

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

Отчет по лабораторной работе №2
по курсу «Алгоритмы и структуры данных»
Тема: Сортировка слиянием. Метод декомпозиции
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. Сортировка слиянием	3
Задача №5. Представитель большинства	8
Задача №6. Поиск максимальной прибыли	12
Дополнительные задачи	16
Задача №2. Сортировка слиянием+	16
Вывод	19

Задачи по варианту

Задача №1. Сортировка слиянием

1 задача. Сортировка слиянием

1. Используя *псевдокод* процедур Merge и Merge-sort из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python и проверьте сортировку, создав несколько случайных массивов, подходящих под параметры:
 - **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 2 \cdot 10^4$) — 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. 2сек.
 - Ограничение по памяти. 256 мб.
 2. Для проверки можно выбрать наихудший случай, когда сортируется массив размера $1000, 10^4, 10^5$ чисел порядка 10^9 , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний. Сравните, например, с сортировкой вставкой на этих же данных.
 3. Перепишите процедуру Merge так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива L или R скопированы обратно в массив A , после чего в этот массив копируются элементы, оставшиеся в непустом массиве.
- или перепишите процедуру Merge (и, соответственно, Merge-sort) так, чтобы в ней не использовались значения границ и середины - p, r и q .

Листинг кода

```
import time, tracemalloc

def merge(a, left, mid, right):
    n1 = mid - left + 1
    n2 = right - mid
    L = [float('inf')] * (n1 + 1)
    R = [float('inf')] * (n2 + 1)
    for i in range(n1):
        L[i] = a[left + i]
```

```

        for j in range(n2):
            R[j] = a[mid + 1 + j]
        i = 0
        j = 0
        for k in range(left, right + 1):
            if L[i] <= R[j]:
                a[k] = L[i]
                i += 1
            else:
                a[k] = R[j]
                j += 1

```

```

def mergeSort(a, left, right):
    if left < right:
        mid = (left + right) // 2
        mergeSort(a, left, mid)
        mergeSort(a, mid + 1, right)
        merge(a, left, mid, right)
    return a

```

```

def checkSorted(a):
    for i in range(1, len(a)):
        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 = int(f1.readline())
a = list(map(int, f1.readline().split()))
for x in mergeSort(a, 0, len(a) - 1):
    f2.write(str(x) + " ")
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())

```

```
print(checkSorted(mergeSort(a, 0, len(a) - 1)))
```

Объяснение

Сортировка слиянием – приведение в порядок по порядку. Чтобы удобнее инициализировать не используя просто значение 10^{10} я решил использовать значение бесконечности.

Проверка задачи на результат работы кода.

The screenshots show the following results:

- 1000:** Время работы: 0.020688700024038553 (93838, 108466). True. The array is sorted from 0 to 999.
- 10000:** Время работы: 0.3110948000103235 (451183, 990376). True. The array is sorted from 10000 to 99999.
- 100000:** Время работы: 3.73579459998291 (3650691, 10210771). True. The array is sorted from 100000 to 999999.

(пример вывода консоли для лучшего случая на 1000, худшего на 10000 и среднего на 100000, а также отдельно для варианта на $2 \cdot 10^4$)

The screenshot shows the following result:

- 20000:** Время работы: 0.0705232999753207 (108022, 222690). True. The array is sorted from 20000 to 19999.

	1000	10^4	10^5
Лучший случай	≈ 0.02 сек	≈ 0.3 сек	≈ 3.65 сек
Худший случай	≈ 0.02 сек	≈ 0.3 сек	≈ 3.65 сек
Средний случай	≈ 0.03 сек	≈ 0.3 сек	≈ 3.7 сек

Пункт 3

```
import time, tracemalloc

def merge(a, left, mid, right):
    n1 = mid - left + 1
    n2 = right - mid
    L = [0] * n1
    R = [0] * n2
    for i in range(n1):
        L[i] = a[left + i]
    for j in range(n2):
        R[j] = a[mid + 1 + j]
    i = 0
    j = 0
    k = left
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            a[k] = L[i]
            i += 1
        else:
            a[k] = R[j]
            j += 1
        k += 1
    while i < n1:
        a[k] = L[i]
        i += 1
        k += 1
    while j < n2:
        a[k] = R[j]
        j += 1
        k += 1

def mergeSort(a, left, right):
    if left < right:
        mid = (left + right) // 2
        mergeSort(a, left, mid)
```

```

        mergeSort(a, mid + 1, right)
        merge(a, left, mid, right)
    return a

def checkSorted(a):
    for i in range(1, len(a)):
        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 = int(f1.readline())
a = list(map(int, f1.readline().split()))
for x in mergeSort(a, 0, len(a) - 1):
    f2.write(str(x) + " ")
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())
print(checkSorted(mergeSort(a, 0, len(a) - 1)))

