

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 7

Выполнила:
Пожидаева Е.Р.

Санкт-Петербург
2024 г.

Содержание отчета

Оглавление

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка слиянием	3
Задача №2. Сортировка слиянием+	6
Дополнительные задачи	12
Задача №3. Бинарный поиск	12
Задача №4. Число инверсий	15
Задача №5. Представитель большинства	19
Задача №8. Умножение многочленов	22

Задачи по варианту

Задача №1. Сортировка слиянием

Текст задачи.

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.

Листинг кода.

```
def merge_sort(A, p, r):  
  
    if p < r:  
  
        q = (p + r) // 2  
  
        merge_sort(A, p, q)  
  
        merge_sort(A, q+1, r)  
  
        merge(A, p, q, r)  
  
    return A  
  
def merge(A, p, q, r):  
  
    n1 = q - p + 1  
  
    n2 = r - q  
  
    L = [0] * (n1)  
  
    R = [0] * (n2)  
  
    for i in range(0, n1):  
  
        L[i] = A[p + i]  
  
    for j in range(0, n2):
```

```

        R[j] = A[q + j + 1]

i,j = 0,0 # war

k = p

while i < len(L) and j < len(R) and k <= r:

    if L[i] <= R[j]:

        A[k] = L[i]

        i=i+1

    else:

        A[k] = R[j]

        j = j + 1

    k = k + 1

if i < len(L):

    while i < len(L) and k <= r:

        A[k] = L[i]

        i=i+1

        k=k+1

if j < len(R):

    while j < len(L) and k <= r:

        A[k] = R[j]

        j=j+1

        k=k+1


with open('input.txt') as f:

    n = int(f.readline())

    stroka = f.readline()

stroka = stroka.split()

spisok = []

for i in stroka:

    spisok.append(int(i))

```

```
merge_sort(spisok, 0, len(spisok)-1)

f = open("output.txt", "w")

for i in spisok:
    f.write(str(i) + " ")

f.close()
```

Текстовое объяснение решения.

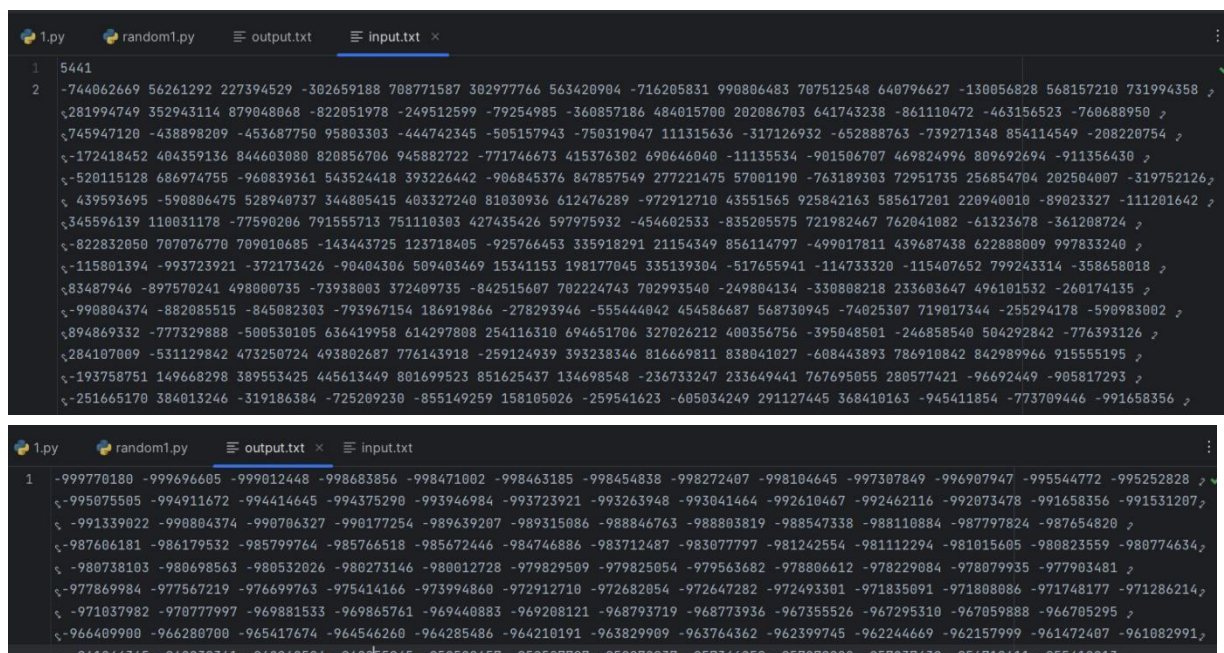
Код представляет функцию сортировки слиянием. Основная идея алгоритма состоит в том, чтобы разделить исходный список на две половины, отсортировать каждую половину отдельно и затем объединить их в отсортированный список.

