

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 14

Выполнила:  
Рудникова Виктория Олеговна  
К3125

Проверила:  
Артамонова В.Е.

Санкт-Петербург  
2022 г.

## **Содержание отчета**

<b>Содержание отчета</b>	<b>2</b>
<b>Задачи по варианту</b>	<b>3</b>
Задача №1. Название задачи [N баллов]	3
<b>Дополнительные задачи</b>	<b>4</b>
Задача №1. Название задачи [N баллов]	4
<b>Вывод</b>	<b>5</b>

## Задачи по варианту

### Задача №1. Сортировка слиянием

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.

```
from time import process_time
from tracemalloc import start, get_traced_memory
```

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        r = arr[:mid]
        l = arr[mid:]
        merge_sort(l)
        merge_sort(r)
        merge(arr, l, r)
    return arr
```

```
def merge(arr, l, r):
    i = j = k = 0
    while i < len(l) and j < len(r):
        if l[i] < r[j]:
            arr[k] = l[i]
            i += 1
        else:
```

```

        arr[k] = r[j]
        j += 1
        k += 1
    while i < len(l):
        arr[k] = l[i]
        i += 1
        k += 1
    while j < len(r):
        arr[k] = r[j]
        j += 1
        k += 1

```

```

if __name__ == '__main__':
    start()
    with open('input.txt', 'r') as f:
        n = int(f.readline())
        a = [int(i) for i in f.readline().split()]
        a = merge_sort(a)
    with open('output.txt', 'w+') as g:
        g.write(' '.join(str(i) for i in a))
    print('Time:', str(process_time()), 'sec')
    print('Memory usage:', str(get_traced_memory()[1] / 1024),
'KB')

```

В функции `merge_sort` реализуется сначала алгоритм деления списка на более маленькие списки, а затем вызывается функция `merge`, которая определяет, в каком порядке соединять маленькие списки в более большие.

Результат работы кода на примерах из текста задачи:

```

Дополнительные задачи\input.txt × Задачи по варианту\input.txt ×
1 10
2 15 15 15 11 15 234 15 556 15 2
Дополнительные задачи\output.txt × Задачи по варианту\output.txt ×
1 2 11 15 15 15 15 15 15 234 556

```

Результат работы кода на максимальных и минимальных значениях:

```

Дополнительные задачи\input.txt x Задачи по варианту\input.txt x
1 10
2 10 9 8 7 6 5 4 3 2 1
Дополнительные задачи\output.txt x Задачи по варианту\output.txt x
1 1 2 3 4 5 6 7 8 9 10

```

```

Дополнительные задачи\input.txt x Задачи по варианту\input.txt x
1 10
2 2 1 3 4 5 6 7 8 9 10
Дополнительные задачи\output.txt x Задачи по варианту\output.txt x
1 1 2 3 4 5 6 7 8 9 10

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB

Вывод по задаче: в задаче я использовала алгоритм сортировки слиянием.

## Задача №4. Бинарный поиск

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.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

```
from task1 import merge_sort
from time import process_time
from tracemalloc import start, get_traced_memory

def binary_search(arr, l, r, n):
    if l <= r:
        mid = (r - l) // 2 + 1
        if arr[mid] == n:
            return mid
        if arr[mid] > n:
            return binary_search(arr, l, mid - 1, n)
        return binary_search(arr, mid + 1, r, n)
    return -1

if __name__ == '__main__':
    start()
    with open('input.txt', 'r') as f:
        n1 = int(f.readline())
        a = [int(i) for i in f.readline().split()]
        n2 = int(f.readline())
        b = [int(i) for i in f.readline().split()]
        b = merge_sort(b)
        idxs = [binary_search(b, 0, len(b) - 1, i) for i in a]
```

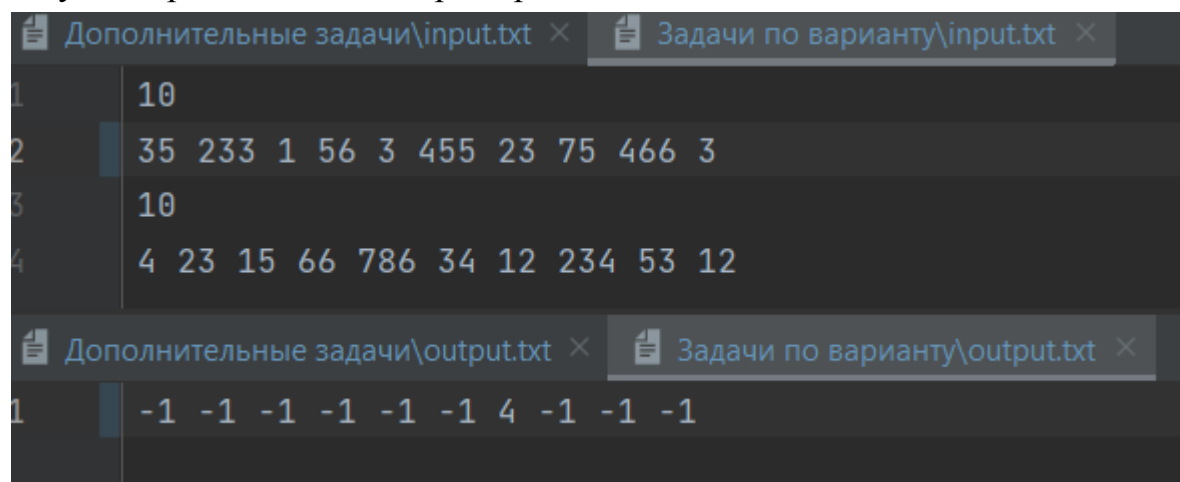
```

with open('output.txt', 'w+') as g:
    g.write(' '.join([str(i) for i in idxs]))
print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] / 1024),
'KB')

