Министерство образования и науки РФ
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

Лабораторная работа №4

“Внутренние сортировки”

Работу выполнил студент
Кладковой Максим Дмитриевич
Группа: 5130904/30005
Руководитель: Червинский Алексей Петрович

Содержание

Общая постановка задачи:

Для выполнения индивидуального задания по внутренним сортировкам нужно:
• Разработать алгоритм решения индивидуальной задачи.
• Реализовать алгоритм на языке C++:
• написать программу,
• выполнить отладку и тестирование программы.
• Подготовить отчет по заданию, состоящий из разделов:
1. Общая постановка задачи
2. Описание алгоритма решения
3. Сравнение результаты экспериментальной оценки временной сложности с теоретическими для массивов, состоящих из 1000, 10000, 100000 и 500000 элементов.
Приложение 1. Код программы.
Замечание:
Выполнить тесты для лучшего, худшего и среднего случаев.
Для тестирования использовать функцию, проверяющую упорядоченность последовательности.
Для генерации тестов среднего случая можно использовать псевдослучайные числа.

Описание алгоритма решения:

Основная идея Tim Sort — использовать существующий порядок данных для минимизации количества сравнений и замен. Это достигается путем разделения массива на небольшие подмассивы, называемые Run(ами), которые уже отсортированы, а затем объединения этих Run(ов) с использованием модифицированного алгоритма сортировки слиянием.

Алгоритм:

В качестве примера рассмотрим массив: [*4, 2, 8, 6, 1, 5, 9, 3, 7*]

Шаг 1. Определите размер тиража

- *Минимальный размер запуска: 32 (мы проигнорируем этот шаг, поскольку наш массив небольшой)*

Шаг 2. Разделите массив на прогоны

- *На этом этапе мы будем использовать сортировку вставкой для сортировки небольших подпоследовательностей (серий) внутри массива.*
- *Исходный массив: [4, 2, 8, 6, 1, 5, 9, 3, 7]*
- *Начальных запусков нет, поэтому мы создадим их с помощью сортировки вставками.*
- *Отсортированные прогоны: [2, 4], [6, 8], [1, 5, 9], [3, 7]*
- *Обновленный массив: [2, 4, 6, 8, 1, 5, 9, 3, 7]*

Шаг 3. Объедините прогоны

•*На этом этапе мы объединим отсортированные серии, используя модифицированный алгоритм сортировки слиянием.*

•*Объединяйте прогоны, пока весь массив не будет отсортирован.*

•*Объединенные прогоны: [2, 4, 6, 8], [1, 3, 5, 7, 9]*

•*Обновленный массив: [2, 4, 6, 8, 1, 3, 5, 7, 9]*

Шаг 4. Отрегулируйте размер тиража

После каждой операции слияния мы удваиваем размер серии, пока он не превысит длину массива.

•*Размер серии удваивается: 32, 64, 128 (мы проигнорируем этот шаг, поскольку наш массив небольшой)*

Шаг 5. Продолжить объединение

•*Повторяйте процесс слияния, пока весь массив не будет отсортирован.*

•*Финальный объединенный прогон: [1, 2, 3, 4, 5, 6, 7, 8, 9]*

Сложность во времени:

Best Case	O(n)
Average Case	O(n log n)
Worst Case	O(n log n)

Требования и тест план.

Требование	Данные	Ожидаемый результат
Реализация сортировки tim sort.	rand()	Вывод результатов экспериментальной оценки временной сложности для массивов из 1000, 10000, 100000 и 500000 элементов в консоль.

Приложение A. Скриншоты работы программы.

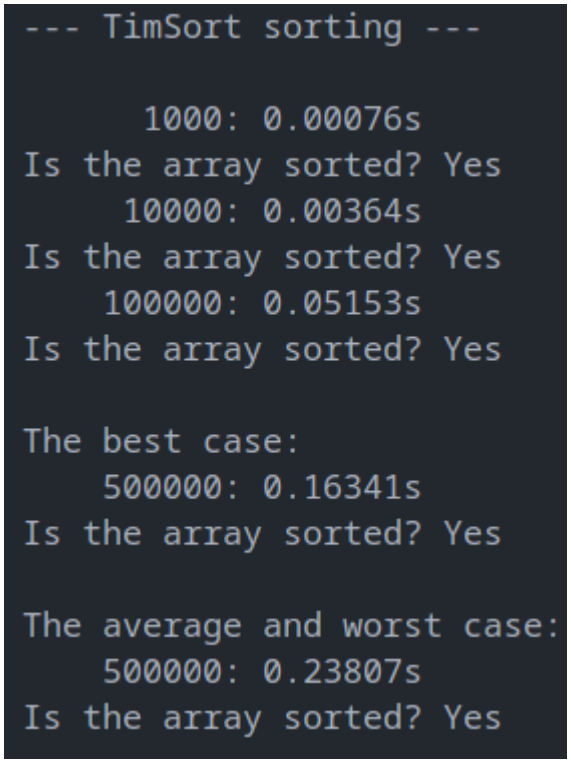

Рис 1: Экспериментальная оценка временной сложности.

