**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**

|  |  |  |
| --- | --- | --- |
| Студент гр. 5382 |  | Никитин В.А. |
| Преподаватель |  | Шолохова О.М. |

Санкт-Петербург

2017 г.

**Цель работы.**

Ознакомиться с основными сортировками, получить навыки программирования сортировок на языке программирования С++. Оценить эффективность работы алгоритма сортировки простым слиянием.

**Постановка задачи.**

Необходимо реализовать сортировку слиянием, исследовать эффективность работы алгоритма.

**Основные теоретические положения.**

**Сортировка слиянием** (*merge sort*) — алгоритм сортировки, который упорядочивает списки в определённом порядке. Эта сортировка — хороший пример использования принципа «разделяй и властвуй». 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.

Решение задачи сортировки выглядят так:

1. Сортируемый массив разбивается на две части примерно одинакового размера. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом.
3. Два упорядоченных массива половинного размера соединяются в один. Сравниваются два наименьших значения в массивах и на их основе строится третий массив, получая значения в порядке увеличения.

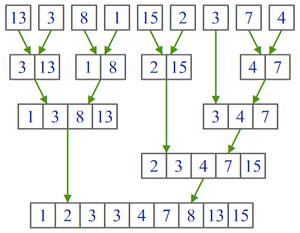


Рисунок 1. «Пример работы алгоритма»

Достоинства:

* Работает даже на структурах данных последовательного доступа.
* 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.

Худшее время: O(n logn).

Среднее время: O(n logn).

Лучшее время: O(n logn) обычно и O(n)на упорядоченном массиве.

Затраты памяти: O(n) вспомогательных.

Хорошо запрограммированная внутренняя сортировка слиянием работает немного быстрее пирамидальной, но медленнее быстрой, при этом требуя много памяти под буфер. Поэтому *mergeSort* используют для упорядочения массивов, лишь если требуется устойчивость метода (которой нет ни у быстрой, ни у пирамидальных сортировок).

Сортировка слиянием является одним из наиболее эффективных методов для односвязных списков и файлов, когда есть лишь последовательный доступ к элементам.

**Спецификация программы.**

*Назначение программы*.

Программа предназначена для сортировки массива методом слияния.

*Описание программы*.

Программа написана на языке C++. Входными данными для программы является количество элементов массива, левая и правая граница значений. Затем создается массив случайных чисел. Выходными данными является упорядоченный по возрастанию массив исходных чисел и данные о времени работы алгоритма.

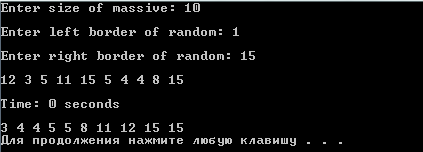
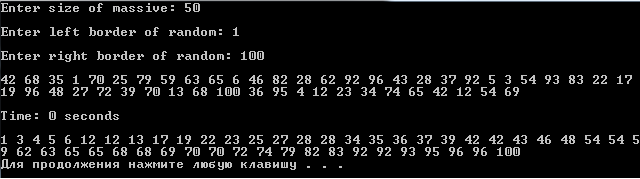
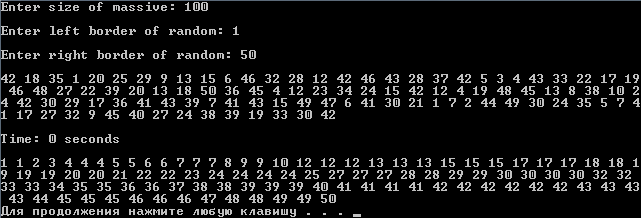


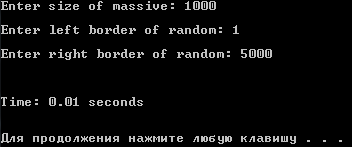
Рисунок 2. «Пример работы программы»

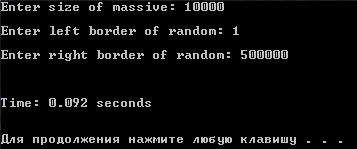
*Реализация*.

Функции:

* Void mergesort(vector<T> &arr, int left, int right> - основная функция, отвечающая за сам алгоритм сортировки. На вход подается сортируемый массив и его левая и правая границы. Результатом функции является отсортированный массив.
* Void input(vector<T> &arr, int size) – функция заполнения массива с вопросами о границах случайных чисел.
* Void output(vector<T> &arr, int size) – функция вывода текущего массива.

**Тестирование.**





**Исследование.**

Приведем таблицу, с подсчитанным временем работы в зависимости от длины последовательности.

