

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 19

Выполнил:

Полегкий А.С,

К3142

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Улучшение Quick Sort	3
Задача №3. Сортировка пугалом	6
Задача №8. К ближайших точек к началу координат.	8
Дополнительные задачи	10
Задача №2. Анти-quick sort	10
Задача №5. Индекс Хирша	13
Вывод	15

Задачи по варианту

Задача №1. Улучшение QuickSort

1. Используя *псевдокод* процедуры Randomized-QuickSort, а также 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 ссек.
- Ограничение по памяти. 256 мб.
- Для проверки можно выбрать наихудший случай, когда сортируется массив размера 10^3 , 10^4 , 10^5 чисел порядка 10^9 , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, 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 + c Partition и cc Partition3 на сетах случайных данных, в которых содержатся всего несколько уникальных элементов при $n = 10^3, 10^4, 10^5$. Что быстрее, Randomized-QuickSort + c Partition3 или Merge-Sort?
- Пример:

input.txt	output.txt
5	2 2 2 3 9
2 3 9 2 2	

Листинг кода

```
import time, tracemalloc, random
```

```
def qSortgood(a):  
    if len(a) <= 1:  
        return a
```

```

    pivot = a[random.randint(0, len(a) - 1)]
    l = [x for x in a if x < pivot]
    r = [x for x in a if x > pivot]
    mid = [x for x in a if x == pivot]
    return qSortgood(l) + mid + qSortgood(r)

def qSortbad(left, right):
    key = a[random.randint(0, len(a) - 1)]
    i = left
    j = right
    while i <= j:
        while a[i] < key: # first while
            i += 1
        while a[j] > key : # second while
            j -= 1
        if i <= j :
            a[i], a[j] = a[j], a[i]
            i += 1
            j -= 1

    if left < j:
        qSortbad(left, j)
    if i < right:
        qSortbad(i, right)

f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = list(map(int, f1.readline().split()))
tracemalloc.start()
s = input("Which?: ")
if s == "bad":
    t_start = time.perf_counter()
    qSortbad(0, n - 1)
elif s == "good":
    t_start = time.perf_counter()
    a = qSortgood(a)
else:

```

```

    print("idk which sort")
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())
for x in a:
    f2.write(str(x) + " ")

```

Объяснение

Использовал QuickSort, который, на мой взгляд является самым легким путем для решения задачи.

Результат работы кода на примерах из текста задачи

put	output
Файл Изменить	Файл Изменить Просмотр
5 2 3 9 2 2	2 2 2 3 9

Результат работы кода на максимальных значениях

100000	1001 29087 33430 33857 35058 40
682911640 656227860 7379370	104926 139001 146476 152874 158
135066377 370946165 26732169	228241 231991 259507 294152 338
618199948 198211718 89957379	406101 410782 414139 436841 451
917967221 453855607 66984740	494734 500991 527595 548798 549
604818857 421409733 73367847	609463 610728 630513 641383 644

Проверка задачи на (асгр и тд при наличии в задаче).

```

Which?: good
Время работы: 0.6631271000078414
(804701, 2602213)

```

(пример вывода консоли для верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Верхняя граница диапазона значений входных данных из текста задачи	≈0.66сек	2602213 bytes ≈ 2541Kb

Вывод по задаче

Смог решить задачу, используя QuickSort.

Задача №3. Сортировка пугалом

5 задача. Сортировка выбором.

Рассмотрим сортировку элементов массива, которая выполняется следующим образом. Сначала определяется наименьший элемент массива, который ставится на место элемента $A[1]$. Затем производится поиск второго наименьшего элемента массива A , который ставится на место элемента $A[2]$. Этот процесс продолжается для первых $n - 1$ элементов массива A .

Напишите код этого алгоритма, также известного как сортировка выбором (selection sort). Определите время сортировки выбором в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

Формат входного и выходного файла и ограничения - как в задаче 1.

Листинг кода

```
import time, tracemalloc, random

def qSortgood(a):
    if len(a) <= 1:
        return a
    pivot = a[random.randint(0, len(a) - 1)]
    l = [x for x in a if x < pivot]
    r = [x for x in a if x > pivot]
    mid = [x for x in a if x == pivot]
    return qSortgood(l) + mid + qSortgood(r)

def scarecrow(k):
    if k == 1:
        return "YES"
    for i in range(n):
        c = 0
        j = 0
        while j < len(A[a[i][0]]):
            if abs(i - A[a[i][0]][j]) % k == 0:
                c += 1;
            A[a[i][0]].pop(j)
```

```

        j += 1;
        if (c == 0):
            return "NO"
    return "YES"

def checkSorted(a, n):
    for i in range(1, n):
        if a[i] < a[i - 1]:
            return False
    return True

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n, k = map(int, f1.readline().split())
a = list(map(int, f1.readline().split()))
A = dict()
for i in range(n):
    a[i] = [int(a[i]), i]
    A[a[i][0]] = A.get(a[i][0], [])
    A[a[i][0]].append(a[i][1]);
a = qSortgood(a);
print(scarecrow(k))
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())

```

Объяснение

Написал несколько функций, по выполнению которых, выводился результат.