Функция `merge_sort` принимает три аргумента: исходный список `A`, индекс первого элемента `p` и индекс последнего элемента `r`. Сначала функция проверяет, что индексы корректны ($p < r$), затем вычисляет середину списка `q`. Затем функция рекурсивно вызывает саму себя для сортировки первой половины списка (от `p` до `q`) и второй половины (от `q+1` до `r`). Наконец, функция вызывает функцию `merge` для объединения отсортированных половинок. Возвращается отсортированный список.

Функция `merge` принимает четыре аргумента: исходный список `A`, индексы начала (`p`), середины (`q`) и конца (`r`) отрезка, который нужно объединить. Сначала функция вычисляет размеры двух половинок (`n1` и `n2`) и создает два временных списка `L` и `R`. Затем функция копирует элементы первой половины списка `A` в список `L`, а элементы второй половины в список `R`. Далее используется цикл, в котором происходит сравнение и объединение элементов из списков `L` и `R` в список `A` до тех пор, пока все элементы не будут обработаны или пока не достигнут конец списка `A`. Если в списке `L` останутся элементы, то они просто копируются в список `A`. Аналогично, если в списке `R` останутся элементы, то они также копируются в список `A`.

В конце код открывает файл "input.txt" для чтения и считывает из него количество элементов в списке n и сам список stroka. Затем создается пустой список spisok, в который будут добавлены элементы списка stroka в виде целых чисел. Затем вызывается функция merge_sort для сортировки списка spisok. Далее открывается файл "output.txt" для записи и в него записываются отсортированные элементы из списка spisok через пробел. Затем файл закрывается.

Результат работы кода на максимальных и минимальных значениях



```
1.py random1.py output.txt input.txt
1 5441
2 -744062669 56261292 227394529 -302659188 708771587 302977766 563420904 -716205831 990806483 707512548 640796627 -130056828 568157210 731994358
3 281994749 352943114 879048068 -822051978 -249512599 -79254985 -360857186 484015700 202086703 641743238 -861110472 -463156523 -760688950
4 745947120 -438898209 -453687750 95803303 -444742345 -505157943 -750319047 111315636 -317126932 -652888763 -739271348 854114549 -208220754
5 -172418452 404359136 844603080 820856706 945882722 -771746673 415376302 690646040 -11135534 -901506707 469824996 809692694 -911356430
6 -520115128 686974755 -960839361 543524418 393226442 -906845376 847857549 277221475 57001190 -763189303 72951735 256854704 202504007 -319752126
7 439593695 -590806475 528940737 344805415 403327240 81030936 612476289 -972912710 43551565 925842163 585617201 220940010 -89023327 -111201642
8 345596139 110031178 -77590206 791555713 751110303 427435426 597975932 -454602533 -835205575 721982467 762041082 -61323678 -361208724
9 -822832050 707076770 709010685 -143443725 123718405 -925766453 335918291 21154349 856114797 -499017811 439687438 622888009 997833240
10 -115801394 -993723921 -372173426 -90404306 509403469 15341153 198177045 335139304 -517655941 -114733320 -115407652 799243314 -358658018
11 83487946 -897570241 498000735 -73938003 372409735 -842515607 702224743 702993540 -249884134 -330808218 233603647 496101532 -260174135
12 -990804374 -882085515 -845082303 -793967154 186919866 -278293946 -555444042 454586687 568730945 -74025307 719017344 -255294178 -590983002
13 894869332 -777329888 -500530105 636419958 614297808 254116310 694651706 327026212 400356756 -395048501 -246858540 504292842 -776393126
14 284107009 -531129842 473250724 493802607 776143918 -259124939 393238346 816669811 838041027 -608443893 786910842 842989966 915555195
15 -193758751 149668298 389553425 445613449 801699523 851625437 134698548 -236733247 233649441 767695055 280577421 -96692449 -905817293
16 -251665170 384013246 -319186384 -725209230 -855149259 158105026 -259541623 -605034249 291127445 368410163 -945411854 -773709446 -991658356
```

```
1.py random1.py output.txt input.txt
1 -999770180 -999696605 -999012448 -998683856 -998471002 -998463185 -998454838 -998272407 -998104645 -997307849 -996907947 -995544772 -995252828
2 -995075505 -994911672 -994414645 -994375290 -993946984 -993723921 -993263948 -993041464 -992610467 -992462116 -992073478 -991658356 -991531207
3 -991339022 -990804374 -990706327 -990177254 -989639207 -989315086 -988846763 -988803819 -988547338 -988110884 -987797824 -987654820
4 -987606181 -986179532 -985799764 -985766518 -985672446 -984746886 -983712487 -983077797 -981242554 -981112294 -981015605 -980823559 -980774634
5 -980738103 -980698563 -980532026 -980273146 -980012728 -979829509 -979825054 -979563682 -978806612 -978229084 -978079935 -977903481
6 -977869984 -977567219 -976699763 -975414166 -973994860 -972912710 -972682054 -972647282 -972493301 -971835091 -971808086 -971748177 -971286214
7 -971037982 -970777997 -969881533 -969865761 -969440883 -969208121 -968793719 -968773936 -967355526 -967295310 -967059888 -966705295
8 -966409900 -966280700 -965417674 -964546260 -964285486 -964210191 -963829909 -963764362 -962399745 -962244669 -962157999 -961472407 -961082991
9 -961044365 -960839361 -960242524 -960055945 -959528657 -959507787 -958878037 -957366259 -957278920 -957037638 -956712611 -956412013
```

