САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

# Отчет по лабораторной работе № 3

## по курсу «Алгоритмы и структуры данных»

## Тема: Быстрая сортировка, сортировки за линейное время

Вариант 21

Выполнила:
Савченко А.С.

Санкт-Петербург
2024 г.

## Содержание отчета

# Задачи по варианту

## Задача №1. Улучшение Quick sort

Текст задачи:

### задача. Улучшение Quick sort

1. Используя *псевдокод* процедуры `Randomized - QuickSort`, а так же `Partition` из презентации к Лекции 3 (страницы 8 и 12), напишите программу быстрой сортировки на Python и проверьте ее, создав несколько рандомных массивов, подходящих под параметры:

   - **Формат входного файла (input.txt).** В первой строке входного файла содержится число $n$ ($1 \leq n \leq 10^4$) — число элементов в массиве. Во второй строке находятся $n$ различных целых чисел, ***по модулю*** не превосходящих $10^9$.
   - **Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
   - Ограничение по времени. 2 сек.
   - Ограничение по памяти. 256 мб.
   - Для проверки можно выбрать наихудший случай, когда сортируется массив рамера $10^3, 10^4, 10^5$ чисел порядка $10^9$, отсортированных в обратном порядке; наилучший, когда массив уже отсортирван, и средний - случайный. Сравните на данных сетах `Randomized-QuickSort` и простой `QuickSort`. (А также есть `Median-QuickSort`, см. задание 10.2; и `Tail-Recursive-QuickSort`, см. Кормен. 2013, стр. 217)

2. **Основное задание.** Цель задачи - переделать данную реализацию рандомизированного алгоритма быстрой сортировки, чтобы она работала быстро даже с последовательностями, содержащими много одинаковых элементов. Чтобы заставить алгоритм быстрой сортировки эффективно обрабатывать последовательности с несколькими уникальными элементами, нужно заменить двухстороннее разделение на трехстороннее (смотри в Лекции 3 слайд 17). То есть ваша новая процедура разделения должна разбить массив на три части:

   - $A[k] < x$ для всех $\ell + 1 \leq k \leq m_1 - 1$
   - $A[k] = x$ для всех $m_1 \leq k \leq m_2$
   - $A[k] > x$ для всех $m_2 + 1 \leq k \leq r$

   - Формат входного и выходного файла аналогичен п.1.
   - Аналогично п.1 этого задания сравните `Randomized-QuickSort` +с `Partition` и ее с `Partition3` на сетах случайных данных, в которых содержатся всего несколько уникальных элементов при $n = 10^3, 10^4, 10^5$. Что быстрее, `Randomized-QuickSort` +с `Partition3` или `Merge-Sort`?
   - Пример:

| input.txt | output.txt |
|---|---|
| 5<br>2 3 9 2 2 | 2 2 2 3 9 |

Листинг кода (именно листинг, а не скрины):

1 задание

```
from random import randint


def partition(a, l, r):
    x = a[l]
    j = l
    for i in range(l + 1, r + 1):
        if a[i] <= x:
            j += 1
            a[i], a[j] = a[j], a[i]
    a[l], a[j] = a[j], a[l]
    return j


def random_qs(a, l, r):
    if l < r:
        k = randint(l, r)
        a[l], a[k] = a[k], a[l]
        m = partition(a, l, r)
        random_qs(a, l, m - 1)
        random_qs(a, m + 1, r)


with open('input.txt', 'r') as file:
    n = int(file.readline().strip())
    a = list(map(int, file.readline().strip().split()))

random_qs(a, 0, n - 1)

with open('output.txt', 'w') as file:
    file.write(' '.join(map(str, a)))
```

2 задание

```
from random import randint
```

```
# разбиваем массив на три части
def partition(a, l, r):
    x = a[l]
    m1 = l
    m2 = r
    i = l
    while i <= m2:
        if a[i] < x:
            a[m1], a[i] = a[i], a[m1]
            m1 += 1
            i += 1
        elif a[i] > x:
            a[m2], a[i] = a[i], a[m2]
            m2 -= 1
        else:
            i += 1
    return m1, m2


def random_qs(a, l, r):
    if l < r:
        k = randint(l, r)
        a[l], a[k] = a[k], a[l]
        m1, m2 = partition(a, l, r)
        random_qs(a, l, m1 - 1)
        random_qs(a, m2 + 1, r)


with open('input.txt', 'r') as file:
    n = int(file.readline().strip())
    a = list(map(int, file.readline().strip().split()))

random_qs(a, 0, n - 1)
```

