**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ФКТИ**

отчет

**по лабораторной работе №2**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Сортировки

|  |  |  |
| --- | --- | --- |
| Студент гр. 3376 |  | Михайлов Н.Ф. |
| Преподаватель |  | Молдовян Д.Н. |

Санкт-Петербург

2024

**Лабораторная работа №2**

**СОРТИРОВКИ**

**Цель работы:** изучить логику работы различных методов сортировки, таких как сортировка вставками, сортировка слиянием, а также Timsort.Сортировку вставками и слиянием реализовать самостоятельно и с помощью них написать Timsort.

**Задание:** реализовать сортировку Timsort. В реализации должны быть выполнены все основные элементы алгоритма: сортировка вставками, поиск последовательностей run, подсчёт minrun, слияние последовательностей run, а также режим галопа при слиянии. Объединение отсортированных подмассивов в массив с использованием изменённого алгоритма сортировки слиянием.

**Базовый теоретический материал:**

1. **Сортировка слиянием** — это способ сортировки, который заключается в том, что мы делим массив, который нужно отсортировать на минимально возможные подмассивы, а потом попарно их «сливаем» вместе так, чтобы новый получившийся массив тоже был отсортирован.
2. **Сортировка вставками** — это способ сортировки, который заключается в том, что мы берём элемент и перемещаем его на n+1 позицию до тех пор, пока не найдём элемент, который меньше выбранного. Так делаем N раз.
3. **Режим галопа** — это ситуация, при которой при слиянии одна из последовательностей значительно лидирует. В этом случае мы не проверяем каждый элемент, а сливаем оставшуюся последовательность с основной в угоду быстроте.

**Алгоритм решения задачи:**

1. Вычисляем минимальную длину последовательности run, основываясь на длине самого массива.
2. Разбиваем изначальный массив на подмассивы длины mirun.
3. Сортируем эти последовательности с помощью InsertionSort().
4. После сортировки мы используем Merge(), чтобы сложить все последовательности mirun в один отсортированный массив.

**Трудоёмкость алгоритма:**

1. Merge()— O(m+n), т.к. мы, склеивая два массива вместе проходим по каждому из них.
2. GetMinRun()— ,т.к. для поиска minrun мы делим пополам.
3. InsertionSort() — в худшем и среднем и в лучшем случае.
4. IdentifyRuns() — в худшем случае.
5. Timsort — т.к. сначала вызывается IdentifyRuns, а потом идут слияния (у них логарифмическая сложность).

**Код алгоритма:**

**main.cpp**

#include <iostream>

#include <vector>

#include <climits>

#include <cmath>

#include <stack>

#include <algorithm>

// Функция слияния с поддержкой режима галопа

void Merge(std::vector<int>& array, int leftStart, int middle, int rightEnd) {

// Копируем левую и правую части исходного массива для слияния

std::vector<int> leftSubarray(array.begin() + leftStart, array.begin() + middle + 1);

std::vector<int> rightSubarray(array.begin() + middle + 1, array.begin() + rightEnd + 1);

int leftIndex = 0; // Текущая позиция в левом подмассиве

int rightIndex = 0; // Текущая позиция в правом подмассиве

int mergeIndex = leftStart; // Индекс для слияния в исходном массиве

int gallopThreshold = 7; // Порог для включения режима галопа

int leftGallopCount = 0; // Счётчик совпадений для включения галопа в левом подмассиве

int rightGallopCount = 0; // Счётчик совпадений для включения галопа в правом подмассиве

while (leftIndex < leftSubarray.size() && rightIndex < rightSubarray.size()) {

if (leftSubarray[leftIndex] <= rightSubarray[rightIndex]) {

array[mergeIndex++] = leftSubarray[leftIndex++];

leftGallopCount++;

rightGallopCount = 0; // Сбросить счётчик для правого подмассива

}

else {

array[mergeIndex++] = rightSubarray[rightIndex++];

rightGallopCount++;

leftGallopCount = 0; // Сбросить счётчик для левого подмассива

}

// Проверяем, нужно ли включить режим галопа для левого подмассива

if (leftGallopCount >= gallopThreshold) {

int rightLimitIndex = std::lower\_bound(rightSubarray.begin() + rightIndex, rightSubarray.end(), leftSubarray[leftIndex]) - rightSubarray.begin();

while (rightIndex < rightLimitIndex) {

array[mergeIndex++] = rightSubarray[rightIndex++];