Результат работы кода на максимальных и минимальных значениях max

```
1.py random1.py output.txt input.txt x
1 20000
2 -44745112 977987611 -226493966 -847682686 -507442952 424792672 -501700239 308212482 -340686158 -940510989 379040375 -405474075 -659997988 ,
  332715761 9877310 554435091 -457195712 -172794781 -486397317 -375470683 684321052 -522180213 -591043517 949214341 754529629 -62417054 ,
  813970685 -929091051 -616524766 427236105 315369976 30205623 116800376 -438833946 197074051 -22731700 -997160108 -712970380 -678340523 ,
  330873250 134458854 -994050492 438674823 121053613 117040694 600712918 714325980 78625611 -847444966 243738323 -556536893 -141754228 ,
  481982175 -358713490 -396659477 -150219903 -774720444 -594921251 -316105632 785242742 -341345916 -565361582 120285590 -213477740 817713133 ,
  645334251 -29011724 -63893322 -574648453 305670716 179384046 -738086795 706481691 -62684257 -238494458 -631766967 761640744 218618902 ,
  411728344 935994174 698499889 -667264868 -995055878 500016645 -842936578 119676968 485868324 -645293984 514423089 -311420568 -343599467 ,
  226995899 58066052 -572387226 -762322118 395866 -454321764 1557783 399928494 -848559841 897297722 789010696 672282633 -195039193 687717251 ,
  990295948 -130072492 388748058 391078223 -660717831 303874275 -159892288 -430026593 923205119 243822888 -877591731 277927056 110530194 ,
  477256838 -28127178 -676508686 -746744699 -608494062 -512953904 -714653473 -795358781 513686730 -803435834 -882530124 569643021 -904281785 ,
```

```
1.py random1.py output.txt x input.txt
1 -999677702 -999399655 -999392165 -999364605 -999143152 -999126508 -999039607 -998806873 -998764473 -998673293 -998534942 -998516845 -9984153
  998274498 -998218107 -998133065 -998011753 -997956752 -997668668 -997515215 -997357898 -997270029 -997201112 -997160108 -997090448 -996894
  996854464 -996790868 -996786631 -996511925 -996450362 -996107664 -995980384 -995962466 -995957817 -995904191 -995901111 -995887213 ,
  995765167 -995735234 -995609612 -995422656 -995398197 -995375946 -995270485 -995136011 -995091199 -995055878 -994880251 -994811928 -994776
  994771497 -994728904 -994522036 -994508337 -994391789 -994311807 -994296198 -994246477 -994222475 -994218819 -994050492 -994034286 ,
  994029470 -993904660 -993817962 -993752456 -993686853 -993651744 -993600054 -993480008 -993338743 -993201084 -993191004 -993164302 -993142
  993095206 -992931465 -992729086 -992728735 -992706196 -992637840 -992627394 -992564183 -992254888 -992026918 -991850641 -991737514 ,
  991685144 -991529487 -991494041 -991426503 -991389773 -991304313 -991278475 -991233159 -991074545 -990891615 -990773441 -990682297 -990573
  990468141 -990440223 -990418684 -990338237 -990304114 -990295948 -990205354 -989921489 -989915108 -989670194 -989646968 -989616111 ,
  989561479 -989499964 -989417980 -989375467 -989297107 -989112090 -989020922 -988696776 -988556878 -988360650 -988310347 -988267531 -988213
  988101435 -988059459 -987975358 -987971848 -987792122 -987730859 -987464720 -987390897 -987333116 -987155005 -987092983 -987080597 ,
```