*Таблица 1*

|  |  |
| --- | --- |
| Число элементов | Время работы |
| 10 | ~0 |
| 100 | ~0 |
| 1000 | 0.01 |
| 10000 | 0.092 |
| 100000 | 0.514 |
| 1000000 | 5.019 |

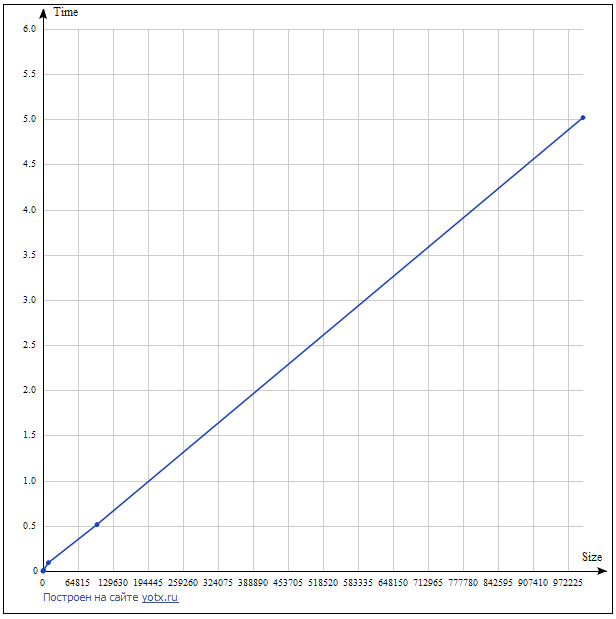


Рисунок 3. «График: Время от Размера»

**Выводы**

В ходе выполнения лабораторной работы был исследован алгоритм сортировки слиянием, было подсчитано время сортировки и построен график зависимости времени от размера массива.

**Приложение А. Исходный код.**

Sort.h

#include <vector>

#include <algorithm>

#include <cmath>

#include <iostream>

#include <time.h>

using namespace std;

template <typename T>

void mergesort(vector<T> &arr, int left, int right){ //left - левая граница исходного массива

if (right - left > 0){ //right - правая граница исходного массива

int delimiter = (right + left) / 2; //delimiter - медиана массива

mergesort(arr, left, delimiter); //разбиение на два меньших подмассива; итерирует до тех пор, пока размер подмассивов > 1

mergesort(arr, delimiter + 1, right);

if (right - left == 1){ //если всего два э-та, то сравниваются "номиналы" и при необходимости меняются местами

if (arr[left] > arr[right]) swap(arr[left], arr[right]);

return;

}

vector<T> helparr(&arr[left], &arr[right] + 1);

for (int idx = left, lidx = 0, ridx = delimiter - left + 1; idx <= right; ++idx){ //idx - счетчик текущего э-та строящегося массива

if (lidx > delimiter - left){ //lidx - счетчик пробега по "левому" массиву из двух

arr[idx] = helparr[ridx++]; //ridx - счетчик пробега по "правому" массиву из двух

}

else{ //если кончился левый или правый подмассив, то остаток другого подмассива прикрепляется к итоговому массиву

if (ridx > right - left){

arr[idx] = helparr[lidx++];

}

else{

if (helparr[ridx] < helparr[lidx]){ //добаление в массив очередного меньшего э-та

arr[idx] = helparr[ridx++];

}

else arr[idx] = helparr[lidx++];

}

}

}

}

else return;

}

template <typename T>

void sort(vector<T> &arr){

mergesort(arr, 0, (int)(arr.size() - 1));

}

template <typename T>

void input(vector<T> &arr, int size){

int a, b;

cout<<endl << "Enter left border of random: ";

cin >> a;

cout << endl << "Enter right border of random: ";

cin >> b;

for (int i = 0; i < size; ++i){

arr[i]= a + rand() % b;

}

return;

}

template <typename T>

void output(vector<T> &arr, int size){

for (int i = 0; i < size; ++i){

cout << arr[i] << " ";

}

}

LR1.cpp

#include "stdafx.h"

#include "Sort.h"

int \_tmain(int argc, \_TCHAR\* argv[])

{

int size = 0;

cout << "Enter size of massive: ";

cin >> size;

vector<int> arr(size);

input(arr, size);

cout << endl;

output(arr, size);

int start = clock();

sort(arr);

int end = clock();

cout <<endl<<endl<< "Time: " << (float)(end - start) / CLOCKS\_PER\_SEC<<" seconds";

cout << endl << endl;

output(arr, size);

cout << endl;

system("pause");

return 0;

}