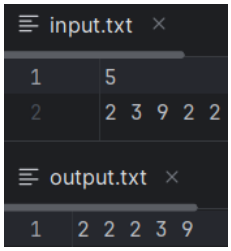
```
with open('output.txt', 'w') as file:
    file.write(' '.join(map(str, a)))
```

Текстовое объяснение решения:

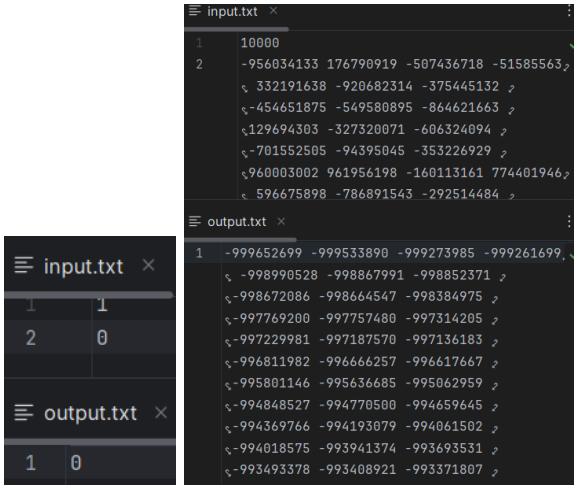
В partition теперь будем разбивать массив на три части - элементы меньшие опорного x, равные и большие. Инициализирум казатели, и начинаем сравнение с опорным элементом: распределяем элементы меньше опорного в влево, большие вправо.

В random_qs рекурсивно сортируем массив. Случайно выбираем опорный элемент, меняем его местами с первым элементом. Используем partition, а затем рекурсивно сортируем левую и правую части массива.

Результат работы кода на примерах из текста задачи:(скрины input output файлов):




Результат работы кода на минимальных и максимальных значениях для 2 задания:(скрины input output файлов):




| | Время выполнения | Затраты памяти |
|---|---|---|
| Нижняя граница диапазона значений | 0.002519 с | 15.046875 Мб |

| | | |
|---|---|---|
| входных данных из текста задачи | | |
| Пример из задачи | 0.0017907 с | 14.9375 Мб |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.026281 с | 16.62109375 Мб |

Вывод по задаче:

В задаче была модифицирована двухсторонняя быстрая сортировка до трехсторонней, которая будет более эффективной для массивов с большим числом уникальных элементов.

Сравнение:

*Randomized-QuickSort + Partition:*

+Хорошо работает на случайных наборах данных

-в худшем случае $O(n^2)$ при наличии большого количества одинаковых элементов

*Randomized-QuickSort + Partition3:*

+ благодаря трехстороннему разбиению устойчив к множеству одинаковых элементов

*Merge-Sort:*

+стабильно, время выполнения O(n log n), независимо от распределения элементов.

В случае наличия большого кол-ва одинак. Эл. Трехсторонняя сортировка работает быстрее Merge-Sort.

**Задача №3. Сортировка пугалом**

Текст задачи:

### 3 задача. Сортировка пугалом

«Сортировка пугалом» — это давно забытая народная потешка. Участнику под верхнюю одежду продевают деревянную палку, так что у него оказываются растопырены руки, как у огородного пугала. Перед ним ставятся $n$ матрёшек в ряд. Из-за палки единственное, что он может сделать — это взять в руки две матрешки на расстоянии $k$ друг от друга (то есть $i$-ую и $i + k$-ую), развернуться и поставить их обратно в ряд, таким образом поменяв их местами.

Задача участника — расположить матрёшки по неубыванию размера. Может ли он это сделать?

- **Формат входного файла (input.txt).** В первой строчке содержатся числа $n$ и $k$ ($1 \leq n, k \leq 10^5$) – число матрёшек и размах рук. Во второй строчке содержится $n$ целых чисел, которые по модулю не превосходят $10^9$ – размеры матрёшек.
- **Формат выходного файла (output.txt).** Выведите «ДА», если возможно отсортировать матрёшки по неубыванию размера, и «НЕТ» в противном случае.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

| input.txt | output.txt |
|---|---|
| 3 2<br>2 1 3 | НЕТ |
| 5 3<br>1 5 3 4 1 | ДА |

