Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет Инфокоммуникационных технологий (ИКТ)

Образовательная программа Мобильные и сетевые технологии

О Т Ч Е Т

по Лабораторной работе 3

Дисциплина: Алгоритмы и структуры данных

Тема: Алгоритмы и сортировки.

Специальность: 09.03.03 Прикладная информатика

Проверил:
Мусаев А. А. _____
Дата: «__»_____2023 г.
Оценка: _____

Выполнил:
Балцат К. И.,
студент группы К33401

Санкт-Петербург
2023

ВЫПОЛНЕНИЕ

1 Задача 1

Написать программу с функциями для быстрой сортировки и сортировки расческой. Оценить время выполнения функций с помощью модуля timeit.

Решение:

Для начала напишем функции быстрой сортировки и сортировки расческой:

```python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[0]
        less = [i for i in arr[1:] if i <= pivot]
        greater = [i for i in arr[1:] if i > pivot]
        return quick_sort(less) + [pivot] + quick_sort(greater)


def comb_sort(arr):
    gap = len(arr)
    shrink = 1.3
    swapped = True
    i = 0

    while gap > 1 or swapped:
        gap = int(gap / shrink)
        if gap < 1:
            gap = 1

        i = 0
        swapped = False
        while i + gap < len(arr):
            if arr[i] > arr[i + gap]:
                arr[i], arr[i + gap] = arr[i + gap], arr[i]
                swapped = True
            i += 1

    return arr
```

Теперь оценим время выполнения функций. Для этого мы можем использовать модуль timeit. Этот модуль позволяет измерить время выполнения фрагмента кода, повторив его несколько раз. Вот пример

использования:

```
33  if __name__ == '__main__':
34      import timeit
35      import sys
36      sys.setrecursionlimit(10000)  # устанавливаем максимальную глубину рекур
37
38      arr = [i for i in range(1000)]
39
40      # Измеряем время выполнения быстрой сортировки
41      quick_sort_time = timeit.timeit(lambda: quick_sort(arr), number=1000)
42      print(f"Время выполнения быстрой сортировки: {quick_sort_time:.6f} сек."
43
44      # Измеряем время выполнения сортировки расческой
45      comb_sort_time = timeit.timeit(lambda: comb_sort(arr), number=1000)
46      print(f"Время выполнения сортировки расческой: {comb_sort_time:.6f} сек.'
```

```
konstantinbaltsat@MacBook-Pro-Konstantin-3 HW3 % python3 task1.py
Время выполнения быстрой сортировки: 37.875502 сек.
Время выполнения сортировки расческой: 4.746238 сек.
```

2 Задача 2

Изучить блочную, пирамидальную и сортировку слиянием. Написать соответствующие программы.

Решение:

Блочная сортировка (Bucket sort) - это алгоритм сортировки, который сначала распределяет элементы массива во множество блоков, называемых «ведрами» или "карманами", затем сортирует каждый карман отдельно, а затем объединяет карманы в один отсортированный массив. Временная сложность блочной сортировки зависит от выбранной структуры данных для хранения ведер и может быть O(n) в лучшем случае и O(n^2) в худшем случае. Вот пример реализации блочной сортировки на Python:

```python
# Блочная сортировка (Bucket sort)
def bucket_sort(arr, num_buckets=20):
    # Создание "карманов" (buckets)
    buckets = [[] for _ in range(num_buckets)]

    # Распределение элементов по "карманам"
    for value in arr:
        index = int(value * num_buckets)
        buckets[index].append(value)

    # Сортировка элементов в каждом "кармане"
    for i in range(num_buckets):
        buckets[i].sort()

    # Объединение "карманов"
    sorted_arr = []
    for bucket in buckets:
        sorted_arr += bucket

    return sorted_arr
```

Пирамидальная сортировка (Heap sort) - это алгоритм сортировки, который использует структуру данных под названием "куча" (heap). Куча - это двоичное дерево, у которого выполнены два свойства: свойство кучи и свойство полноты. Свойство кучи означает, что значение каждого узла больше (меньше) значений его потомков. Свойство полноты означает, что куча заполнена слева направо без пропусков. Для сортировки массива с помощью сортировки пирамидой, массив преобразуется в кучу, а затем извлекаются элементы из кучи по одному и добавляются в упорядоченный массив.

