САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики – процессов управления

Лабораторная работа по курсу

«Алгоритмы и анализ сложности»

на тему «Эмпирический анализ алгоритма сортировки слиянием»

Выполнил студент
3-го курса, группы 18.Б13-пу
Усеинов Л.В

Санкт-Петербург, 2020

Описание алгоритма сортировки слиянием

Алгоритм сортировки слиянием (merge sort) является классическим примером реализации парадигмы “разделяй и властвуй”.

Он был изобретен Джоном фон Нейманом в 1945 году во время проведения тестов на адекватность поведения некоторых кодовых слов, предложенных им для EDVAC [1].

Интуитивно алгоритм сортировки слиянием работает следующим образом:

1. **Разделение.** Делим *n*-элементную сортируемую последовательность на две подпоследовательности по *n/2* элементов.
2. **Властвование.** Рекурсивно сортируем две подпоследовательности с использованием сортировки слиянием (если подпоследовательность имеет единичную длину, то считаем, что она уже отсортирована).
3. **Комбинирование.** Соединяем две отсортированные подпоследовательности для получения упорядоченного массива [2].

За счёт большого потенциала для распараллеливания, модификации merge sort находят применение в расчётах на вычислительных кластерах [3]. Кроме того, в различных языках программирования сыскали популярность адаптивные алгоритмы сортировки на базе Timsort [4] — идейного продолжателя сортировки слиянием.

Математический анализ алгоритма [2]

Ключевыми этапами в алгоритме сортировки слиянием являются **разделение**, **властвование** (выполнение сортировки подмассивов) и **комбинирование**.

Далее, для упрощения априорного анализа алгоритма будем считать, что количество входных элементов в массиве представляет собой степень двойки. В книге Кормена показано [2], что данное предположение не влияет на порядок роста, получаемый в результате решения рекуррентного уравнения.

Введём следующие обозначения: $T(n)$ — временная сложность алгоритма при входных данных размера n, $D(n)$ — время, затрачиваемое на разбиение на вспомогательные подзадачи при входных данных размера

n, $C(n)$ — время, требуемое на объединение подзадач при входных данных размера n.

Теперь попробуем оценить время работы для каждого из этапов алгоритма для произвольного $n > 1$, где n — количество элементов в входном массиве, тогда:

1. **Разделение.** В ходе разбиения определяется, где находится середина массива путём деления длины массива нацело на 2. Очевидно, что эта операция выполняется за константное время $D(n) = \Theta(1)$.
2. **Властвование.** Рекурсивно решается две подзадачи, размер каждой из которых составляет $n/2$. Соответственно, решения этих подзадач равно $2T(n/2)$.
3. **Комбинирование.** Операция представляет собой объединение двух отсортированных массивов в один. Для этого мы заведём три указателя: два — на упорядоченные массивы, и один — на результирующий. Установим их на начала соответствующих массивов. Будем на каждой итерации сравнивать элементы отсортированных последовательностей, на которые ссылаются указатели (без потери общности предположим, что сортируем последовательность по возрастанию). Если элемент первого массива больше элемента второго, то записываем значение второго в результирующий массив и увеличиваем значение счетчиков, соответствующих массивов (не включая первый). Иначе, делаем то же самое с первым (не включая второй). Как только один из упорядоченных массивов заканчивается, просто дописываем оставшиеся элементы другого в конец результирующей последовательности, завершая тем самым данный этап. Наиболее времязатратная операция на данном шаге — сравнение. Отметим, что мы делаем максимум n сравнений за время комбинирования. Следовательно, сложность операции $C(n) = \Theta(n)$.

Из приведённых выше оценок получаем следующее рекуррентное соотношение:

$$T(n) = \Theta(1), \text{ если } n = 1,$$
$$T(n) = 2T(n/2) + \Theta(n), \text{ если } n > 1$$

Решая его (при помощи метода деревьев рекурсии или основной теоремы), имеем — $T(n) = \Theta(nlog(n))$.