min

```
1.py test.py random1.py output.txt input.txt x
1 2
2 -367073651 -394851710
```

```
1.py test.py random1.py output.txt x input.txt
1 -394851710 -367073651
```

	Затраты памяти (Мб)	Время выполнения (сек)
Нижняя граница диапазона значений входных данных из текста задачи	14.9453125	0.0010025501251220703
Рандомный пример	17.453125	0.11502194404602051
Верхняя граница диапазона значений входных данных из текста задачи	17.46484375	0.12552499771118164

Вывод по задаче:

Удалось поработать отсортировать числа с помощью метода слияния.

Задача №2. Сортировка слиянием+

Текст задачи.

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.

Листинг кода.

```
def merge_sort(A, p, r):
    if p < r:
        q = (p + r) // 2
        merge_sort(A, p, q)
        merge_sort(A, q + 1, r)
        merge(A, p, q, r)
    return A

def merge(A, p, q, r):
    n1 = q - p + 1
    n2 = r - q
    L = [0] * n1      #временные массивы
    R = [0] * n2

    for i in range(0, n1):      #копируем данные во временные массивы
        L[i] = A[p + i]
    for j in range(0, n2):
        R[j] = A[q + j + 1]
    i = j = 0
    k = p      # индекс объединенного подмассива

    while i < len(L) and j < len(R) and k <= r:
        if L[i] <= R[j]:
            A[k] = L[i]
            i = i + 1
        else:
            A[k] = R[j]
            j = j + 1
        k = k + 1
    if i < len(L):
        while i < len(L) and k <= r:
            A[k] = L[i]
            i = i + 1
            k = k + 1
    if j < len(R):
        while j < len(R) and k <= r:
            A[k] = R[j]
            j += 1
            k += 1
    f.write(str(p + 1) + " ")
    f.write(str(r + 1) + " ")
    f.write(str(A[p]) + " ")
```



```

        f.write(str(A[r]) + " ")
        f.write("\n")

with open("input.txt", "r") as f:
    n = int(f.readline())
    stroka = f.readline()
    stroka = stroka.split()
    spisok = []
    for i in stroka:
        spisok.append(int(i))

f = open('output.txt', 'w')
merge_sort(spisok, 0, len(spisok) - 1)
for i in spisok:
    f.write(str(i) + " ")
f.close()

```

Текстовое объяснение решения.

Функция `merge_sort` делит список пополам, рекурсивно сортирует каждую часть, а затем данные заносятся в один отсортированный список. А в функции `merge` реализуется слияние двух списков. После каждого шага выводим индексы граничных элементов и их значения.