Листинг кода:

```
def is_sorted(arr):
    for i in range(1, len(arr)):
        if arr[i] < arr[i - 1]:
            return False
    return True


def pugalo(arr, k):
    n = len(arr)
    for i in range(n - k):
        if arr[i] > arr[i + k]:
            arr[i], arr[i + k] = arr[i + k], arr[i]
    return 'ДА' if is_sorted(arr) else 'НЕТ'
```

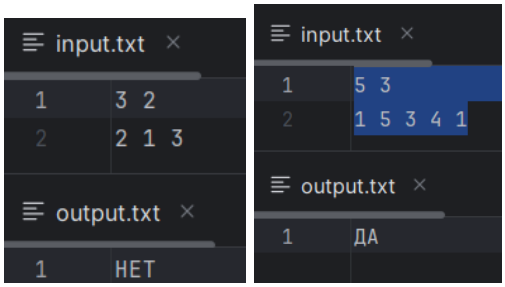
```
if __name__ == "__main__":
    with open('input.txt', 'r', encoding='utf-8') as
file:
                                n, k = map(int,
file.readline().strip().split())
                                arr = list(map(int,
file.readline().strip().split()))

   result = pugalo(arr, k)

    with open('output.txt', 'w', encoding='utf-8') as
file:
       file.write(result)
```

Текстовое объяснение решения:

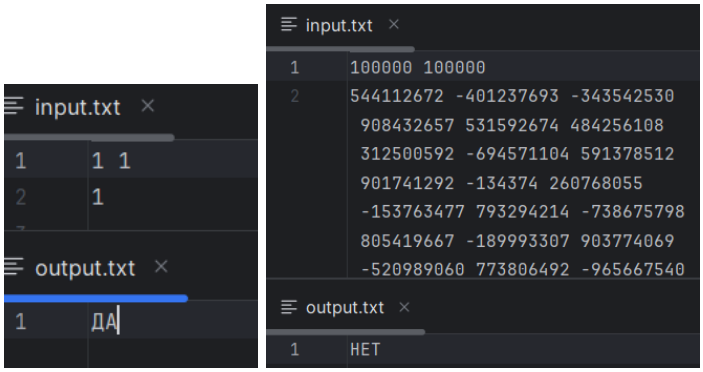
Будем проверять все возможные пары. Для каждого элемента массива arr[i] и arr[i + k] проверяется, если они находятся на расстоянии k друг от друга и не отсортированы, то они меняются местами.

После перестановок проверяем массив на отсортированность и выводим ответ

Результат работы кода на примерах из текста задачи:(скрины input output файлов):




Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):




| | Время выполнения | Затраты памяти |
|---|---|---|
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0005653 с | 14.7734375 Мб |
| Пример из задачи | 0.0006631 с | 14.92578125 Мб |
| Пример из задачи | 0.0051757 с | 14.984375 Мб |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.0210943 с | 18.0078125 Мб |

Вывод по задаче:

Алгоритм определяет возможно ли отсортировать матрешки стоящие на расстоянии k. Алгоритм успешно проходит тесты из задачи, а также на максимальных значениях входных данных.

## Задача №5. Индекс Хирша

Текст задачи:

Для заданного массива целых чисел citations, где каждое из этих чисел - число цитирований i-ой статьи ученого-исследователя, посчитайте индекс Хирша этого ученого.

По определению Индекса Хирша на Википедии: Учёный имеет индекс $h$, если $h$ из его/её $N_p$ статей цитируются как минимум $h$ раз каждая, в то время как оставшиеся $(N_p - h)$ статей цитируются не более чем $h$ раз каждая. Иными словами,

Если существует несколько возможных значений $h$, в качестве $h$-индекса принимается максимальное из них.

- **Формат ввода или входного файла (input.txt).** Одна строка citations, содержащая $n$ целых чисел, по количеству статей ученого (длина citations), разделенных пробелом или запятой.
- **Формат выхода или выходного файла (output.txt).** Одно число - индекс Хирша ($h$-индекс).
- Ограничения: $1 \leq n \leq 5000, 0 \leq citations[i] \leq 1000$.
- Пример.

| input.txt | output.txt |
|---|---|
| 3,0,6,1,5 | 3 |

Пояснение. citations = [3,0,6,1,5] означает, что ученый опубликовал 5 статей в целом, и каждая из них оказалась процитирована 3, 0, 6, 1, 5 раз соответственно. Поскольку у ученого есть 3 статьи с минимум тремя цитированиями, а у оставшихся двух - не более 3 цитирований, его индекс Хирша равен 3.