Описание характеристик входных данных

Так как сортировка слиянием является одним из классических алгоритмов сортировки, то используется он довольно часто в проектах самого разного типа и размера (особенно, если известен тот факт, что данные приходят в частично отсортированном виде). Однако, нужно учитывать тот факт, что merge sort (не модифицированный) требует $\Theta(n)$ дополнительной памяти для входного массива длиной n. И если для кластерных систем с большим объёмом памяти это может быть не так критично, то для пользовательских приложений данное требование имеет большой вес.

Таким образом имеет смысл сравнивать работу алгоритма и на сравнительно маленьком количестве входных элементов — 10^2, 10^3, 10^4; и на большом — 10^5, 10^6, 10^7.

В качестве типа входных данных будут использоваться вещественные числа одинарной точности. Этот тип данных включает в себя наиболее полный и относительно часто используемый числовой диапазон, на настоящий момент.

Описание способа измерения трудоемкости алгоритма

С целью измерения сложности алгоритма будут подсчитываться количества операций сравнения и времени, затрачиваемые на сортировку. Для каждого размера входных данных будут проводиться 100 запусков merge sort с разными рандомно-генерируемыми данными, результаты по которым будут в дальнейшем усредняться и рассматриваться в качестве оценки трудоемкости работы алгоритма на входных последовательностях разного объёма соответственно.

Способ генерации входных данных

Генератор принимает на вход размер массива и при помощи стандартной python библиотеки random генерирует массив чисел величины *input_size*, заполненный числами типа *CustomFloat32* со встроенным подсчетом операций сравнения (Рис. 1).

```
class InputGenerator:
    def __init__(self, input_size):
        self.input_size = input_size

    def __call__(self):
        arr = []
        for i in range(self.input_size):
            num = random.uniform(FLOAT32_MIN, FLOAT32_MAX)
            num = CustomFloat32(num)
            arr.append(num)
        return arr
```

Рис. 1 (Код генератора входных данных)

Реализация алгоритма

Реализация алгоритма взята с сайта *GeekForGeeks* (Рис. 2). Код также был проверен и протестирован на наличие возможных ошибок.

```
# Code was taken from GeekForGeeks site:
# https://www.geeksforgeeks.org/python-program-for-merge-sort/

# Merges two subarrays of arr[].
# First subarray is arr[l..m]
# Second subarray is arr[m+1..r]
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m

    # create temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

    # Copy data to temp arrays L[] and R[]
    for i in range(0, n1):
        L[i] = arr[l + i]

    for j in range(0, n2):
        R[j] = arr[m + 1 + j]
```

```
    # Merge the temp arrays back into arr[l..r]
    i = 0     # Initial index of first subarray
    j = 0     # Initial index of second subarray
    k = l     # Initial index of merged subarray

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    # Copy the remaining elements of L[], if there
    # are any
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    # Copy the remaining elements of R[], if there
    # are any
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1


# l is for left index and r is right index of the
# sub-array of arr to be sorted
def merge_sort(arr, l, r):
    if l < r:

        # Same as (l+r)//2, but avoids overflow for
        # large l and h
        m = (l + (r - 1)) // 2

        # Sort first and second halves
        merge_sort(arr, l, m)
        merge_sort(arr, m + 1, r)
        merge(arr, l, m, r)

    return arr
```

Рис. 2 (Код тестируемого алгоритма)

Результаты экспериментов

Кол-во входных элементов	Количество операций сравнения	Затраченное время (с)	Отношение количества операций сравнения [T(n)/T(n/10)]	Отношение $g(n) = nlog_2 n$ [g(n)/g(n/10)]
10^2	542	0.0007	-	-
10^3	8708	0.0112	16,066	15
10^4	120444	0.1574	13,831	13.333
10^5	1536355	2.0298	12,756	12.5
10^6	18674212	25.1163	12,155	12
10^7	220101151	283.9356	11,786	11.666

Табл. 1 (Сводная таблица трудоемкости для различных размеров массивов)

Исходя из данных эксперимента (Табл. 1) можно сказать, что результаты эксперимента соответствуют полученным ранее теоретическим