Вывод по задаче

Смог написать код для решения задачи, используя «шуточный» метод сортировки.

Задача №8. К ближайших точек к началу координат

8 задача. K ближайших точек к началу координат

В этой задаче, ваша цель - найти K ближайших точек к началу координат среди данных n точек.

- Цель. Заданы n точек на поверхности, найти K точек, которые находятся ближе к началу координат $(0, 0)$, т.е. имеют наименьшее расстояние до начала координат. Напомним, что расстояние между двумя точками (x_1, y_1) и (x_2, y_2) равно $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
- Формат ввода или входного файла (input.txt). Первая строка содержит n - общее количество точек на плоскости и через пробел K - количество ближайших точек к началу координат, которые надо найти. Каждая следующая из n строк содержит 2 целых числа x_i, y_i , определяющие точку (x_i, y_i) . Ограничения: $1 \leq n \leq 10^5$; $-10^9 \leq x_i, y_i \leq 10^9$ - целые числа.
- Формат выхода или выходного файла (output.txt). Выведите K ближайших точек к началу координат в строчку в квадратных скобках через запятую. Ответ вывести в порядке возрастания расстояния до начала координат. Если оно равно, порядок произвольный.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 256 мб.
- Пример 1.

input.txt	output.txt
2 1 1 3 -2 2	[-2,2]

- Пример 2.

input.txt	output.txt
3 2 3 3 5 -1 -2 4	[3,3],[-2,4]

Листинг кода

```
import time, tracemalloc

def qSort(a):
    if len(a) <= 1:
        return a
    else:
        xx = a[0]
        l = [x for x in a[1:] if x <= xx]
        r = [x for x in a[1:] if x > xx]
        return qSort(l) + [xx] + qSort(r)

tracemalloc.start()
```



```

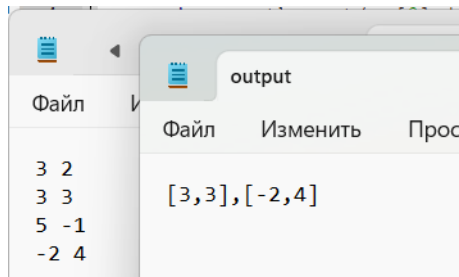
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = list(map(float, f1.readline().split()))
b = a.copy()
a = qSort(a)
f2.write(str(b.index(a[0]) + 1) + " ")
f2.write(str(b.index(a[n // 2] + 1) + " ")
f2.write(str(b.index(a[-1]) + 1) + " ")
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())
if sorted(a) == a:
    print("True")

```

Объяснение

Решил использовать дополнительные функции для решения задачи.

Результат работы кода на примерах из текста задачи



Проверка задачи на (асгр и тд при наличии в задаче).

```

Время работы: 0.00016149997827596962
(1487, 9536)

```

(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈0.00016сек	9536 bytes ≈ 9.31Kb

Вывод по задаче

Понял, как решать задачу на расположение точек, используя различные функции.

Дополнительные задачи

Задача №2. Анти-quick sort

2 задача. Анти-quick sort

Для сортировки последовательности чисел широко используется быстрая сортировка - QuickSort. Далее приведена программа на языке ~~Pascal~~ Python, которая сортирует массив `a`, используя этот алгоритм.

```
def qsort (left, right):
    key = a [(left + right) // 2]
    i = left
    j = right
    while i <= j:
        while a[i] < key: # first while
            i += 1
        while a[j] > key : # second while
            j -= 1
        if i <= j :
            a[i], a[j] = a[j], a[i]
            i += 1
            j -= 1
    if left < j:
        qsort(left, j)
    if i < right:
        qsort(i, right)

qsort(0, n - 1)
```

Хотя QuickSort является очень быстрой сортировкой в среднем, существуют тесты, на которых она работает очень долго. Оценивать время работы алгоритма будем числом сравнений с элементами массива (то есть, суммарным числом сравнений в первом и втором while). Требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений. [Задача на астр](#).