```

Вывод по задаче

Использовал merge. Код затрачивает оптимальное количество времени. Вариант без сигнальных значений должен быть быстрее, т.к. не производится сравнение оставшихся элементов другого списка с сигнальным.

Задача №5. Представитель большинства

5 задача. Представитель большинства

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность A элементов a_1, a_2, \dots, a_n , и нужно проверить, содержит ли она элемент, который появляется больше, чем $n/2$ раз. Наивный метод это сделать:

```
Majority(A):  
for i from 1 to n:  
    current_element = a[i]  
    count = 0  
    for j from 1 to n:  
        if a[j] = current_element:  
            count = count+1  
    if count > n/2:  
        return a[i]  
return "нет элемента большинства"
```

Очевидно, время выполнения этого алгоритма квадратично. Ваша цель - использовать метод "Разделяй и властвуй" для разработки алгоритма проверки, содержится ли во входной последовательности элемент, который встречается больше половины раз, за время $O(n \log n)$.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n положительных целых чисел, по модулю не превосходящих 10^9 , $0 \leq a_i \leq 10^9$.
- **Формат выходного файла (output.txt).** Выведите 1, если во входной последовательности есть элемент, который встречается строго больше половины раз; в противном случае - 0.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример 1:

input.txt	output.txt
5 2 3 9 2 2	1

Число "2" встречается больше $5/2$ раз.

- Пример 2:

input.txt	output.txt
4 1 2 3 4	0

Нет элемента, встречающегося больше $n/2$ раз.

Листинг кода

```
import time, tracemalloc

def is_majority_element(a, x, left, right):
    if left == right:
        return a[left] == x
    mid = (left + right) // 2
    left_majority = is_majority_element(a, x, left,
mid)
    right_majority = is_majority_element(a, x, mid +
1, right)
    if left_majority and right_majority:
        return True
    left_count = 0
    right_count = 0
    for i in range(left, mid + 1):
        if a[i] == x:
            left_count += 1
    for i in range(mid + 1, right + 1):
        if a[i] == x:
            right_count += 1
    return left_count + right_count > (right - left
+ 1) // 2

def find_majority_element(a):
    if len(a) == 0:
        return 0
    x = a[0]
    k = 1
    for i in range(1, len(a)):
        if a[i] == x:
            k += 1
        else:
            k -= 1
        if k == 0:
            x = a[i]
            k = 1
    if is_majority_element(a, x, 0, len(a) - 1):
```

```

        return x
    else:
        return 0

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = list(map(int, f1.readline().split()))
f2.write(str(find_majority_element(a)))
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())

```

Объяснение

Сначала проходил по списку, и убирал пары чисел. Если какое-то число встречается больше половины раз, то других элементов будет недостаточно, чтобы его count был 0, и мы бы нашли какое-то другое число. Даже если в какой-то момент времени будет не нужным нам числом, под конец оно обязательно им станет. После этого мы просто проверяем, является ли число нужным нам, разделяя список на подсписки, проверяя их и т.д.