Результат работы кода на случайном примере

2.py		input.txt		output.txt	
1	7				
2	4 2	11	4 56 77 8		

input.txt		output.txt		2.py	
1	1 2 2 4				
2	3 4 4 11				
3	1 4 2 11				
4	5 6 56 77				
5	5 7 8 77				
6	1 7 2 77				
7	2 4 4 8 11 56 77				

Результат работы кода на максимальных и минимальных значениях

max

```
2.py  input.txt  memory1.py  time1.py  output.txt  random1.py
1  200000
2  -26723121 640681313 -962045277 45754631 -589129693 502324151 249564471 236194674 137921099 447318310 858813316 -596471485
   574230086 870894581 -538803813 -267374805 190711059 -684139289 -549784835 -191317937 320175390 -725154031 -520954748
   591792460 -200628076 -485438157 -157704498 312721654 562768995 -809457825 -807713005 -413845754 275433677 687816464
   -79802245 517633828 -629860722 -886406464 -414328347 -993170816 -534768022 559601555 271849170 534932471 -895623992
   197212436 -965648866 725944626 764974918 -725999729 718966415 201995906 -720960862 287042032 -317390538 -130864777 3480032
   -891178522 -115710769 855223253 6569111 766123996 642676121 -859663009 -919098819 401450435 624924229 961881962 843389259
   608116077 180854781 114531414 -357188342 -177466041 108628223 -433970504 156431691 553298509 -601794071 -288828290
   -428661766 -913127243 839144388 413216096 722889370 -174199948 -774919341 -144104108 372937075 166449920 391234243
```

```
2.py  input.txt  memory1.py
1  1 2 -26723121 640681313
2  3 4 -962045277 45754631
3  1 4 -26723121 45754631
4  5 6 -589129693 502324151
5  5 7 -589129693 249564471
6  1 7 -26723121 249564471
7  8 9 236194674 236194674
8  8 10 236194674 447318310
9  11 12 858813316 858813316
10 11 13 858813316 858813316
11 8 13 236194674 858813316
12 1 13 -26723121 858813316
13 14 15 870894581 870894581
14 14 16 870894581 870894581
15 17 18 190711059 190711059
16 17 19 190711059 190711059
17 14 19 870894581 190711059
18 20 21 -191317937 320175390
19 20 22 -191317937 320175390
```

min

```
2.py  input.txt
1  1
2  5
```

```
2.py  input.txt  memory1.py  time1.py  output.txt
1  5
```

	Время выполнения (сек)	Затраты памяти (Мб)
Нижняя граница диапазона значений входных данных из текста задачи	0.04832768440246582	13.87890625

Рандомный пример	0.09985685348510742	14.11328125
Верхняя граница диапазона значений входных данных из текста задачи	1.6158428192138672	33.5859375

Вывод по задаче:

Еще раз удалось поработать с методом слияния.

Дополнительные задачи

Задача №3. Число инверсий

Текст задачи.

Инверсией в последовательности чисел A называется такая ситуация, когда $i < j$, а $A_i > A_j$. Количество инверсий в последовательности в некотором роде определяет, насколько близка данная последовательность к отсортированной. Например, в отсортированном массиве число инверсий равно 0, а в массиве, отсортированном наоборот - каждые два элемента будут составлять инверсию (всего $n(n-1)/2$).

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем.

Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

- **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).** В выходной файл надо вывести число инверсий в массиве.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

Листинг кода.

```
inv_count = 0
def merge_sort(A):
    global inv_count
    if len(A) < 2:
        return A
    else:
        q = len(A) // 2
        L = merge_sort(A[:q])
        R = merge_sort(A[q:])
        sorted_A = merge(L, R)
        return sorted_A

def merge(L, R):
    global inv_count
    result = []
    i, j = 0, 0
    while i < len(L) and j < len(R):
        if L[i] <= R[j]:
            result.append(L[i])
            i += 1
        else:
            result.append(R[j])
            inv_count += len(L) - i
            j += 1
    if i < len(L):
        result += L[i:]
```

```
if j < len(R):  
    result += R[j:]  
return result  
  
with open('input.txt') as f:  
    n = int(f.readline())
```

```

    stroka = f.readline()
    stroka = stroka.split()
    spisok = []
    for i in stroka:
        spisok.append(int(i))

f = open("output.txt", "w")
merge_sort(spisok)
f.write(str(inv_count))

```

Текстовое объяснение решения.

Присваиваем переменной `inv_count` значение 0 и в функции `merge_sort` делаем ее глобальной, чтобы иметь к ней доступ из другой функции.

Функция `merge_sort` рекурсивно делит список `A` пополам и вызывает себя для каждой половины, пока не дойдет до базового случая (когда длина списка меньше 2). Функция `merge` объединяет отсортированные подсписки `L` и `R` в порядке возрастания, подсчитывая при этом количество инверсий.

