list = input();
   list.sort();
   std::cout << "RESULT: ";
   list.print();
   return 0;
}


custom_vector<int>  input() {
      std::cout << "Lab 3(5) -> Проверить является ли текст правильной скобочной конструкцией\n";
   std::cout << "1) Input from file ('file.txt' by default)\n";
   std::cout << "2) Terminal input\n";
   std::cout << "Input type (1, 2): \n> ";

   int type = 0;
   do {
      std::cin >> type;
      if (std::cin.fail()) {
         std::cout << "Please enter an integer (1, 2)\n ";
```

```
            std::cin.clear();
            std::cin.ignore();
        }
    } while(type != 1 && type != 2);

    custom_vector<int> v;

    if(type == 1) v = readListFromFile();
    else if(type == 2) v = readListFromTerminal();
    return v;
}

custom_vector<int> readListFromFile() {
    custom_vector<int> v;
    std::ifstream infile ("file.txt");
    int x;
    while (infile >> x) {
        v.push_back(x);
    }
    return v;
}

custom_vector<int> readListFromTerminal() {
    custom_vector<int> v;
    std::string line;
    int x;
    std::cin.ignore();
    std::getline(std::cin, line);
    std::istringstream iss(line);
    while(iss >> x) {
        std::cout << x << std::endl;
       v.push_back(x);
    }
    return v;
}
```