**МИНОБРНАУКИ РОСИИ**

**САНКТ\_ПЕТЕРБУРГСКИЙ ГОСУДАРСВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**<<ЛЭТИ>> ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра MO ЭВМ**

**Тема лабораторной работы:**

**Реализация и исследование алгоритма сортировки TimSort**

Студент гр. 2300

Хидда Абдула

Санк-Петербург

2023

**Цель работы.**

Реализовать и исследовать алгоритма сортировки TimSort.

**Задание.**

1. Реализовать сортировку.
2. Провести исследование для различного размера данных.
3. Сравнить с теоретическими зависимостями и при различных значениях min\_run.

**Выполнение работы.**

**Реализованы функции:**

void timsort(vector <int>& a) – сама сортировка timsort, сортирующая вектор по ссылке.

int GetMinrun(int n) – функция возвращающая подходящее значение min\_run для заданной длины вектора.

bool comp(int a, int b) – функция возвращающая результат сравнения двух элементов.

void insrt\_sort(vector <int>& a, int first, int last) – сортировка вставками, использующаяся в алгоритме timsort, сортирует вектор с индекса first до индекса last.

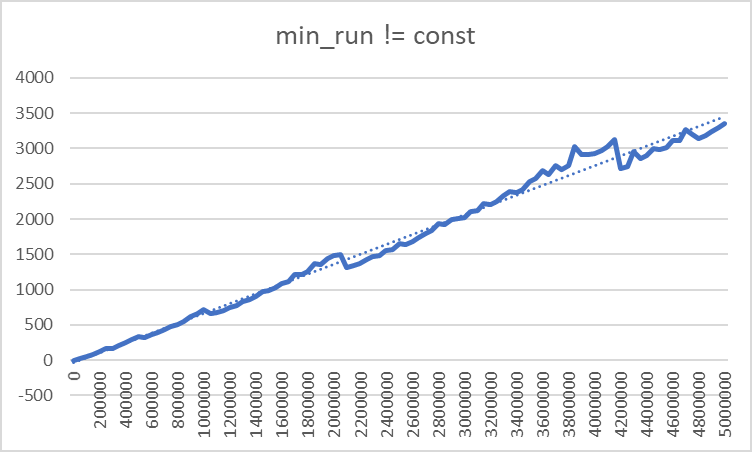
void merge\_sort(vector <int>& a, int f, int m, int l) – по сути сортировка слиянием, так же использующаяся в алгоритме timsort. В данной реализации эта функция соединяет два подряд идущих отсортированных подмассива. f – индекс началf первого подмассива, m – конца первого и начала второго, l – конец второго.

**В результате тестов получено:**

Данные использованные для построения графиков крайне объёмны и приведены в соответствующей папке на github.

Все данные для тестов генерировались библиотечной функцией rand(), по горизонтальной оси располагается размер массива, по вертикальной – время.

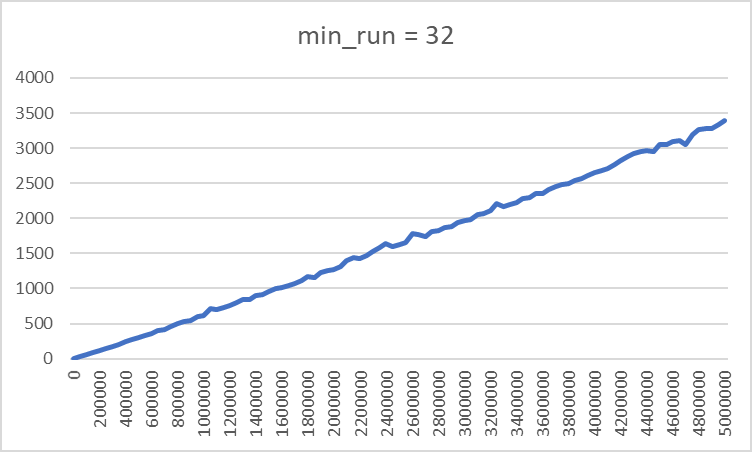
**Данный график был построен при изменяющемся значении min\_run в зависимости от размера массива:**



В целом можно увидеть, что алгоритм крайне правдоподобно работает за линейное время, лишь слегка отклоняясь от линии.

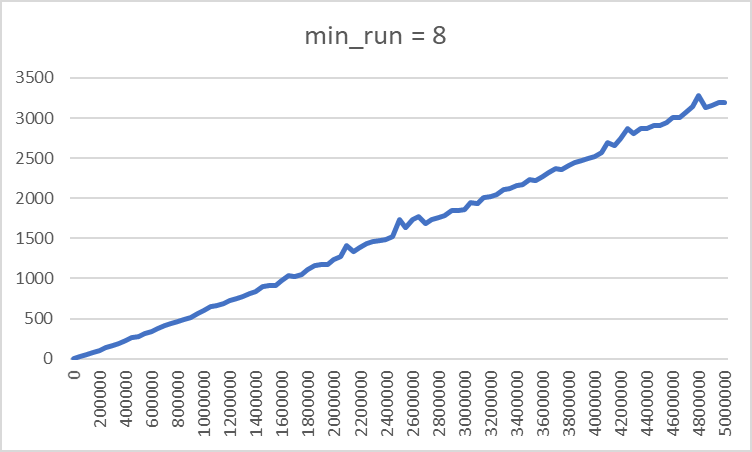
Можно заметить характерные ступеньки. Они связаны с изменением значения min\_run при сортировке большего количества данных.

