Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторная работа №4
Вариант №1

Выполнил:
Бацанова Е. А.
Проверил
Мусаев А.А.

Санкт-Петербург,
2024

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Методы сортировки являются неотъемлемой частью программирования и используются в различных областях для упорядочивания данных. Поэтому важно понимать особенности и эффективность различных методов сортировки для выбора наиболее подходящего в конкретной ситуации.

Целью данной работы является изучение различных методов сортировки и их сравнительный анализ. Задачи работы включают в себя написание программ с функциями для:

1) быстрой сортировки;
2) сортировки расческой;
3) блочной сортировки;
4) пирамидальной сортировки

Также будет произведена оценка времени выполнения быстрой сортировки и сортировки расческой с помощью модуля timeit.

Задание 1

Задание: Написать программу с функциями для быстрой сортировки и сортировки расческой. Использовать данные функции как модуль в другой программе. Пользователь выбирает один из двух методов сортировки. Оценить время выполнения программы с помощью модуля timeit.

Решение:

1) Быстрая сортировка

Алгоритм быстрой сортировки заключается в следующем:

➔ Сначала мы выбираем элемент из списка, который называется опорным (в данном случае это переменная element). Для улучшения эффективности и во избежание снижения производительности на отсортированных массивах выберем опорным средний элемент массива;

➔ Разделяем список на две части: левую часть (список *less*), содержащую элементы меньше опорного, и правую часть (список *greater*), содержащую элементы, больше опорного;

➔ Рекурсивно применяем алгоритм к левой и правой частям списка, пока не получим список, содержащий единственный элемент;

➔ В конце возвращаем результат слияния отсортированных списков *less* и *greater* с добавленным между ними опорным элементом.

```
def quick_sort(arr):
    n = len(arr)
    if len(arr) <= 1:
        return arr
    else:
        element = arr[n // 2]
        less = [x for x in arr[:n // 2] + arr[n // 2 + 1:] if x <= element]
        greater = [x for x in arr[:n // 2] + arr[n // 2 + 1:] if x > element]
        return quick_sort(less) + [element] + quick_sort(greater)
```

Рисунок 1 – Реализация алгоритма быстрой сортировки

2) Сортировка расчёской

Алгоритм сортировки расческой заключается в следующем:

➔ На первом этапе определяется расстояние между сравниваемыми элементами массива. В данной программе это делается путем уменьшения расстояния *n* с каждой итерацией в 1,25 раз;

➔ На втором этапе два элемента массива, отстоящих друг от друга на расстояние *n*, сравниваются между собой. Если элемент с большим индексом оказывается меньше элемента с меньшим индексом, то они меняются местами;

➔ Этот процесс продолжается до тех пор, пока расстояние *n* не станет меньше 1 – тогда массив будет отсортирован.

```python
def comb_sort(arr):
    n = len(arr)
    while True:
        n = int(n / 1.25)
        if n < 1:
            break
        i = 0
        while i + n < len(arr):
            print(arr[i], arr[i + n])
            if arr[i] > arr[i + n]:
                arr[i], arr[i + n] = arr[i + n], arr[i]
            i += 1
    return arr
```

Рисунок 2 – Реализация алгоритма сортировки расческой

Теперь используем данные функции как модуль в другой программе, в которой пользователь выбирает один из двух методов сортировки. Создадим файл module.py, в котором будут содержаться написанные ранее функции. Импортируем модуль и реализуем процесс выбора.

```python
import module

arr = input("Введите массив: ").split(' ')
case = int(input("Выберите сортировку (1 - быстрая, 2 - расческой): "))

print(f"Быстрая сортировка массива {arr}: {module.quick_sort(arr)}" if case == 1 else
      f"Сортировка расческой массива {arr}: {module.comb_sort(arr)}")
```

Рисунок 3 – Реализация программы, в которой пользователь выбирает один из двух методов сортировки (быструю или расческой)

```
Введите массив: 4 5 34 45 222 3 4 -99 0
Выберите сортировку (1 - быстрая, 2 - расческой): 1
Быстрая сортировка массива ['4', '5', '34', '45', '222', '3', '4', '-99', '0']: ['-99', '0', '222', '3', '34', '4', '4', '45', '5']
```

Рисунок 4 – Вывод программы, в которой пользователь выбирает один из двух методов сортировки (быструю или расческой)

Оценим время выполнения программы с помощью модуля timeit.

