МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский технологический университет «МИСИС»
СТАРООСКОЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ ИМ. А.А. УГАРОВА
(филиал) федерального государственного автономного образовательного учреждения высшего образования
«Национальный исследовательский технологический университет «МИСИС»
(СТИ НИТУ «МИСИС»)

ФАКУЛЬТЕТ АВТОМАТИЗАЦИИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ КАФЕДРА АВТОМАТИЗИРОВАННЫХ И ИНФОРМАЦИОННЫХ СИСТЕМ УПРАВЛЕНИЯ ИМ. Ю.И. ЕРЕМЕНКО

Домашняя работа №1

по дисциплине: «Теория алгоритмов и структур данных»

Выполнил студент группы: АТ/МС-23Д, Небольсин Василий Дмитриевич
группа, ФИО полностью подпись

Проверил: ассистент кафедры АИСУ, Жуков Петр Игоревич
Должность, звание, ФИО полностью подпись

Старый Оскол, 2023

Задание

Реализовать алгоритм пузырьковой сортировки случайных значений длиной 10 и 100 тысяч. Оптимизировать метод до логарифмической вычислительной сложности.

Решение

Пузырьковая сортировка, иногда называемая сортировкой по потоку, представляет собой простой алгоритм сортировки, который многократно перебирает входной список элемент за элементом, сравнивая текущий элемент с следующим за ним, меняя их значения при необходимости. Эти проходы по списку повторяются до тех пор, пока во время прохода не перестанут выполняться замены, что означает, что список стал полностью отсортированным. Алгоритм, представляющий собой сравнительную сортировку.

Общая идея алгоритма состоит в следующем:

Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность (см. ниже).

Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие»[2].

Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения (листинг).

Listing 1: Алгоритм пузырьковой сортировки

```
def bubble_sort(arr):
  for i in range(len(arr)):
    for j in range(i+1, len(arr)):
      if arr[j] > arr[i]: # Comparing two list elements.
        arr[j], arr[i] = arr[i], arr[j] # Swap elements.
```

Эта функция принимает на вход массив целых чисел и сортирует его с помощью пузырьковой сортировки.

Функция сначала инициализирует переменную «i», чтобы отслеживать текущий индекс в массиве. Затем он проходит по всему массиву, используя два вложенных цикла for. На каждой итерации он сравнивает текущий элемент со следующим элементом и меняет их местами, если они находятся

в неправильном порядке. Функция также обновляет индексы элементов по ходу работы.

После завершения цикла функция возвращает отсортированный массив.

Оптимизация пузырьковой сортировки можно провести с помощью нескольких подходов. Приведу примеры с кодом на Python.

Установка флага, чтобы отслеживать, были ли выполнены какие-либо обмены во внутреннем цикле. Если обмены не были выполнены, значит, массив уже отсортирован, и сортировка может быть завершена.

Listing 2: Вид оптимизации алгоритма пузырьковой сортировки

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
```

Запоминание позиции последнего выполненного обмена внутреннего цикла. Все элементы после этой позиции уже отсортированы, поэтому в следующей итерации внешнего цикла можно сократить длину внутреннего цикла.

Listing 3: Вид оптимизации алгоритма пузырьковой сортировки

```
def bubble_sort(arr):
    n = len(arr)
    k = n - 1
    for i in range(n):
        swapped = False
        for j in range(0, k):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
                k = j
        if not swapped:
            break
    return arr
```

Оптимизация за счет использования переменной, которая будет

отслеживать последнюю позицию, в которой был выполнен обмен. В следующей итерации внешнего цикла можно ограничить внутренний цикл до этой позиции.

Listing 4: Вид оптимизации алгоритма пузырьковой сортировки

```
def bubble_sort(arr):
    n = len(arr)
    last_swapped = n - 1
    for i in range(n):
        new_last_swapped = -1
        for j in range(0, last_swapped):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                new_last_swapped = j
        if new_last_swapped == -1:
            break
        last_swapped = new_last_swapped
    return arr
```

Все эти оптимизации позволяют сократить количество операций сравнения и обмена, что улучшает производительность пузырьковой сортировки.

Пузырьковая сортировка с рекурсией - это вариант пузырьковой сортировки, который использует рекурсию для повторного прохождения по списку до тех пор, пока он не будет полностью отсортирован.

Рекурсивная пузырьковая сортировка начинается с обычного прохода по списку с помощью цикла for или while. Если в текущем проходе были произведены перестановки, то рекурсивно вызывается та же функция для неотсортированной части списка (листинг).

Listing 5: Алгоритм рекурсивной пузырьковой сортировки

```
def bubble_sort_recursive(arr, n):
    if n == 1:
        return
    for i in range(n-1):
        if arr[i] > arr[i+1]:
            arr[i], arr[i+1] = arr[i+1], arr[i]
    bubble_sort_recursive(arr, n-1)
```

Результаты

Время работы пузырьковой сортировки со списками в 10 и 100 тысяч значений составил порядка 22 и 2328 секунд соответственно, что

доказывает её неэффективность. В тоже время алгоритм с рекурсией показал наилучший результат, справившись с темиже наборами за 13 и 1783 секунд.