**Данный график был построен при значении min\_run = 32:**



В результате мы видим всё равно крайне близкую в линейной зависимость.

**Данный график был построен при значении min\_run = 8:**



**Вывод:** Реализованная сортировка оправдывает предсказанную скорость работы, более того, в пределах данных размером до 10 000 000 скорость работы крайне близка к линейной.

**Приложение**

**Исходный код программы**

Название файла: main.cpp

#include <bits/stdc++.h>

#include "timsort.h"

#define STEP 50000

#define MAX\_N 5000000

using namespace std;

int main()

{

vector <int> n\_test(0);

int n = 0;

while (n <= MAX\_N){

n\_test.push\_back(n);

n += STEP;

}

bool is\_all\_ok = true;

std::ofstream out;

out.open("tests/result.txt");

for (int i = 0; i < n\_test.size(); i++){

vector <int> to\_timsort(n\_test[i]);

vector <int> to\_check(n\_test[i]);

for (int j = 0; j < n\_test[i]; j++){

to\_timsort[j] = rand() % MAX\_N;

to\_check[j] = to\_timsort[j];

}

sort(to\_check.begin(), to\_check.end());

clock\_t startTime, endTime;

startTime = clock( );

timsort(to\_timsort);

endTime = clock( );

cout << "N = " << n\_test[i] << ":\n";

cout << "\ttime - " << endTime - startTime << "\n";

bool is\_ok = true;

for (int j = 0; j < n\_test[i]; j++)

if (to\_timsort[j] != to\_check[j])

is\_ok = false;

if (is\_ok)

cout << "\tsort work correct\n";

else{

cout << "\tNOT CORRECT\n";

is\_all\_ok = false;

}

out << n\_test[i] << " " << endTime - startTime << "\n";

}

out.close();

if (is\_all\_ok)

cout << "\nALL TESTS WERE CORRECT\n";

else

cout << "\nSOME TEST GET INCORRECT ANSWER\n";

return 0;

}

Название файла: timsort.h

#pragma once

#include <iostream>

#include <vector>

using namespace std;

void timsort(vector <int>& a);

int GetMinrun(int n);

bool comp(int a, int b);

void insrt\_sort(vector <int>& a, int first, int last);

void merge\_sort(vector <int>& a, int f, int m, int l);

Название файла: timsort.cpp

#pragma once

#include "timsort.h"

#include <iostream>

#include <vector>

using namespace std;

void timsort(vector <int>& a){

int min\_run = GetMinrun(a.size());

for (int i = 0; i < a.size(); i = i + min\_run){

insrt\_sort(a, i, i + min\_run);

/\*

cout << "Part " << i / min\_run << ": ";

for (int j = i; j < min((int) a.size(), i + min\_run); j++)

cout << a[j] << " ";

cout << endl;

\*/

}

while (min\_run < a.size()){

for (int i = 0; i < a.size(); i += 2 \* min\_run)

merge\_sort(a, i, i + min\_run, i + 2 \* min\_run);

min\_run += min\_run;

}

}

int GetMinrun(int n){

//return 8;

int r = 0;

while (n >= 64) {

r |= n & 1;

n >>= 1;

}

return n + r;

}

bool comp(int a, int b){

if (a > b)

return true;

return false;

/\*

if (a \* a > b \* b)

return true;

if (a \* a == b \* b && a < b)

return true;

return false;

\*/

}

void insrt\_sort(vector <int>& a, int first, int last){

if (last > a.size())

last = a.size();

int counter = 0;

for (int i = first + 1; i < last; i++){

for (int j = i; j > first && comp(a[j - 1], a[j]); j--){

counter++;

swap(a[j], a[j - 1]);

}

}

}

void merge\_sort(vector <int>& a, int f, int m, int l){

if (m >= a.size())

return;

if (l > a.size())

l = a.size();

vector <int> result(l - f);

int i\_1 = f, i\_2 = m, i = 0;

while(i\_2 < l) {

if(comp(a[i\_1], a[i\_2]) || i\_1 == m){

result[i] = a[i\_2];

i++;

i\_2++;

}

else {

if (i\_1 < m) {

result[i] = a[i\_1];

i++;

i\_1++;

}

}

}

while (i\_1 < m) {

result[i] = a[i\_1];

i++;

i\_1++;

}

for (int i = 0; i < result.size(); i++)

a[f + i] = result[i];

}