Результат работы кода на примерах из текста задачи

1	10
2	1 8 2 1 4 7 3 2 3 6

1	17
---	----

Результат работы кода на максимальных и минимальных значениях

max

1	200000
2	799987871 -748602639 -387300863 939829136 716196419 505547193 622779493 -998066771 49953842 -689844560 -490747827 -5981253467747 928757999 -104460547 -212771155 -244348106 429411494 -557381965 332995679 708128177 920683781 477439140 670182604 -155052069 -169312330 875337215 847436377 -732087272 294987593 -784078992 -381026757 -154718750 818158486 -763696114 300128841 359517980 -471899758 -259914729 494010753 -583565065 -924329158 137904223 394469513 936220610 -513209881 623493037 716203594 768632760 233015984 247072377 -452567365 336666473 -118375796 -204608457 241389938 9501952493488 333994894 -986516459 -446550054 760300513 -761824970 -584260796 236143975 286239811 -872082633 -330952329 -168480220 -255146704 716803019 770673444 -262343627 -334703312 406876852 44566283 580758332 638735742 366040896 4648614981860 713154660 -574182087 182415573 -901079871 -536502798 856591558 -505607587 -392761898 -9587363 -968356224 320493633 -936557387 -690418543 -786838566 843004850 -376587671 536822565 169091124 -803531271 569607696 822340398 972460288 -248073224 -204776893 385121086 908963719 341391323 -98983460 -647576465 522409374 -319612208 -496410861 07220205 -7168174 -68087511 -110518048 -570175084 -300951115 781385873 -250518218 -222010718 -681805173 -94078824

1	9984883521
---	------------

min

```

1 1
2 65

```

```

1 p

```

	Время выполнения (сек)	Затраты памяти (Мб)
Нижняя граница диапазона значений входных данных из текста задачи	0.014140605926513672	14.49609375
Пример из задачи	0.0009992122650146484	14.34375
Верхняя граница диапазона значений входных данных из текста задачи	1.933122158050537	39.75

Вывод по задаче:

Удалось поработать с инверсиями.

Задача №4. Бинарный поиск

Текст задачи.

В этой задаче вы реализуете алгоритм бинарного поиска, который позволяет очень эффективно искать (даже в огромных) списках при условии, что список отсортирован. Цель - реализация алгоритма двоичного (бинарного) поиска.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве, и последовательность $a_0 < a_1 < \dots < a_{n-1}$ из n **различных** положительных целых чисел в порядке возрастания, $1 \leq a_i \leq 10^9$ для всех $0 \leq i < n$. Следующая строка содержит число k , $1 \leq k \leq 10^5$ и k положительных целых чисел b_0, \dots, b_{k-1} , $1 \leq b_j \leq 10^9$ для всех $0 \leq j < k$.
- **Формат выходного файла (output.txt).** Для всех i от 0 до $k - 1$ вывести индекс $0 \leq j \leq n - 1$, такой что $a_i = b_j$ или -1, если такого числа в массиве нет.

Листинг кода.

```
def Binary_search(A, low, high, V):
    if high < low:
        return -1
    mid = low + (high - low) // 2
    if V == A[mid]:
        return mid
    elif V < A[mid]:
        return Binary_search(A, low, mid - 1, V)
    else:
        return Binary_search(A, mid + 1, high, V)

with open('input.txt') as f:
    n = int(f.readline())
    a = f.readline().split()
    k = int(f.readline())
    b = f.readline().split()
    spisok = []
    for i in a:
        spisok.append(int(i))
    find = []
    for i in b:
        find.append(int(i))
    itog = []
    for j in find:
        itog.append(Binary_search(spisok, 0, len(spisok) - 1, j))
```



```
f = open("output.txt", "w")
for i in itog:
    f.write(str(i) + " ")
f.close()
```

Текстовое объяснение решения.