Вот пример реализации пирамидальной сортировки на Python:

```python
# Пирамидальная сортировка (Heap sort)
def heap_sort(arr):
    def heapify(arr, n, i):
        largest = i
        l = 2 * i + 1
        r = 2 * i + 2

        if l < n and arr[i] < arr[l]:
            largest = l

        if r < n and arr[largest] < arr[r]:
            largest = r

        if largest != i:
            arr[i], arr[largest] = arr[largest], arr[i]
            heapify(arr, n, largest)

    n = len(arr)
    for i in range(n, -1, -1):
        heapify(arr, n, i)

    for i in range(n-1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)
    return arr
```

Сортировка слиянием (Merge Sort) - это алгоритм сортировки, который разделяет массив на две половины, сортирует каждую половину рекурсивно, а затем объединяет две отсортированные половины в один отсортированный массив. Основным преимуществом сортировки слиянием является то, что он гарантирует стабильность (сохранение порядка элементов с одинаковыми ключами).

Код для сортировки слиянием:

```python
# Сортировка слиянием (Merge Sort)
def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left = arr[:mid]
    right = arr[mid:]

    left = merge_sort(left)
    right = merge_sort(right)

    def merge(left, right):
        result = []
        i = 0
        j = 0

        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                result.append(left[i])
                i += 1
            else:
                result.append(right[j])
                j += 1

        result += left[i:]
        result += right[j:]

        return result

    return merge(left, right)

```

3 Задача 3

Оцените достоинства, недостатки и сложность изученных методов сортировок.

Решение:

Быстрая сортировка (QuickSort):

Достоинства: обладает быстрой асимптотической сложностью, проста в реализации, эффективна на больших наборах данных.

Недостатки: может работать медленно на некоторых входных данных, таких как уже отсортированные массивы или массивы с повторяющимися значениями. Также может страдать от переполнения стека вызовов из-за рекурсивной реализации.

Сложность: O(nlogn) в среднем и лучшем случаях, O(n^2) в худшем случае.

Сортировка расческой (CombSort):

Достоинства: проста в реализации, эффективна на больших наборах данных, отличается быстрой работой на случайно распределенных входных данных.

Недостатки: может работать медленно на некоторых входных данных, таких как уже отсортированные массивы. Не так распространена и известна, как другие методы.

Сложность: O(n^2) в худшем случае, но на практике она имеет лучшую производительность.

Блочная сортировка (BucketSort):

Достоинства: хорошо подходит для сортировки больших объемов данных, особенно когда элементы находятся в узком диапазоне. Также может быть эффективна при работе с дисками или другими внешними устройствами.

Недостатки: требует больше памяти, чем другие алгоритмы, из-за необходимости создания дополнительных списков (карманов). Также может быть неэффективна при работе с данными, которые не могут быть разбиты на категории.

Сложность: в среднем случае O(n+k), где n - количество элементов, k - количество "карманов".

Пирамидальная сортировка (HeapSort):

Достоинства: имеет гарантированную асимптотическую сложность, проста в реализации, эффективна на больших наборах данных.

Недостатки: может работать медленно на некоторых входных данных, таких как уже отсортированные массивы. Из-за необходимости использования дополнительной памяти для хранения кучи может быть неэффективна на больших объемах данных.

Сложность: O(nlogn) в любом случае.

Сортировка слиянием (MergeSort):

Достоинства: гарантированная сложность O(nlog(n)) в худшем, лучшем и среднем случаях. Хорошо работает с большими объемами данных, не зависит от расположения элементов в массиве. Не имеет практических ограничений по размеру входных данных.

Недостатки: требует дополнительной памяти для хранения временных массивов при слиянии. Некоторые реализации могут быть нестабильными.

Сложность: O(nlog(n)) в худшем, лучшем и среднем случаях, где n - количество элементов в массиве.

ВЫВОД

Я изучил алгоритмы сортировок и понял достоинства и недостатки каждого. Теперь я способен эффективно применять каждый изученный алгоритм в зависимости от решаемой задачи.