- Пример.

| input.txt | output.txt |
|---|---|
| 1,3,1 | 1 |

- Ограничений по времени (и памяти) не предусмотрено, проверьте максимальный случай при заданных ограничениях на данные, и оцените асимптотическое время.
- Подумайте, если бы массив citations был бы изначально отсортирован по возрастанию, можно было бы еще ускорить алгоритм?

Листинг кода:

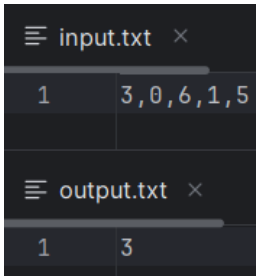
```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x > pivot]
    mid = [x for x in arr if x == pivot]
    right = [x for x in arr if x < pivot]
    return quicksort(left) + mid + quicksort(right)


def hirsch_i(citations):
    citations = quicksort(citations)
    # citations.sort()
    h = 0
```

```
    for i, c in enumerate(citations):
        if c >= i + 1:
            h = i + 1
        else:
            break
    return h


with open('input.txt', 'r') as file:
                    citations       =       list(map(int,
file.read().strip().split(',')))

h = hirsch_i(citations)

with open('output.txt', 'w') as file:
    file.write(str(h))
```

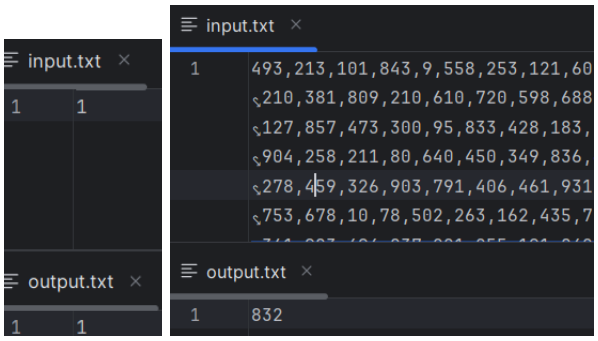
Текстовое объяснение решения:

Записываем короткий алгоритм быстрой сортировки:
Если длина arr <= 1, массив уже отсортирован и возвращается
В остальных случаях:
- выбирается опорный элемент pivot
- элементы массива делятся на три части: left -меньше опорного, mid - равные, right -больше опорного
- рекурсивно вызывается quicksort для левой и правой частей массива. Результат - соединение трех частей

Вычисляем индекс хирша: сортируем массив с помощью `quicksort`. Изначально h = 0. Проходимся по списку цитирований,где i - индекс текущего элемента, а c — кол-во цитирований. Если текущее количество цитирований c >= чем индекс текущего элемента + 1, то обновляем индекс Хирша. Если нет, прерываем цикл.
Результат работы кода на примерах из текста задачи:(скрины input output файлов):




Результат работы кода на минимальных и максимальных значениях:(скрины input output файлов):




| | Время выполнения | Затраты памяти |
|---|---|---|
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0011314 с | 14.75 Мб |
| Пример из задачи | 0.0005447 с | 14.7421875 Мб |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.0073873 с | 15.46484375 Мб |

Вывод по задаче: Я узнала о довольно интересном способе оценки продуктивности ученого на основе количества его цитирований.

После сортировки функция hirsch_i перебирает массив один раз и вычисляет индекс Хирша и имеет временную сложность O(n).
Учитывая сложность быстрой сортировки и сложность вычисления индекса Хирша, получим такую асимптотическую сложность алгоритма:
В среднем/наилучшем случае случае: O(n*log n)

В худшем случае: O(n^2)

Прим: вместо citations = quicksort(citations) можно использовать встроенную функцию сортировки в Python citations.sort() так алгоритм будет работать быстрее по времени (тк  O(n log n) в среднем и худшем случае.)

**Вывод** (по всей лабораторной)

В ходе данной лабораторной работы был улучшен алгоритм Quick sort, который благодаря трехстороннему разбиению алгоритм стал более устойчив к множеству одинаковых элементов. Также я узнала об индексе Хирша - способе оценки продуктивности ученого на основе количества его цитирований и реализовала алгоритм для его вычисления, рассчитала его асимптотику. Таким образом,  были изучены и успешно применены различные алгоритмы сортировки, все алгоритмы являются эффективными, что подкрепляется замерами памяти и времени.