- **Формат входного файла (input.txt).** В первой строке находится единственное число n ($1 \leq n \leq 10^6$).
- **Формат выходного файла (output.txt).** Вывести перестановку чисел от 1 до n , на которой быстрая сортировка выполнит максимальное число сравнений. Если таких перестановок несколько, вывести любую из них.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

Листинг кода

```
import random

f1 = open("input.txt", "w")
s = int(input("1 - random, 2 - backwards, 3 - algorithm: "))
n = int(input())
a = []
f1.write(str(n) + "\n")
if s == 1:
    for i in range(n):
        f1.write(str(random.randint(1, 10000000)) + " ")
elif s == 2:
    for i in range(n):
        f1.write(str(n - i) + " ")
else:
    for i in range(1, n + 1):
        a.append(i)
    for i in range(2, n):
        a[i], a[i//2] = a[i//2], a[i]
    for x in a:
        f1.write(str(x) + " ")
```

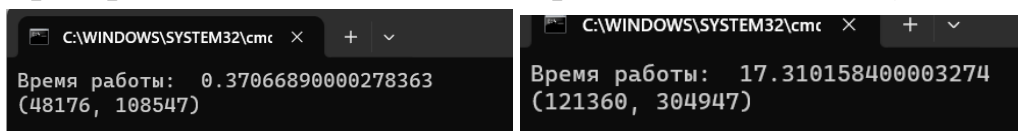
Объяснение

Я привёл код именно самого генератора. Для собственного удобства проведения большого количества тестов я сделал выбор варианта генератора.


Результат работы кода на примерах из текста задачи

```
3
1 3 2
```

Проверка задачи на (асгр и тд при наличии в задаче).



(пример вывода консоли для каких-то значений)

	10000	3000	1000
Случайный генератор	≈ 0.108 сек	≈0.0344	≈0.028
Список отсортирован в обратном порядке	≈0.069 сек	≈0.028	≈0.006
Худший случай		≈17.3	≈0.343

Вывод по задаче

Сначала планировалось сделать проверку на 10^4 , 10^5 и 10^6 , однако даже на 10^4 алгоритм при худшем случае сталкивается с превышением глубины рекурсии.

Задача №5. Индекс Хирша

5 задача. Индекс Хирша

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,

учёный с индексом h опубликовал как минимум h статей, на каждую из которых сослались как минимум h раз.

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

```
import time, tracemalloc, random

def qSortgood(a):
    if len(a) <= 1:
        return a
    pivot = a[random.randint(0, len(a) - 1)]
    l = [x for x in a if x > pivot]
    r = [x for x in a if x < pivot]
    mid = [x for x in a if x == pivot]
```

```

    return qSortgood(l) + mid + qSortgood(r)

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
citations = list(map(int, f1.readline().split(',')))
a = qSortgood(citations)
h = 0
for i, k in enumerate(a, 1):
    if i <= k:
        h = i
    else:
        break
f2.write(str(h))
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())

```

Объяснение

Весьма простой алгоритм, чтобы не проверять для каждого числа просто идём вниз по отсортированному в обратном порядке списку.

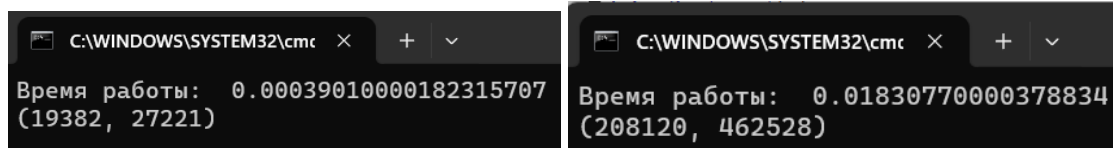
Результат работы кода на примерах из текста задачи

Файл	Изменить	Файл	Изменить
3,0,6,1,5		3	

Результат работы кода на максимальных значениях

417,795,900,466,473,398,730	831
616,864,819,672,478,43,766,	
,926,516,806,590,28,113,276	

Проверка задачи на (астр и тд при наличии в задаче).



(пример вывода консоли для примера из задачи и верхней границы диапазона значений)

	Время выполнения	Затраты памяти
Пример из задачи	≈ 0.00039 сек	27221 bytes ≈ 26.7 Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈ 0.018 сек	462528bytes ≈ 451.7 Kb

Вывод по задаче

Ознакомился с задачей индекса Хирша.

Вывод

Научился решать данные типовые задачи и разобрался в использовании QuickSort.