Теоретическая оценка временной сложности.
Для массива из 1000 элементов: O(n log n) = O(1000 * log(1000)) = O(9965,78)
Для массива из 10000 элементов: O(n log n) = O(10000 * log(10000)) = O(132877,12)
Для массива из 100000 элементов: O(n log n) = O(100000 * log(100000)) = O(1,66 * 10^6)
Для массива из 500000 элементов: O(n log n) = O(500000 * log(500000)) = O(9,47 * 10^6)
Теоретическая сложность не коррелирует с практической. Возможно дело в недостаточно рандомной генерации массива чисел.

Приложение B. Код программы.

main.cpp

```
#include "Timsort.hpp"

int main()
{
test();
return 0;
}
```

test,cpp

```
#include "Timsort.hpp"
#include <time.h>
#include <chrono>
#include <iomanip>

void time(int* array, std::size_t size)
{
auto start = std::chrono::high_resolution_clock::now();
tim_sort(array, size);
auto end = std::chrono::high_resolution_clock::now();
std::chrono::duration<double> seconds = end - start;
std::cout << std::setw(10) << std::setprecision(5) << std::fixed << size << ": " << seconds.count() <<
"s" << std::endl;
}

void fillArrayWithRandom(int* array, size_t size)
{
for (size_t i = 0; i < size; ++i)
{
int x = rand() * 100000 + rand();
array[i] = x;
}
}

bool isOrder(int* array, std::size_t size)
{
bool isOrder = true;
for (std::size_t i = 0; i < size - 1; i++)
{
if (array[i] > array[i + 1])
{
isOrder = false;
}
}
return isOrder;
}
```

```
void test()
{
std::cout << "--- TimSort sorting ---" << std::endl << std::endl;
for (size_t size : {1000, 10000, 100000})
{
int* array = new int[size];
fillArrayWithRandom(array, size);
time(array, size);
std::cout << "Is the array sorted? " << (isOrder(array, size)? "Yes" : "No" ) << std::endl;
delete [] array;
}
std::cout << std::endl << "The best case: " << std::endl;
int* array = new int[500000];
for(std::size_t i; i < 500000; i++)
{
array[i] = i;
}
time(array, 500000);
std::cout << "Is the array sorted? " << (isOrder(array, 500000)? "Yes" : "No" ) << std::endl;
std::cout << std::endl << "The average and worst case: " << std::endl;
fillArrayWithRandom(array, 500000);
time(array, 500000);
std::cout << "Is the array sorted? " << (isOrder(array, 500000)? "Yes" : "No" ) << std::endl;

delete [] array;
}
```

Timsort.hpp

```
#ifndef TIMSORT_HPP
#define TIMSORT_HPP
const int RUN = 32;

#include <iostream>

void merge(int* result, std::size_t begin, std::size_t mid, std::size_t end);

void insertion_sort(int* array, std::size_t begin, std::size_t end);

void tim_sort(int* array, std::size_t size);

void test();

void time(int* array, std::size_t size);
```

```
void fillArrayWithRandom(int* array, size_t size);

#endif
```

Timsort.cpp

```
#include "Timsort.hpp"

void merge(int* result, std::size_t begin, std::size_t mid, std::size_t end)
{
std::size_t lenLeft = mid - begin + 1;
std::size_t lenRight = end - mid;
int* left = new int[lenLeft];
int* right= new int[lenRight];

for (std::size_t i = 0; i < (lenLeft); i++)
left[i] = result[begin+i];
for (std::size_t j = 0; j < (lenRight); j++)
right[j] = result[mid + j + 1];

std::size_t leftIn = 0, rightIn = 0, resIn = begin;
while (leftIn != lenLeft && rightIn != lenRight)
{
if(left[leftIn] <= right[rightIn])
result[resIn++] = left[leftIn++];
else
result[resIn++] = right[rightIn++];
}

while (leftIn < lenLeft)
result[resIn++] = left[leftIn++];

while (rightIn < lenRight)
result[resIn++] = right[rightIn++];

delete [] left;
delete [] right;
}

void insertion_sort(int* array, std::size_t begin, std::size_t end)
{
for(std::size_t i = begin + 1; i <= end; i++)
{
std::size_t j = i - 1;
int key = array[i];
while((array[j] > key) && (j >= begin))
```

```
{
array[j+1]= array[j];
if(j == 0)
break;
j--;
}
if(j==0)
array[j] = key;
else
array[j+1] = key;
}
}

void tim_sort(int* array, std::size_t size)
{
for (std::size_t i = 0; i < size; i += RUN)
{
std::size_t right = std::min((i + RUN - 1), (size - 1));
bool isSorted = true;
for (std::size_t j = i; j< right; j++)
{
if (array[j] > array[j+1])
{
isSorted = false;
break;
}
}

if (!isSorted)
insertion_sort(array, i, right);
}

for (std::size_t j = RUN; j < size; j = 2 * j)
{
for (std::size_t begin = 0; begin < size; begin += (2 * j))
{
std::size_t mid = begin + j - 1;
std::size_t end = std::min((begin + 2 * j - 1), (size - 1));

if (mid < end)
{
merge(array, begin, mid, end);
}
}
}
}
```

Заключение

В результате выполнения данной работы я разработал алгоритм для сортировки Timsort на языке C++, сравнил теоретическую и экспериментальную временную сложность.