**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Построение и Анализ Алгоритмов»**

**Тема: Сортировка слиянием**

**Вариант 7**

|  |  |  |
| --- | --- | --- |
| Студент гр. 5382 |  | Забалуев Н.Е. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017

**Цель работы**

Ознакомиться с основными понятиями и приёмами программирования сортировок, получить навыки программирования сортировок на языке программирования C++.

**Постановка задачи**

Написать сортировку слиянием как дополнение к проекту вычисления количества операций и времени работы различных сортировок в GitHub.

**Основные теоретические положения**

Сортировка слиянием — алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке. Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получается решение исходной задачи.

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.  
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по неубыванию подмассива. Тогда:  
3.2. Слияние двух подмассивов в третий результирующий массив.  
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.  
3.3.«Прицепление» остатка.  
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Достоинства:

* Работает даже на структурах данных последовательного доступа.
* Хорошо сочетается с подкачкой и кэшированием памяти.
* Неплохо работает в параллельном варианте: легко разбить задачи между процессорами поровну, но трудно сделать так, чтобы другие процессоры взяли на себя работу, в случае если один процессор задержится.
* Не имеет «трудных» входных данных.
* Устойчивая - сохраняет порядок равных элементов (принадлежащих одному классу эквивалентности по сравнению).

Недостатки:

* На «почти отсортированных» массивах работает столь же долго, как на хаотичных.
* Требует дополнительной памяти по размеру исходного массива.

**Спецификация программы**

*Назначение программы:*

Программа предназначена для сортировки массива слиянием.

*Описание программы:*

Входными данными для программы является длина массива, тип элементов и тип начальной сортировки массива.

Выходными данными являются:

1. Отсортированный массив;

2. Кол-во операций;

3. Скорость работы сортировки;

4. График зависимости времени и кол-ва операций от длины массива.

**Реализация**

*Функции:*

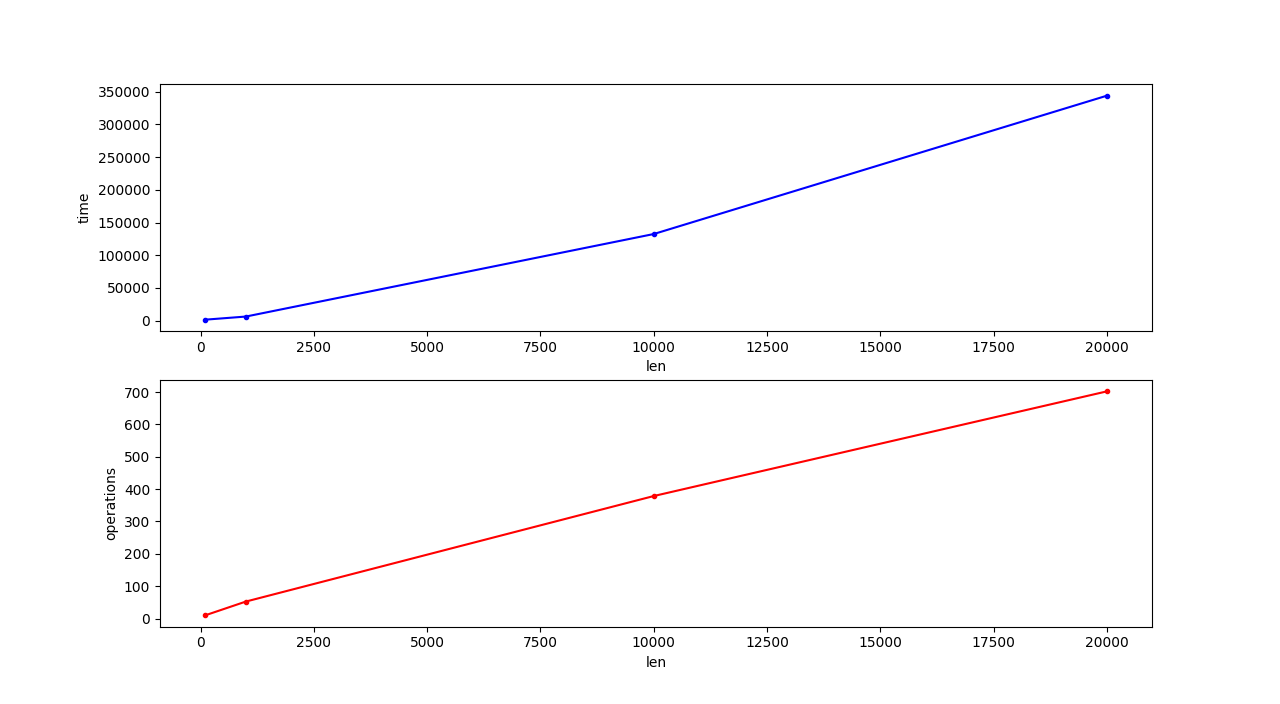
void merge(std::vector<T> &left, std::vector<T> &right, std::vector<T> &arr)

//слияние двух подмассивов в исходный

void sort (std::vector<T> &arr)

//Сортировка слиянием

**Результат работы программы**

****

*Рисунок 1. Построенные графики*

**Выводы.**

В результате выполнения лабораторной работы я ознакомился с основными понятиями и приёмами программирования сортировок, получил навыки программирования сортировок на языке программирования С++. Создал программу, удовлетворяющую всем требованиям и понял преимущества и недостатки сортировки слиянием.

**Приложение А. Исходный код.**

sort.h

#ifndef SORT\_H

#define SORT\_H

#include <vector>

#include <algorithm>

template<typename T>

void merge(std::vector<T> &left, std::vector<T> &right, std::vector<T> &arr) {

arr.clear();

while ((left.size()!=0)&&(right.size()!=0)) {

if (left.front()<=right.front()) {

arr.push\_back(left.front());

left.erase(left.begin());

}

else {

arr.push\_back(right.front());

right.erase(right.begin());

}

}

arr.insert(arr.end(), left.begin(), left.end());

arr.insert(arr.end(), right.begin(), right.end());

}

template<typename T>

void sort(std::vector<T> &arr) {

if (arr.size()>1) {

std::vector<T> left( arr.begin(), arr.begin()+(arr.size()>>1) );

sort(left);

std::vector<T> right( arr.begin()+(arr.size()>>1), arr.end() );

sort(right);

merge(left, right, arr);

}

}

#endif // SORT\_H