В функции `binary_search` принимаются следующие параметры: `list` (список, в котором будет выполняться поиск), `first` (индекс первого элемента в текущем диапазоне поиска), `last` (индекс последнего элемента в текущем диапазоне поиска), `midelement` (элемент, который нужно найти). Внутри функции `binary_search` проверяется случай, когда `first` больше `last`. Это означает, что диапазон поиска стал пустым, и искомый элемент не найден. В этом случае функция возвращает `-1`. Вычисляется средний индекс `mid` в текущем диапазоне поиска. Проверяется, равен ли `midelement` элементу с индексом `mid` в списке. Если да, возвращается индекс `mid`, и поиск успешно завершается. Если `midelement` меньше элемента с индексом `mid`, то искомый элемент находится в левой половине диапазона. Рекурсивно вызывается `binary_search` для поиска в левой половине, где `last` становится `mid - 1`. Если `midelement` больше элемента с индексом `mid`, то искомый элемент находится в правой половине диапазона. Рекурсивно вызывается `binary_search` для поиска в правой половине, где `first` становится `mid + 1`.

Результат работы кода на примерах из текста задачи

input.txt	output.txt	4.py
1 5		
2 1 5 8 12 13		
3 5		
4 8 1 23 1 11		

input.txt	output.txt	4.py
1 2 0 -1 0 -1		

Результат работы кода на максимальных и минимальных значениях
min

input.txt	output.txt	4.py
1 1		
2 548		
3 1		
4 7		

input.txt	output.txt	4.py
1 -1		

[illegible]

	Время выполнения (сек)	Затраты памяти (Мб)
Нижняя граница диапазона значений входных данных из текста задачи	0.0009992122650146484	14.4453125
Пример из задачи	0.0019881725311279297	14.4609375
Верхняя граница диапазона значений входных данных из текста задачи	0.7562649250030518	38.7578125

Вывод по задаче:

Удалось поработать с бинарным поиском.

Задача №5. 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 $O(n \log n)$.

Листинг кода:

```
def majority(A, left, right):
    if left > right:
        return 0
    if left == right:
        return A[left]

    middle = (left + right) // 2

    left_majority = majority(A, left, middle)
    right_majority = majority(A, middle + 1, right)

    if left_majority == right_majority:
        return left_majority

    left_count = sum(1 for i in range(left, right + 1) if A[i] == left_majority)
    if left_count > (right - left + 1) // 2:
```

```

        return left_majority

    right_count = sum(1 for i in range(left, right +
1) if A[i] == right_majority)
    if right_count > (right - left + 1) // 2:
        return right_majority

    return 0

with open("input.txt", "r") as input_file:
    n = int(input_file.readline())
    A = list(map(int, input_file.readline().split()))

result = majority(A, 0, n - 1)

with open("output.txt", "w") as output_file:
    if result != 0:
        output_file.write("1")
    else:
        output_file.write("0")

```

Текстовое объяснение решения:

Данный код реализует алгоритм для нахождения элемента массива, который встречается в нем более чем в половине случаев.

Функция `majority` принимает на вход массив `A`, индексы `left` и `right`, которые ограничивают текущий подмассив, соответствующий рекурсивному вызову.

Сначала проверяется базовый случай: если `left` больше `right`, значит, текущий подмассив пустой, и возвращается значение 0.

Если `left` равно `right`, значит, текущий подмассив состоит из одного элемента, и возвращается этот элемент.

Далее находится средний индекс `middle` как среднее значение от `left` и `right`. Затем рекурсивно вызывается функция `majority` для левой половины массива и правой половины массива.

Далее проверяется условие, если левый и правый элементы массива соответствуют одному и тому же числу, то этот элемент является большинством, и он возвращается.

Затем подсчитывается количество левого элемента в промежутке от `left` до `right`. Если это количество больше, чем половина длины промежутка, то левый элемент является большинством, и он возвращается.

Затем аналогично проверяется количество правого элемента в промежутке от `left` до `right`. Если это количество больше, чем половина длины промежутка, то правый элемент является большинством, и он возвращается.

Если ни одно из предыдущих условий не выполнилось, то возвращается значение 0, что означает, что среди элементов в промежутке нет элемента, встречающегося более чем в половине случаев.

В конце программа считывает входные данные из файла `"input.txt"`, вызывает функцию `majority` и записывает результат в файл `"output.txt"`. Если результат не равен 0, то записывается `"1"` в файл, иначе записывается `"0"`.