```

Массив сортируется с помощью сортировки слиянием, импортированной из первой задачи. В функции `binary_search` реализуется алгоритм бинарного поиска: искомое число сравнивается с серединой отсортированного списка, и если оно больше, то дальше поиск проводится только в левой половине массива, а если меньше - только в правой. Таким образом, когда в списке остаётся один элемент, выводится его индекс (а если не остаётся, значит искомого элемента нет).

Результат работы кода на примерах из текста задачи:



The screenshot shows a code editor with two tabs: 'Дополнительные задачи\input.txt' and 'Задачи по варианту\input.txt'. The first tab is active, showing the following input data:

```

1 10
2 35 233 1 56 3 455 23 75 466 3
3 10
4 4 23 15 66 786 34 12 234 53 12

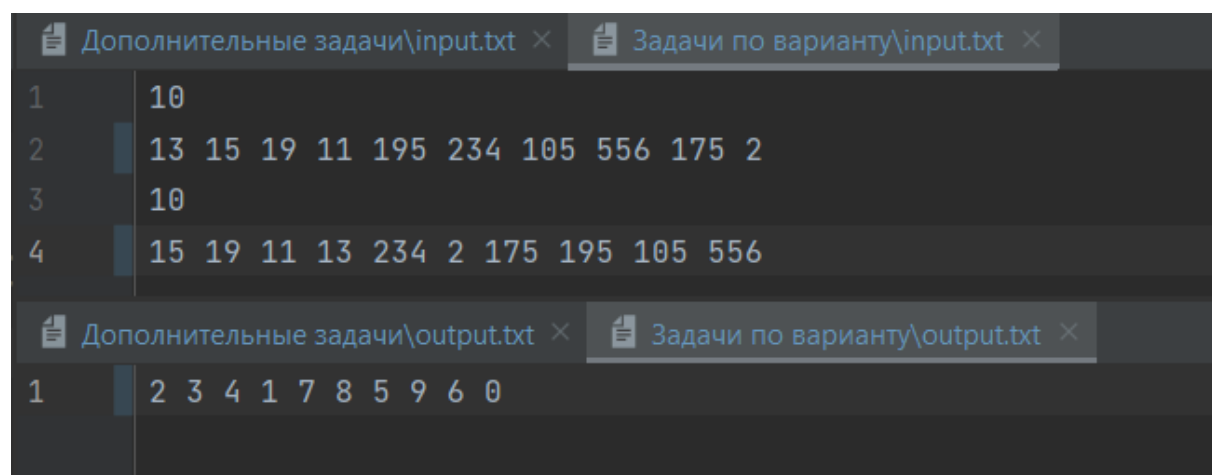
```

Below the input files, there are two output file tabs: 'Дополнительные задачи\output.txt' and 'Задачи по варианту\output.txt'. The first tab is active, showing the following output data:

```

1 -1 -1 -1 -1 -1 -1 4 -1 -1 -1

```



The screenshot shows a code editor with two tabs: 'Дополнительные задачи\input.txt' and 'Задачи по варианту\input.txt'. The second tab is active, showing the following input data:

```

1 10
2 13 15 19 11 195 234 105 556 175 2
3 10
4 15 19 11 13 234 2 175 195 105 556

```

Below the input files, there are two output file tabs: 'Дополнительные задачи\output.txt' and 'Задачи по варианту\output.txt'. The second tab is active, showing the following output data:

```

1 2 3 4 1 7 8 5 9 6 0

```

Результат работы кода на максимальных и минимальных значениях:

```

Дополнительные задачи\input.txt  Задачи по варианту\input.txt
1 1000
2 626489 83945 107588 838988 348904 452600 71584 98623 853825 85
3 1000
4 950637 53927 956530 98229 171270 512884 492249 797080 15724 33

Дополнительные задачи\output.txt  Задачи по варианту\output.txt
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1

```

```

input.txt
1 1
2 11
3 10
4 45 22 245 23 11 46 2 42 15 77

output.txt
1 1

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.1875 sec	17.5947265625 KB
Пример из задачи	0.125 sec	17.7724609375 KB
Пример из задачи	0.125 sec	17.7548828125 KB
Верхняя граница диапазона значений входных данных из	0.15625 sec	156.7783203125 KB



текста задачи		
---------------	--	--

Вывод по задаче: в этой задаче я реализовала алгоритм бинарного поиска.

## 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. 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.