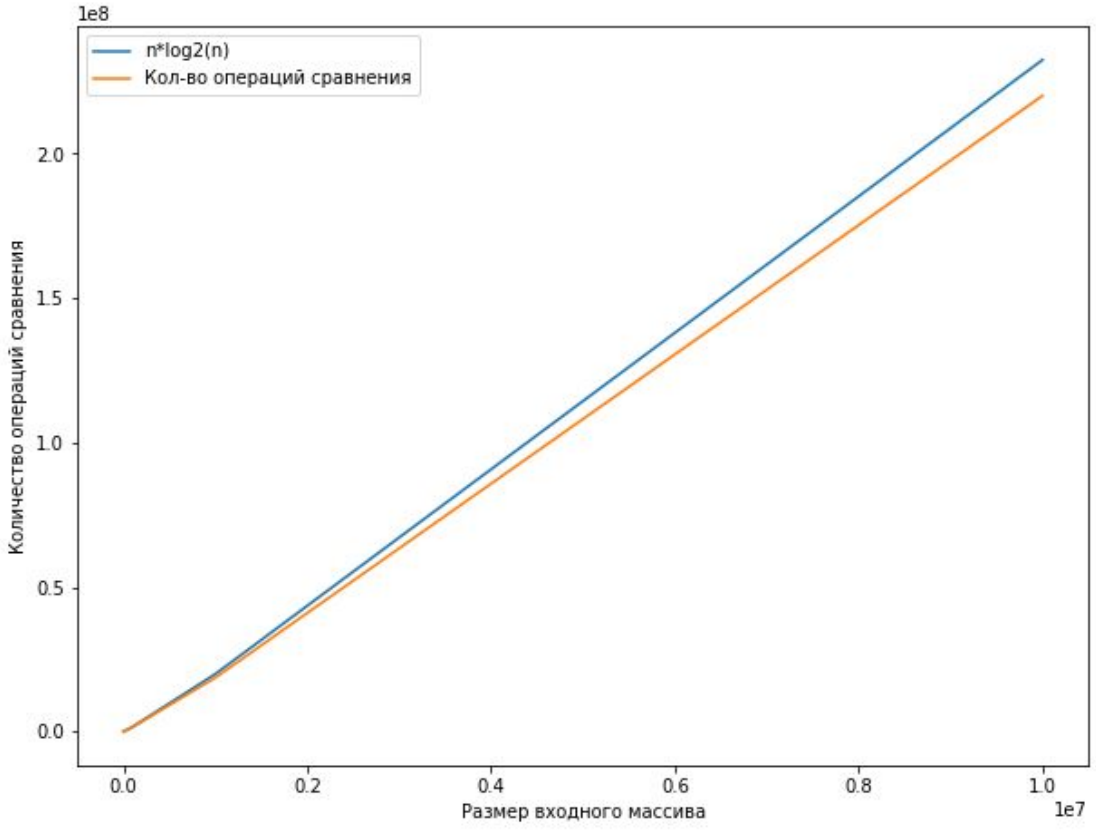

Рис.3 (Зависимость количества операций сравнения от размера входного массива)

оценкам. Также можно отметить, что количество операций сравнений в среднем на ста примерах случайных входных данных меньше чем $nlog_2n$. Это хорошо заметно на графике ниже (Рис. 3). Также хочу заметить, что временная оценка показала похожее поведение алгоритма (рис. 4). Из этого следует, что достаточно было воспользоваться измерением затраченного вычислительного времени для данного эксперимента.

