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

Лабораторная работа
Предмет: практикум по программированию
Тема: внутренние сортировки
Вариант 13: TimSort

Работу выполнила студентка
Груздева Анна Сергеевна
Группа: 5130904/30005
Руководитель:
Червинский Алексей Петрович

Санкт-Петербург
2024

Содержание

Задание:

1. Разработать алгоритм решения индивидуальной задачи.
2. Реализовать алгоритм на языке C++:
   2.1. написать программу,
   2.2. выполнить отладку и тестирование программы.
3. Подготовить отчет по заданию, состоящий из разделов:
   3.1. общая постановка задачи,
   3.2. описание алгоритма решения,
   3.3. сравнение результатов экспериментальной оценки временной сложности с теоретическими для массивов, состоящих из 1000, 10000, 100000 и 500000 элементов.
   3.4. Приложение 1. Код программы.

Цели:

1. Реализованный алгоритм сортировки TimSort (по не убыванию)
2. Сравнение результатов экспериментальной оценки временной сложности с теоретическими для массивов, состоящих из 1000, 10000, 100000 и 500000 элементов.

Задачи:

1. Реализовать алгоритм сортировки TimSort (по не убыванию)
2. Выполнить отладку и тестирование программы
3. Теоретически оценить временную сложность алгоритма для массивов состоящих из 1000, 10000, 100000 и 500000 элементов
4. Экспериментально оценить временную сложность алгоритма для массивов состоящих из 1000, 10000, 100000 и 500000 элементов
5. Сравнить экспериментальную и теоретическую временные сложности, полученные в п. 3 и 4

Алгоритм решения:

TimSort - гибридный алгоритм сортировки, совмещающий сортировку слиянием (merge sort) и сортировку вставкой (insertion sort) и хорошо работающий на многих видах реальных данных, пользуясь тем, что многие части отсортированы изначально.

Принцип работы алгоритма:

1. Разделить массив на маленькие части (раны - eng. runs) и отсортировать каждую из них, используя сортировку вставкой. Размер рана выбирается вручную, зачастую это 32 или 64, так на массивах такого размера или меньше сортировка вставкой эффективна.
2. Соединить части используя слияние.
   Оптимизации:
1. Галлоп. Если наименьший элемент одного рана больше максимального элемента другого рана, то сравнение остальных элементов избыточно. Сначала добавляются все элементы меньшего из ранов, потом - большего.

Теоретическая оценка сложности:

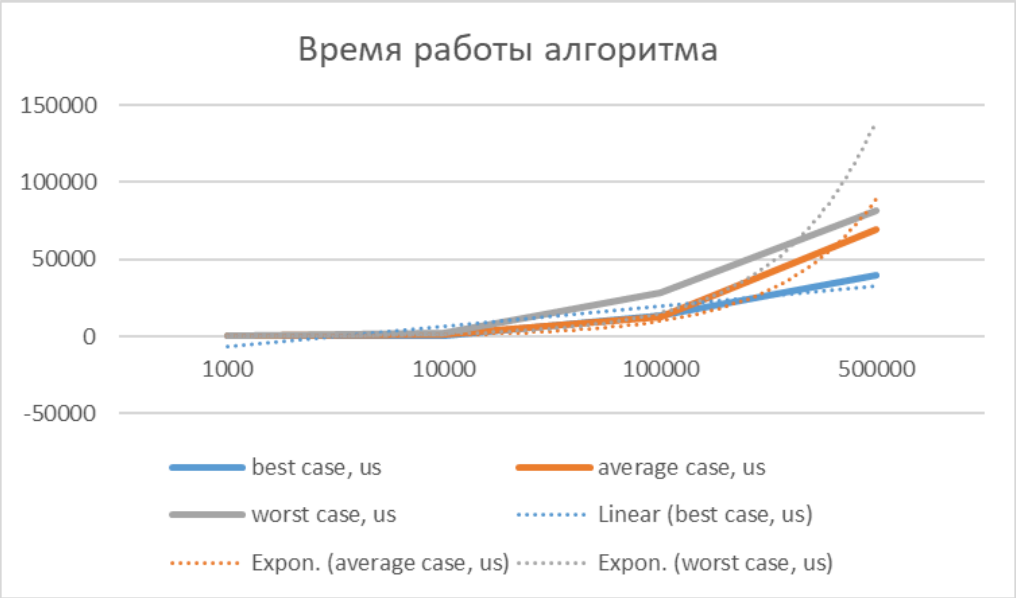
n - количество элементов в массиве
Лучший случай (массив уже отсортирован): Ω(n)
Средний случай: θ(nlog)
Худший случай (массив в обратном порядке): O(nlogn)

1. n = 1000: Ω(1000) θ(9966) O(9966)
2. n = 10000: Ω(10000) θ(132877) O(132877)
3. n = 100000: Ω(100000) θ(1660960) O(1660960)
4. n = 500000: Ω(500000) θ(9465780) O(9465780)

Практическая оценка временной сложности:

n - количество элементов в массиве

n	best case, us	average case, us	worst case, us
1000	116	98	111
10000	782	1376	2588
100000	13174	12984	27977
500000	39473	69628	81243



Как видно на графике, теоретические и практические данные совпадают (в пределах погрешности), то есть в лучшем случае - при уже отсортированном массиве - зависимость времени работы алгоритма от количества элементов в массиве - линейная, в среднем и худшем случаях - nlogn.

Вывод:

В работе была изучена сортировка TimSort, написан алгоритм, измерено время работы алгоритма на различных размерах массива и случаях изначальной сортировки данных, а также дано сравнение с теоретическими значениями.

Приложения:

1. Приложение 1. Код программы