Таблица 1. Время выполнения быстрой сортировки и сортировки расческой в зависимости от размера массива из случайных чисел *n*

n	Быстрая сортировка	Сортировка расчёской
50	0.04701020900392905	0.030791583005338907
100	0.10871362499892712	0.07849958399310708
1000	1.5686119160382077	1.9210345409810543
2000	3.7547024579835124	4.746734582993668

Таблица 2. Время выполнения быстрой сортировки и сортировки расческой в зависимости от размера массива из частично отсортированных чисел *n*

n	Быстрая сортировка	Сортировка расчёской
50	0.04573695897124708	0.03471645899116993
100	0.10310916701564565	0.07976416702149436
1000	1.2048440420185216	1.8657764170202427
2000	3.5837977079791017	4.934624709014315

Из таблицы видно, что сортировка расчёской работает быстрее при малых объемах массивов, в то время как быстрая сортировка – при больших. Также быстрая сортировка, в отличие от сортировки расческой, улучшает свою скорость, если массив частично отсортирован.

Задание 2

Задание: Изучить блочную и пирамидальную сортировку. Написать соответствующие программы.

Решение:

1) Блочная сортировка

Алгоритм блочной сортировки заключается в следующем:

➔ Исходный массив *arr* делим на блоки одинаковой длины *n* и добавляем каждый из них в массив *blocks*;

➔ Когда все числа размещены по блокам, каждый из блоков сортируется отдельно, после чего все отсортированные блоки объединяются в один список *result*;

➔ В результате получается отсортированный список чисел *result*.

```
def bucket_sort(arr):
    n = (max(arr) - min(arr)) // 10
    blocks, result = [], []
    for i in range(0, len(arr), n):
        blocks.append(arr[i:i + n])
    for block in blocks:
        block.sort()
        result += block
    return result
```

Рисунок 5 – Реализация алгоритма блочной сортировки

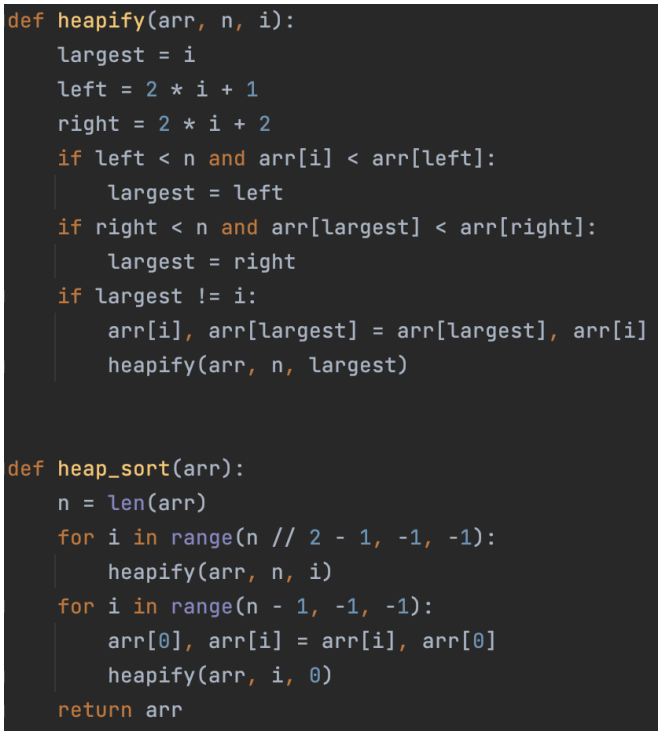
2) Пирамидальная сортировка

Алгоритм пирамидальной сортировки заключается в следующем:

➔ Саму сортировку реализует функция *heap_sort*, в то время как функция *heapify* является вспомогательной (выстраивает элементы массива в виде сортирующего дерева);

➔ Функция *heapify* среди трех узлов (родительского и двух дочерних) выбирает наибольший элемент, чтобы переместить его в корень дерева (при необходимости меняя значение *arr[i]* с *arr[largest]*) и вызывает себя рекурсивно для дочернего элемента;