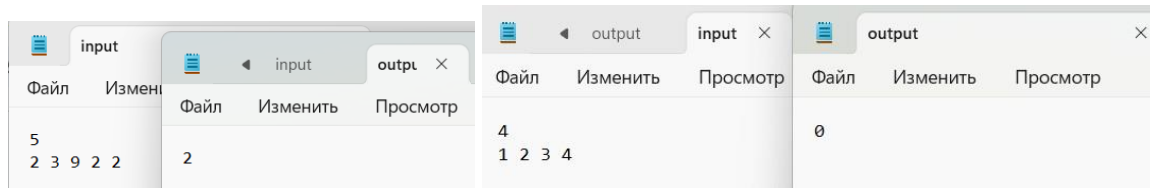
```

def is_majority_element(a, x, left, right):
    k = 0
    for y in a:
        if x == y:
            k += 1
    if k > len(a) // 2:
        return True
    else:

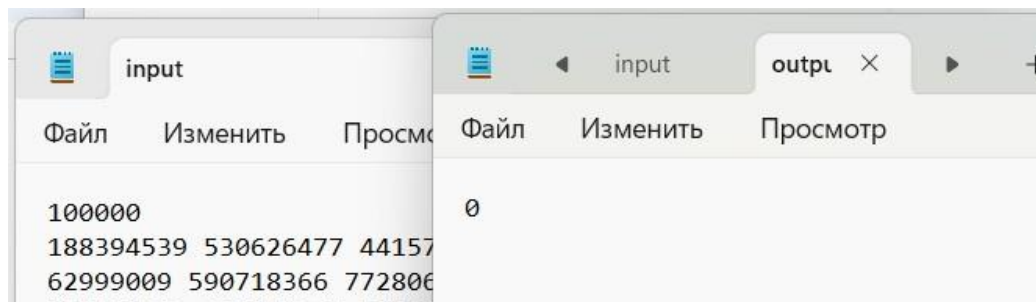
```

```
return False
```

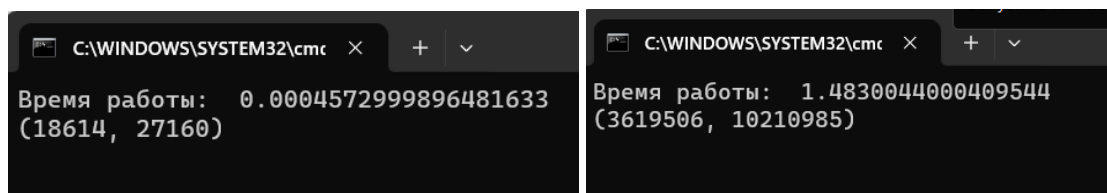
Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных значениях



Проверка задачи на (асгр 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	≈ 0.00045 сек	27168 bytes $\approx 26,5$ Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈ 1.48 сек	10210985 bytes ≈ 9981 Kb

Вывод по задаче

Используя псевдокод процедур Merge и Merge-sort, смог написать программу сортировки слиянием на Python.

Задача №6. Поиск максимальной прибыли

Используя *псевдокод* процедур Find Maximum Subarray и Find Max Crossing Subarray из презентации к Лекции 2 (страницы 25-26), напишите программу поиска максимального подмассива.

Примените ваш алгоритм для ответа на следующий вопрос. Допустим, у нас есть данные по акциям какой-либо фирмы за последний месяц (год, или иной срок).

Проанализируйте этот срок и выдайте ответ, в какой из дней при покупке единицы акции данной фирмы, и в какой из дней продажи, вы бы получили максимальную прибыль? Выдайте дату покупки, дату продажи и максимальную прибыль.

Вы можете использовать любые данные для своего анализа. Например, я набрала в Google "акции" и мне поиск выдал акции Газпрома, [тут](#) - можно скачать информацию по стоимости акций за любой период. (Перейдя по ссылке, нажмите на вкладку "Настройки" → "Скачать")

Соответственно, вам нужно только выбрать данные, посчитать *изменение цены* и применить алгоритм поиска максимального подмассива.

- **Формат входного файла** в данном случае на ваше усмотрение.
- **Формат выходного файла (output.txt)**. Выведите название фирмы, рассматриваемый вами срок изменения акций, дату покупки и дату продажи единицы акции, чтобы получилась максимальная выгода; и сумма этой прибыли.

Листинг кода

```
import time, tracemalloc

def find_max_crossing_subarray(a, low, mid, high):
    leftsum = float('-inf')
    s = 0
    maxleft = 0
    for i in range(mid, low - 1, -1):
        s += a[i]
        if s > leftsum:
            leftsum = s
            maxleft = i
    rightsum = float('-inf')
    s = 0
    maxright = 0
    for i in range(mid + 1, high + 1):
        s += a[i]
```

```

        if s > rightsum:
            rightsum = s
            maxright = i
    return maxleft, maxright, leftsum + rightsum

