**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ФКТИ**

**отчет**

**по лабораторной работе №3**

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

**Тема: Сортировка timsort**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3376 |  | Константинов Р.И. |
| Преподаватель |  | Молдовян Д.Н. |

Санкт-Петербург

2024

**Тема**

Реализация алгоритма сортировки Timsort с учётом всех основных элементов алгоритма: сортировка вставками, поиск последовательности run, подсчёт minrun, слияние последовательностей run, режим галопа при слиянии.

**Цель работы**

Целью работы является разработка и реализация эффективного алгоритма сортировки Timsort, который сочетает в себе элементы сортировки вставками и сортировки слиянием, а также включает режим галопа для оптимизации работы на больших наборах данных.

**Задание**

Реализовать алгоритм сортировки Timsort, включающий следующие элементы:

* Сортировка вставками.
* Выделение последовательностей run.
* Подсчёт минимального размера подмассива (minrun).
* Слияние последовательностей run.
* Использование режима галопа для ускорения слияния.

**Алгоритм решения задачи**

1. **Сортировка вставками**: Используется для сортировки небольших подмассивов длиной RUN. Это базовая сортировка, которая эффективно работает на малых данных.
2. **Поиск последовательностей run**: Последовательности run представляют собой упорядоченные (по возрастанию или убыванию) подмассивы, которые являются основными строительными блоками алгоритма Timsort.
3. **Подсчёт minrun**: Минимальный размер подмассива рассчитывается на основе длины исходного массива, чтобы уменьшить число операций слияния.
4. **Слияние последовательностей**: Подмассивы (run) объединяются с использованием изменённого алгоритма сортировки слиянием. Для ускорения применяется режим галопа.
5. **Режим галопа**: Режим оптимизирует процесс слияния подмассивов путём выполнения бинарного поиска, что позволяет быстро находить место вставки.
6. **Объединение подмассивов**: Итеративное слияние подмассивов выполняется, пока не останется один общий отсортированный массив.

**Трудоёмкость алгоритма**: В худшем и среднем случаях сложность алгоритма составляет O(n\*log (n)), а в лучшем случае — O(n), если массив уже частично отсортирован.

**Текст программы**

#include <iostream>

#include <random>

#include <iostream>

#include <vector>

// Минимальный размер подмассива для вставки

const int RUN = 32;

// Функция сортировки вставками

void insertionSort(std::vector<int>& arr, int left, int right) {

for (int i = left + 1; i <= right; i++) { // Перебор слева

int temp = arr[i];

int j = i - 1;

while (j >= left && arr[j] > temp) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = temp;

}

}

// Функция слияния двух отсортированных подмассивов

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

int len1 = mid - left + 1, len2 = right - mid;

std::vector<int> leftArr(len1), rightArr(len2);

for (int i = 0; i < len1; i++)

leftArr[i] = arr[left + i];

for (int i = 0; i < len2; i++)

rightArr[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

while (i < len1 && j < len2) {

if (leftArr[i] <= rightArr[j]) {

arr[k] = leftArr[i];

i++;

}

else {

arr[k] = rightArr[j];

j++;

}

k++;

}

while (i < len1) {

arr[k] = leftArr[i];

i++;

k++;

}

while (j < len2) {

arr[k] = rightArr[j];

j++;

k++;

}

}

// Основная функция Timsort

void timSort(std::vector<int>& arr) {

int n = arr.size();

// Сортировка подмассивов размером RUN вставками

for (int i = 0; i < n; i += RUN)

insertionSort(arr, i, std::min((i + RUN - 1), (n - 1)));

// Слияние подмассивов увеличивающегося размера

for (int size = RUN; size < n; size = 2 \* size) {

for (int left = 0; left < n; left += 2 \* size) {

int mid = left + size - 1;

int right = std::min((left + 2 \* size - 1), (n - 1));

if (mid < right)

merge(arr, left, mid, right);

}

}

}

void mergeGalloping(std::vector<int>& arr, int left, int mid, int right) {

int len1 = mid - left + 1, len2 = right - mid;

std::vector<int> leftArr(len1), rightArr(len2);

for (int i = 0; i < len1; i++)

leftArr[i] = arr[left + i];

for (int i = 0; i < len2; i++)

rightArr[i] = arr[mid + 1 + i];

int i = 0, j = 0, k = left;

while (i < len1 && j < len2) {

if (leftArr[i] <= rightArr[j]) {

arr[k++] = leftArr[i++];

}

else {

arr[k++] = rightArr[j++];

}

// Галопирование

if (i < len1 && j < len2 && leftArr[i] > rightArr[j]) {

auto gallopRight = [&]() {

int low = j, high = len2;

while (low < high) {

int mid = low + (high - low) / 2;

if (rightArr[mid] < leftArr[i])

low = mid + 1;

else

high = mid;