➔ Функция *heap_sort* использует функцию *heapify* для создания сортирующего дерева, а затем поочередно извлекает элементы дерева, обменивая их с первым элементом и выполняя *heapify* для уменьшенного подмассива.



```python
def heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2
    if left < n and arr[i] < arr[left]:
        largest = left
    if right < n and arr[largest] < arr[right]:
        largest = right
    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)


def heap_sort(arr):
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)
    for i in range(n - 1, -1, -1):
        arr[0], arr[i] = arr[i], arr[0]
        heapify(arr, i, 0)
    return arr
```

Рисунок 6 – Реализация алгоритма пирамидальной сортировки

Задание 3

Задание: Оцените достоинства, недостатки и сложность изученных методов сортировок.

Решение:

Таблица 3. Сравнение изученных методов сортировок

Параметры сравнения	Достоинства	Недостатки	Сложность
Быстрая сортировка	- Один из самых эффективных методов сортировки; - Не требует много памяти; - Показывает хорошие результаты на больших массивах данных.	- Неустойчива к вариациям размера входных данных. - Может быть неэффективна на небольших размерах массива.	в лучшем случае: $O(n\ logn)$ в худшем случае: $O(n^2)$
Сортировка расчёской	- При малых массивах из случайных чисел показывает хорошие результаты; - Не работает оптимально при частично отсортированном массиве	- Не работает оптимально при частично отсортированном массиве и большом объёме исходного массива; - Неустойчива.	в лучшем случае: $O(n\ logn)$ в худшем случае: $O(n^2)$
Блочная сортировка	- Эффективный метод сортировки для больших объемов данных; - Устойчива к разнообразным видам данных.	- Требует больших объемов памяти для эффективной работы; - Скорость может упасть при неудачных входных данных (большом количестве элементов, а следовательно и размере блоков).	в лучшем случае: $O(n\ +\ k)$, где k – количество блоков в худшем случае: $O(n^2)$

Пирамидальная сортировка	- Устойчива к разнообразным видам данных; - проста в реализации; - Может быть крайне эффективна при удачных входных данных.	- Необходима дополнительная память для хранения пирамиды; - Нестабильна; - Низкая эффективность для частично упорядоченных данных.	в лучшем случае: $O(n)$ в худшем случае: $O(n\ logn)$

ЗАКЛЮЧЕНИЕ

В ходе работы были изучены и реализованы различные методы сортировки, такие как быстрая сортировка, сортировка расческой, блочная и пирамидальная сортировки. Также были написаны программы, реализующие данные виды сортировок.

Было проведено сравнение быстрой сортировки и сортировки расчёской в контексте затраченного на выполнение программы времени. Быстрая сортировка показала лучшие результаты на больших объемах входного массива, сортировка расчёской – на малых. Также было выявлено, что быстрая сортировка работает значительно быстрее для частично отсортированных массивов, в то время как сортировка расчёской – нет.

В конце работы было проведено сравнение всех четырех алгоритмов сортировки, выявлены их достоинства, недостатки и оценена сложность (в лучшем и худшем случае). Можно сказать, что каждый из методов сортировки имеет сильные и слабые стороны, поэтому необходимо выбирать подходящий метод в зависимости от конкретной задачи и требований к производительности.

СПИСОК ЛИТЕРАТУРЫ

1) Викиконспекты. Быстрая сортировка. [Электронный ресурс] – (Дата последнего обращения 16.05.2024);
2) Habr. Описание алгоритмов сортировки и сравнение их производительности. [Электронный ресурс] – (Дата последнего обращения 16.05.2024);
3) Wikipedia. Сортировка расчёской. [Электронный ресурс] – (Дата последнего обращения 16.05.2024);
4) Wikipedia. Блочная сортировка. [Электронный ресурс] – (Дата последнего обращения 16.05.2024);
5) Habr. Пирамидальная сортировка (HeapSort). [Электронный ресурс] – (Дата последнего обращения 16.05.2024);
6) Wikipedia. Пирамидальная сортировка. [Электронный ресурс] – (Дата последнего обращения 16.05.2024).

ПРИЛОЖЕНИЕ

Для удобства все файлы выгружены на GitHub: https://github.com/kathykkKk/Algorithms-and-Data-Structures-ICT.git