Примеры:

```
1.3.py 2.py 3.py 4.py вторая лаба/6.py 5.py input.txt x
1 5
2 2 3 9 2 2
```

```
1.3.py 2.py 3.py 4.py вторая лаба/6.py 5.py input.txt output.txt x
1 1
```

```
1.3.py 2.py 3.py 4.py вторая лаба/6.py 5.py input.txt x
1 4
2 1 2 3 4
```

```
1.3.py 2.py 3.py 4.py вторая лаба/6.py 5.py input.txt output.txt x
1 0
```

Задача №8. Умножение многочленов

Текст задачи.

Выдающийся немецкий математик Карл Фридрих Гаусс (1777—1855) заметил, что хотя формула для произведения двух комплексных чисел $(a + bi)(c + di) = ac - bd + (bc + ad)i$ содержит *четыре* умножения вещественных чисел, можно обойтись и *тремя*: вычислим ac, bd и $(a + b)(c + d)$ и воспользуемся тем, что $bc + ad = (a + b)(c + d) - ac - bd$.

Задача. Даны 2 многочлена порядка $n - 1$: $a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0$ и $b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \dots + b_1x + b_0$. Нужно получить произведение:

$c_{2n-2}x^{2n-2} + c_{2n-3}x^{2n-3} + \dots + c_1x + c_0$, где:

$$\begin{aligned}c_{2n-2} &= a_{n-1}b_{n-1} \\c_{2n-3} &= a_{n-1}b_{n-2} + a_{n-2}b_{n-1} \\&\dots \\c_2 &= a_2b_0 + a_1b_1 + a_0b_2 \\c_1 &= a_1b_0 + a_0b_1 \\c_0 &= a_0b_0\end{aligned}$$

Пример. Входные данные: $n = 3$, $A = (3, 2, 5)$, $B = (5, 1, 2)$

$$\begin{aligned}A(x) &= 3x^2 + 2x + 5 \\B(x) &= 5x^2 + x + 2 \\A(x)B(x) &= 15x^4 + 13x^3 + 33x^2 + 9x + 10\end{aligned}$$

Ответ: $C = (15, 13, 33, 9, 10)$.

- **Формат входного файла (input.txt).** В первой строке число n - порядок многочленов A и B . Во второй строке коэффициенты многочлена A через пробел. В третьей строке коэффициенты многочлена B через пробел.
- **Формат выходного файла (output.txt).** Ответ - одна строка, коэффициенты многочлена $C(x) = A(x)B(x)$ через пробел.
- Нужно использовать метод "Разделяй и властвуй". Подсказка: любой многочлен $A(x)$ можно разделить на 2 части, например, $A(x) = 4x^3 + 3x^2 + 2x + 1$ разделим на $A_1 = 4x + 3$ и $A_2 = 2x + 1$. И многочлен $B(x) = x^3 + 2x^2 + 3x + 4$ разделим на 2 части: $B_1 = x + 2$, $B_2 = 3x + 4$. Тогда произведение $C = A(x) * B(x) = (A_1B_1)x^n + (A_1B_2 + A_2B_1)x^{n/2} + A_2B_2$ - требуется 4 произведения (проверьте правильность данной формулы). Можно использовать формулу Гаусса и обойтись всего тремя произведениями.

Листинг кода.

```
f = open('input.txt', 'r')
n = int(f.readline())
a = list(map(int, f.readline().split()))
b = list(map(int, f.readline().split()))
res = [0] * (n+2)
for i in range(n):
    for j in range(n):
        res[i+j] += (a[i]*b[j])
f = open('output.txt', 'w')
f.write(' '.join(list(map(str, res))))
```

Текстовое объяснение решения.

Читаем входные данные, где у нас заложены порядок многочленов, коэффициенты многочлена A и B . Создаем список, который состоит из $n+2$ нулей. Проходимся по коэффициентам A и B и перемножаем их. На выходе получаем $n+2$ коэффициентов.

Результат работы кода на примерах из текста задачи

```
8.py input.txt output.txt
1 3
2 3 2 5
3 5 1 2
```

```
8.py input.txt output.txt
1 15 13 33 9 10
```

	Время выполнения (сек)	Затраты памяти (Мб)
Пример из задачи	0.039031028747558594	14.41796875

Вывод по задаче:

Удалось перемножить два многочлена.

Вывод

Поработала с сортировкой слиянием и методом «Разделяй и властвуй».