}

return low;

};

int gallopIndex = gallopRight();

for (; j < gallopIndex; ++j) {

arr[k++] = rightArr[j];

}

}

if (i < len1 && j < len2 && rightArr[j] >= leftArr[i]) {

auto gallopLeft = [&]() {

int low = i, high = len1;

while (low < high) {

int mid = low + (high - low) / 2;

if (leftArr[mid] <= rightArr[j])

low = mid + 1;

else

high = mid;

}

return low;

};

int gallopIndex = gallopLeft();

for (; i < gallopIndex; ++i) {

arr[k++] = leftArr[i];

}

}

}

while (i < len1) {

arr[k++] = leftArr[i++];

}

while (j < len2) {

arr[k++] = rightArr[j++];

}

}

// Основная функция Timsort галопом

void timSortGalloping(std::vector<int>& arr) {

int n = arr.size();

// Сортировка подмассивов размером RUN вставками

for (int i = 0; i < n; i += RUN)

insertionSort(arr, i, std::min((i + RUN - 1), (n - 1)));

// Слияние подмассивов увеличивающегося размера

for (int size = RUN; size < n; size = 2 \* size) {

for (int left = 0; left < n; left += 2 \* size) {

int mid = left + size - 1;

int right = std::min((left + 2 \* size - 1), (n - 1));

if (mid < right)

mergeGalloping(arr, left, mid, right);

}

}

}

// Пример использования

int main() {

std::vector<int> arr = { 43, 21, 7, 23, 19, 15, 2, 10, 12, 9, 1, 33, 54, 12, 56, 32, 82, 432, 43, 53, 123, 43, 63, 23 };

std::cout << "unsorted arr: ";

for (int x : arr) std::cout << x << " ";

timSortGalloping(arr);

std::cout << "sorted arr: ";

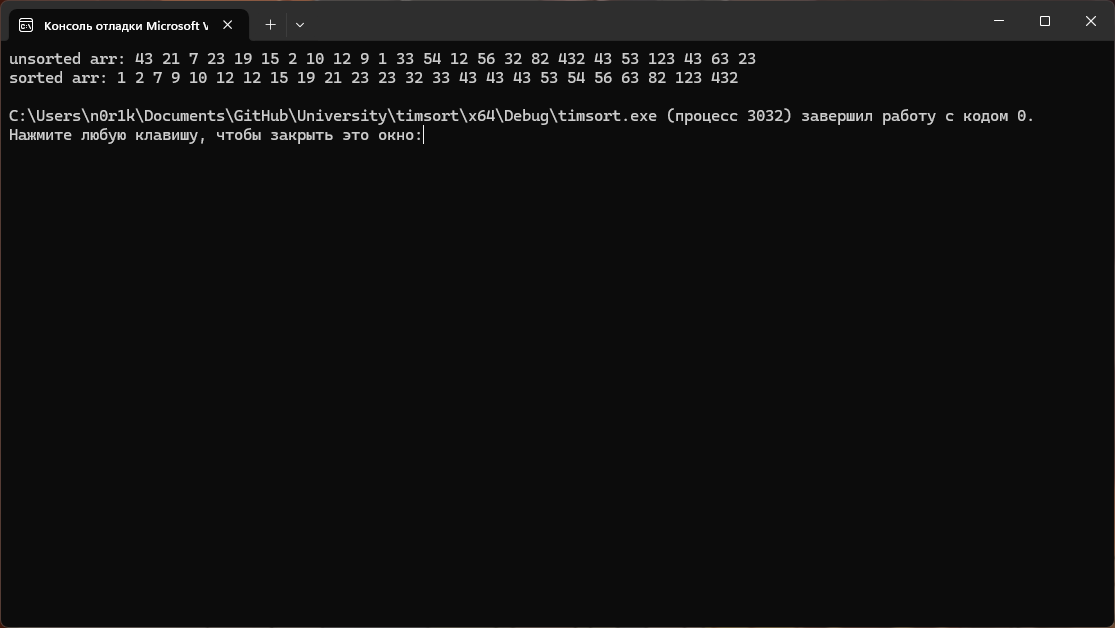
for (int x : arr) std::cout << x << " ";

std::cout << std::endl;

return 0;

}

**Результаты работы программы**



**Выводы**

Алгоритм Timsort успешно реализован. Он показал высокую производительность благодаря использованию сортировки вставками на малых подмассивах и режима галопа при слиянии. Реализация позволяет эффективно сортировать массивы с различной степенью упорядоченности, достигая наилучших показателей времени работы для частично отсортированных данных.