```

```

def find_maximum_subarray(a, low, high):
    if high == low:
        return low, high, a[low]
    else:
        mid = (low + high) // 2
        left_low, left_high, left_sum =
find_maximum_subarray(a, low, mid)
        right_low, right_high, right_sum =
find_maximum_subarray(a, mid + 1, high)
        cross_low, cross_high, cross_sum =
find_max_crossing_subarray(a, low, mid, high)
        if left_sum >= right_sum and left_sum >=
cross_sum:
            return left_low, left_high, left_sum
        elif right_sum >= left_sum and right_sum >=
cross_sum:
            return right_low, right_high, right_sum
        else:
            return cross_low, cross_high, cross_sum

```

```

tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = f1.readline()
dates = []
prices = []
diffs = [0]
while n != "\n":
    date, price = n.split()
    dates.append(date)
    prices.append(float(price))

```

```

n = f1.readline()
for i in range(1, len(prices)):
    diffs.append(prices[i] - prices[i-1])
zakupaem,        prodaem,        profit        =
find_maximum_subarray(diffs, 0, len(diffs) - 1)
s = "SBERBANK: " + str(dates[0]) + "-" +
str(dates[-1]) + " buy: " + str(dates[zakupaem -
1]) + " sell: " + str(dates[prodaem]) + " profit: "
+ str(profit)
f2.write(s)
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())

```

Объяснение

Взял данные вот отсюда: <https://www.finam.ru/quote/moex/sber/export/>

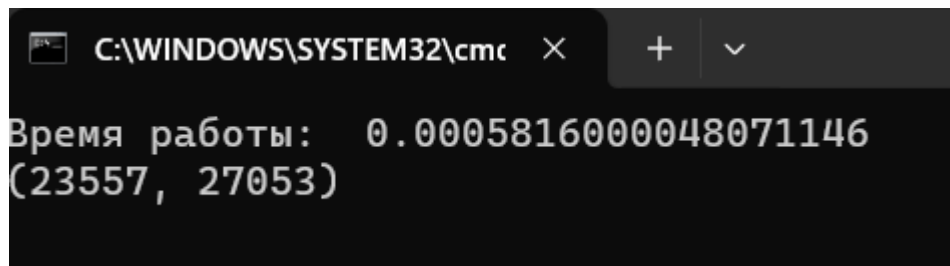
Задачу получилось решить, используя опыт из предыдущих работ, лекций и доп. источников.

Результат работы кода на примерах из текста задачи

The image shows two terminal windows. The left window displays a list of stock prices for Sberbank (ticker 231) from 2023 to 2022. The right window shows the program's output, which is a string representing the best trading strategy found: 'SBERBANK: 231023-231122 buy: 231101 sell: 231114 profit: 15.370000000000005'.

Year	Price
231023	270.4700000
231024	270.0300000
231025	271.5000000
231026	274.1000000
231027	270.0000000
231030	269.9000000
231031	269.9100000
231101	268.2900000
231102	270.0100000
231103	269.0700000
231106	269.0000000
231107	273.0800000
231108	273.5300000
231109	278.6000000
231110	276.9900000
231113	280.4000000
231114	283.6600000
231115	280.8700000
231116	282.4000000
231117	279.6900000
231120	281.8000000
231121	282.2900000
231122	283.1400000

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



```
C:\WINDOWS\SYSTEM32\cmd.exe
Время работы: 0.0005816000048071146
(23557, 27053)
```

	Время выполнения	Затраты памяти
Пример из задачи	≈ 0.00058 сек	27053 bytes $\approx 26,42$ Kb

Вывод по задаче

Научился делать сортировку другим способом, узнал новый алгоритм.

Дополнительные задачи

Задача №2. Сортировка слиянием+

2 задача. Сортировка слиянием+

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания с помощью сортировки слиянием.

Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим Вас, после каждого осуществленного слияния (то есть, когда соответствующий подмассив уже отсортирован!), выводить индексы граничных элементов и их значения.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .
- **Формат выходного файла (output.txt).** Выходной файл состоит из нескольких строк.
 - В последней строке выходного файла требуется вывести отсортированный в порядке неубывания массив, данный на входе. Между любыми двумя числами должен стоять ровно один пробел.
 - Все предшествующие строки описывают осуществленные слияния, по одному на каждой строке. Каждая такая строка должна содержать по четыре числа: I_f, I_l, V_f, V_l , где I_f — индекс начала области слияния, I_l — индекс конца области слияния, V_f — значение первого элемента области слияния, V_l — значение последнего элемента области слияния.
 - Все индексы начинаются с единицы (то есть, $1 \leq I_f \leq I_l \leq n$). **Индексы области слияния должны описывать положение области слияния в исходном массиве!** Допускается не выводить информацию о слиянии для подмассива длиной 1, так как он отсортирован по определению.