}

leftGallopCount = 0; // Сбрасываем счётчик после режима галопа

}

// Проверяем, нужно ли включить режим галопа для правого подмассива

if (rightGallopCount >= gallopThreshold) {

int leftLimitIndex = std::lower\_bound(leftSubarray.begin() + leftIndex, leftSubarray.end(), rightSubarray[rightIndex]) - leftSubarray.begin();

while (leftIndex < leftLimitIndex) {

array[mergeIndex++] = leftSubarray[leftIndex++];

}

rightGallopCount = 0; // Сбрасываем счётчик после режима галопа

}

}

// Копируем оставшиеся элементы левого подмассива, если есть

while (leftIndex < leftSubarray.size()) {

array[mergeIndex++] = leftSubarray[leftIndex++];

}

// Копируем оставшиеся элементы правого подмассива, если есть

while (rightIndex < rightSubarray.size()) {

array[mergeIndex++] = rightSubarray[rightIndex++];

}

}

// Функция для получения minrun

size\_t GetMinRun(size\_t n) {

size\_t r = 0;

while (n >= 64) {

r |= n & 1;

n >>= 1;

}

return n + r;

}

// Функция сортировки вставками

template <typename T>

void InsertionSort(std::vector<T>& array, size\_t startIdx, size\_t endIdx) {

for (size\_t currentIdx = startIdx + 1; currentIdx <= endIdx; currentIdx++) {

T currentValue = array[currentIdx];

size\_t positionIdx = currentIdx;

// Перемещаем элементы, которые больше currentValue, на одну позицию вперёд

while (positionIdx > startIdx && array[positionIdx - 1] > currentValue) {

array[positionIdx] = array[positionIdx - 1];

positionIdx--;

}

// Вставляем currentValue на его правильное место

array[positionIdx] = currentValue;

}

}

// Поиск run и его добавление в стек

std::vector<std::pair<int, int>> IdentifyRuns(std::vector<int>& array, size\_t minrun) {

std::vector<std::pair<int, int>> runs;

size\_t i = 0;

while (i < array.size()) {

size\_t run\_start = i;

while (i + 1 < array.size() && array[i] <= array[i + 1]) {

i++;

}

while (i + 1 < array.size() && array[i] > array[i + 1]) {

i++;

}

size\_t run\_end = std::min(run\_start + minrun - 1, array.size() - 1);

InsertionSort(array, run\_start, run\_end);

runs.emplace\_back(run\_start, run\_end);

i = run\_end + 1;

}

return runs;

}

// Основная функция Timsort

template <typename T>

void Timsort(std::vector<T>& array) {

size\_t array\_len = array.size();

size\_t minrun = GetMinRun(array\_len);

// Шаг 1: Определяем run и сортируем их

auto runs = IdentifyRuns(array, minrun);

// Шаг 2: Слияние отсортированных run

while (runs.size() > 1) {

std::vector<std::pair<int, int>> new\_runs;

for (size\_t i = 0; i < runs.size(); i += 2) {

if (i + 1 < runs.size()) {

Merge(array, runs[i].first, runs[i].second, runs[i + 1].second);

new\_runs.emplace\_back(runs[i].first, runs[i + 1].second);

}

else {

new\_runs.push\_back(runs[i]);

}

}

runs = new\_runs;

}

}

int main() {

std::vector<int> array;

for (int i = 0; i < 15; i++) {

array.push\_back(rand() % 10); // генерируем массив случайных чисел

printf("%d ", array[i]);

}

Timsort(array);

std::cout << "\n\nОтсортированный массив:\n";

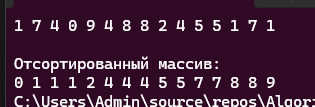
for (int i = 0; i < array.size(); i++) {

printf("%d ", array[i]);

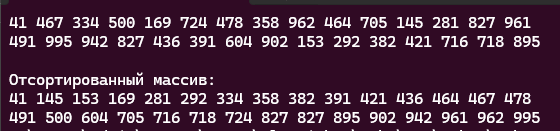
}

return 0;

}

** Примеры работы программы:**

*рис.1.Массив случайных чисел и его отсортированная версия.*

**

*рис.2. Массив случайных чисел и его отсортированная версия.*

**Вывод:**

В ходе выполнения лабораторной работы было достигнуто понимание работы таких алгоритмов сортировки как: сортировка слиянием, сортировка вставками и Timsort. Так же в работы были реализованы эти сортировки или их части, если того требовало задание.