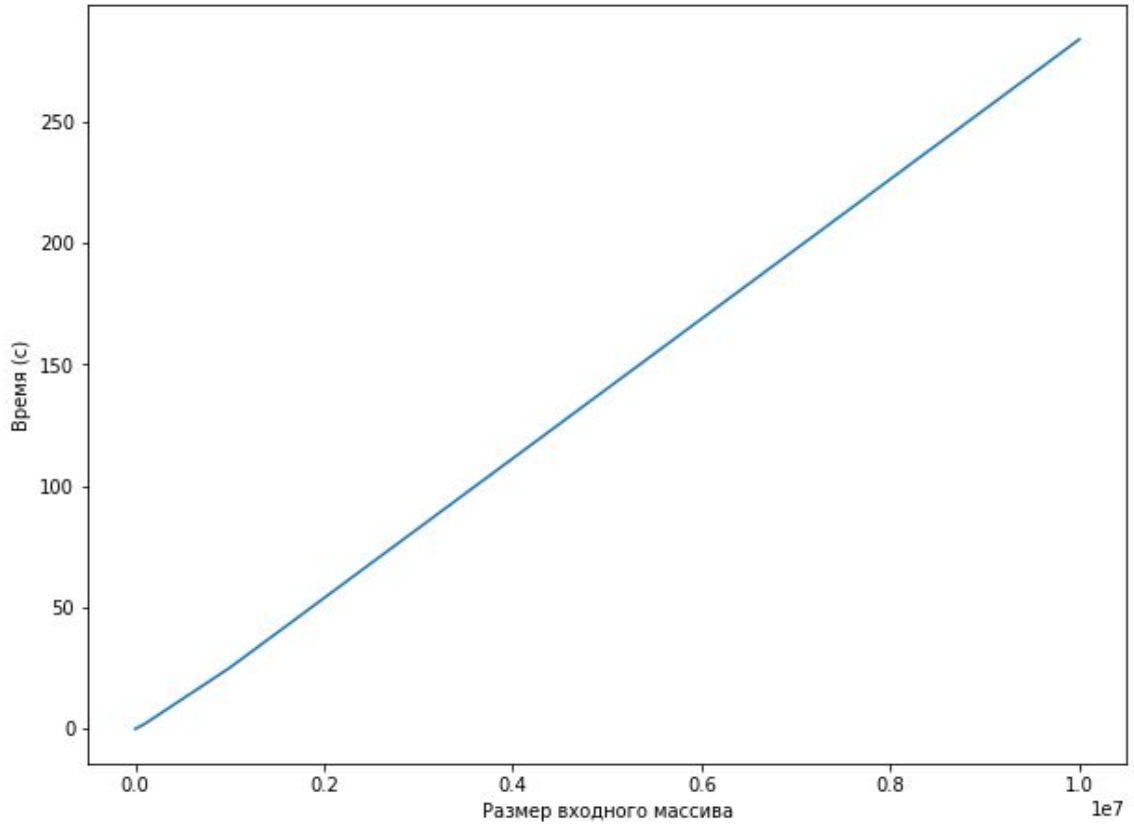

Рис. 4 (Зависимость затраченного времени от размера входного массива)

Данная тема требует дальнейшего анализа. Так, например, merge sort чаще используют для сортировки уже частично упорядоченных массивов, на которых он показывает себя намного лучше. Было бы неплохо проверить скорость работы алгоритма в зависимости от упорядоченности входных данных. Также в будущем можно проверить поведение сортировки слиянием для разных конкретных случаев использования (более конкретными входными данными и размерами массивов).

Характеристики вычислительной системы

Операционная система: Windows 10 Pro 1909
Процессор: Intel Core i5-8300H @ 2.3GHz
Объём ОЗУ: 8 Гб
Среда разработки: Visual Studio Code
Язык разработки: Python 3.9.0 64-bit
Сторонние библиотеки: NumPy, PyTest

Источники

1. Knuth, Donald (1998). The Art of Computer Programming: Volume 3 Sorting and Searching. Boston: Addison-Wesley. pp. 490-500 ISBN 978-0-201-89685-5.
2. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 3-е изд. — М.: Вильямс, 2005. — С. 52—61, 52—61, 90—140. — 1296 с. — ISBN 5-8459-0857-4.
3. Axtmann, Michael; Bingmann, Timo; Sanders, Peter; Schulz, Christian (2015). "Practical Massively Parallel Sorting". Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures: 13–23. doi:10.1145/2755573.2755595. ISBN 9781450335881.
4. Auger, Nicolas; Nicaud, Cyril; Pivoteau, Carine (2015). "Merge Strategies: from Merge Sort to TimSort". hal-01212839.

Дополнительные материалы

- Ананий В. Левитин. Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Algorithms. — М.: «Вильямс», 2006. — С. 127—133. — ISBN 0-201-74395-7.
- "Merge sort". en.wikipedia.org. Дата обращения: 23 ноября 2020 года.
- "Python Program for Merge Sort". geeksforgeeks.org. Дата обращения: 25 ноября 2020 года.
- "MergeSortAnalysis". https://github.com/PnthrLeo/MergeSortAnalysis. Дата обращения: 27 ноября 2020 года.