Ограничение по времени. 2сек.

Ограничение по памяти. 256 мб.

Листинг кода

```
import time, tracemalloc
```

```
def merge(a, left, mid, right):  
    n1 = mid - left + 1  
    n2 = right - mid  
    L = [0] * n1  
    R = [0] * n2  
    for i in range(n1):
```



```

        L[i] = a[left + i]
    for j in range(n2):
        R[j] = a[mid + 1 + j]
    i = 0
    j = 0
    k = left
    while i < n1 and j < n2:
        if L[i] <= R[j]:
            a[k] = L[i]
            i += 1
        else:
            a[k] = R[j]
            j += 1
        k += 1
    while i < n1:
        a[k] = L[i]
        i += 1
        k += 1
    while j < n2:
        a[k] = R[j]
        j += 1
        k += 1

def mergeSort(a, left, right):
    if left < right:
        mid = (left + right) // 2
        mergeSort(a, left, mid)
        mergeSort(a, mid + 1, right)
        f2.write(str(left + 1) + " " + str(right +
1) + " " + str(a[left]) + " " + str(a[right]) +
"\n")
        merge(a, left, mid, right)
    return a

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

```

```

return True

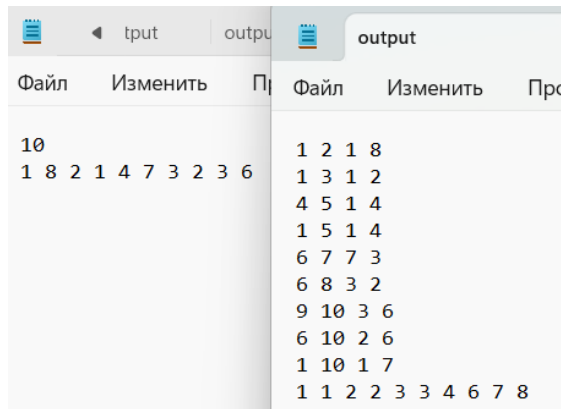
tracemalloc.start()
t_start = time.perf_counter()
f1 = open("input.txt", "r")
f2 = open("output.txt", "w")
n = int(f1.readline())
a = list(map(int, f1.readline().split()))
for x in mergeSort(a, 0, len(a) - 1):
    f2.write(str(x) + " ")
print("Время работы: ", (time.perf_counter() -
t_start))
print(tracemalloc.get_traced_memory())
print(checkSorted(a))

```

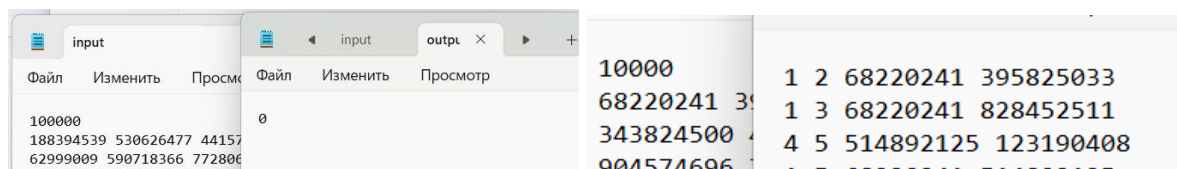
Объяснение

Использовал алгоритмы, сортировки и код из предыдущих работ и т.п.

Результат работы кода на примерах из текста задачи



Результат работы кода на максимальных значениях



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

```
Время работы: 0.0005224999913480133
(19567, 27181)
True
```

```
Время работы: 0.3838546999904793
(394378, 1047952)
True
```

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

	Время выполнения	Затраты памяти
Пример из задачи	≈ 0.0005 сек	27181 bytes $\approx 26,5$ Kb
Верхняя граница диапазона значений входных данных из текста задачи	≈ 0.38 сек	1047952 bytes ≈ 1023 Kb

Вывод по задаче

Нечего особо говорить, просто выводим промежуточное действие.

Вывод

В ходе выполнения лабораторной работы понял, как работает сортировка слиянием, научился правильно её использовать и применять.