```
from task1 import merge_sort
from time import process_time
from tracemalloc import start, get_traced_memory

def left_binary_search(arr, l, r, n, idx):
    if l <= r:
        mid = (r - l) // 2 + 1
        if arr[mid] == n:
            return left_binary_search(arr, l, mid - 1, n, mid)
        elif arr[mid] > n:
            return left_binary_search(arr, l, mid - 1, n, idx)
        return left_binary_search(arr, mid + 1, r, n, idx)
    return idx

def right_binary_search(arr, l, r, n, idx):
    if l <= r:
        mid = (r - l) // 2 + 1
        if arr[mid] == n:
            return right_binary_search(arr, mid + 1, r, n, mid)
        elif arr[mid] > n:
            return right_binary_search(arr, l, mid - 1, n, idx)
        return right_binary_search(arr, mid + 1, r, n, idx)
```

```
return idx
```

```
def majority(arr, n):  
    if left_binary_search(arr, 0, len(arr) - 1, n, -1) != -1  
    and right_binary_search(arr, 0, len(arr) - 1, n, -1) != -1 and  
    right_binary_search(arr, 0, len(arr) - 1, n, -1) -  
    left_binary_search(arr, 0, len(arr) - 1, n, -1) + 1 > len(arr)  
    // 2:  
        return 1  
    return 0
```

```
if __name__ == '__main__':  
    start()  
    with open('input.txt', 'r') as f:  
        n = int(f.readline())  
        a = [int(i) for i in f.readline().split()]  
        a = merge_sort(a)  
        flag = False  
        for i in a:  
            if majority(a, i) == 1:  
                flag = True  
        with open('output.txt', 'w+') as g:  
            if flag:  
                g.write('1')  
            else:  
                g.write('0')  
        print('Time:', str(process_time()), 'sec')  
        print('Memory usage:', str(get_traced_memory()[1] / 1024),  
            'KB')
```

Реализованы функции левого и правого бинарного поиска, которые ищут самый левый и самый правый подходящий элемент соответственно. Выполняется проверка, больше ли половины длины списка разность индекса самого правого и самого левого искомого элемента.

Результат работы кода на примерах из текста задачи:

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

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

Результат работы кода на максимальных и минимальных значениях:

```
Задачи по варианту\input.txt x
1 1000
2 13 29 20 30 6 29 23 9 9 17 14 7 26 12 10 9 18 6 13 4 24 1
output.txt x
1 0
```

```
Задачи по варианту\input.txt x
1 3
2 1 2 1
output.txt x
1 1
```

	Время выполнения	Затраты памяти
Нижняя граница	0.203125 sec	17.5087890625 KB

диапазона значений входных данных из текста задачи		
Пример из задачи	0.234375 sec	17.5166015625 KB
Пример из задачи	0.21875 sec	17.5126953125 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.265625 sec	59.958984375 KB

Вывод по задаче: в задаче я использовала алгоритмы левого и правого бинарного поиска.

## Дополнительные задачи

### Задача №3. Число инверсий

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 сортированном массиве число инверсий равно 0, а в массиве, сортированном 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.

```
from time import process_time
from tracemalloc import start, get_traced_memory
```

```
def inversions(arr):
    if len(arr) < 2:
        return 0
    mid = len(arr) // 2
    l = arr[:mid]
    r = arr[mid:]
    return inversions(l) + inversions(r) + merge(arr, l, r)
```

```
def merge(arr, l, r):
```

```

i = j = k = 0
while i < len(l) or j < len(r):
    if i == len(l):
        arr[i + j] = r[j]
        j += 1
    elif j == len(r):
        arr[i + j] = l[i]
        i += 1
    elif l[i] <= r[j]:
        arr[i + j] = l[i]
        i += 1
    else:
        arr[i + j] = r[j]
        k += len(l) - i
        j += 1
return k

```

```

if __name__ == '__main__':
    start()
    with open('input.txt', 'r') as f:
        n = int(f.readline())
        a = [int(i) for i in f.readline().split()]
        inv = inversions(a)
        with open('output.txt', 'w+') as g:
            g.write(str(inv))
        print('Time:', str(process_time()), 'sec')
        print('Memory usage:', str(get_traced_memory()[1] / 1024),
'KB')

```

В функции merge считаем инверсии, проверяя соседние элементы соединяемых списков. В функции inversions список делится на маленькие списки, для каждого из которых считается количество инверсий.

Результат работы кода на примерах из текста задачи:

```

Дополнительные задачи\input.txt x
1 10
2 1 8 2 1 4 7 3 2 3 6

output.txt x
1 17

```

Результат работы кода на максимальных и минимальных значениях:

```

Дополнительные задачи\input.txt x
1 1000 ✓
2 1000 999 998 997 996 995 994 993 992 991 990 989 988 987 986 985 984 983 982 981 980 979 978 977 976 975 974 973 972 971 970 969 968 967 966 965 964 963 962 961 960 959 958 957 956 955 954 953 952 951 950 949 948 947 946 945 944 943 942 941 940 939 938 937 936 935 934 933 932 931 930 929 928 927 926 925 924 923 922 921 920 919 918 917 916 915 914 913 912 911 910 909 908 907 906 905 904 903 902 901 900 899 898 897 896 895 894 893 892 891 890 889 888 887 886 885 884 883 882 881 880 879 878 877 876 875 874 873 872 871 870 869 868 867 866 865 864 863 862 861 860 859 858 857 856 855 854 853 852 851 850 849 848 847 846 845 844 843 842 841 840 839 838 837 836 835 834 833 832 831 830 829 828 827 826 825 824 823 822 821 820 819 818 817 816 815 814 813 812 811 810 809 808 807 806 805 804 803 802 801 800 799 798 797 796 795 794 793 792 791 790 789 788 787 786 785 784 783 782 781 780 779 778 777 776 775 774 773 772 771 770 769 768 767 766 765 764 763 762 761 760 759 758 757 756 755 754 753 752 751 750 749 748 747 746 745 744 743 742 741 740 739 738 737 736 735 734 733 732 731 730 729 728 727 726 725 724 723 722 721 720 719 718 717 716 715 714 713 712 711 710 709 708 707 706 705 704 703 702 701 700 699 698 697 696 695 694 693 692 691 690 689 688 687 686 685 684 683 682 681 680 679 678 677 676 675 674 673 672 671 670 669 668 667 666 665 664 663 662 661 660 659 658 657 656 655 654 653 652 651 650 649 648 647 646 645 644 643 642 641 640 639 638 637 636 635 634 633 632 631 630 629 628 627 626 625 624 623 622 621 620 619 618 617 616 615 614 613 612 611 610 609 608 607 606 605 604 603 602 601 600 599 598 597 596 595 594 593 592 591 590 589 588 587 586 585 584 583 582 581 580 579 578 577 576 575 574 573 572 571 570 569 568 567 566 565 564 563 562 561 560 559 558 557 556 555 554 553 552 551 550 549 548 547 546 545 544 543 542 541 540 539 538 537 536 535 534 533 532 531 530 529 528 527 526 525 524 523 522 521 520 519 518 517 516 515 514 513 512 511 510 509 508 507 506 505 504 503 502 501 500 499 498 497 496 495 494 493 492 491 490 489 488 487 486 485 484 483 482 481 480 479 478 477 476 475 474 473 472 471 470 469 468 467 466 465 464 463 462 461 460 459 458 457 456 455 454 453 452 451 450 449 448 447 446 445 444 443 442 441 440 439 438 437 436 435 434 433 432 431 430 429 428 427 426 425 424 423 422 421 420 419 418 417 416 415 414 413 412 411 410 409 408 407 406 405 404 403 402 401 400 399 398 397 396 395 394 393 392 391 390 389 388 387 386 385 384 383 382 381 380 379 378 377 376 375 374 373 372 371 370 369 368 367 366 365 364 363 362 361 360 359 358 357 356 355 354 353 352 351 350 349 348 347 346 345 344 343 342 341 340 339 338 337 336 335 334 333 332 331 330 329 328 327 326 325 324 323 322 321 320 319 318 317 316 315 314 313 312 311 310 309 308 307 306 305 304 303 302 301 300 299 298 297 296 295 294 293 292 291 290 289 288 287 286 285 284 283 282 281 280 279 278 277 276 275 274 273 272 271 270 269 268 267 266 265 264 263 262 261 260 259 258 257 256 255 254 253 252 251 250 249 248 247 246 245 244 243 242 241 240 239 238 237 236 235 234 233 232 231 230 229 228 227 226 225 224 223 222 221 220 219 218 217 216 215 214 213 212 211 210 209 208 207 206 205 204 203 202 201 200 199 198 197 196 195 194 193 192 191 190 189 188 187 186 185 184 183 182 181 180 179 178 177 176 175 174 173 172 171 170 169 168 167 166 165 164 163 162 161 160 159 158 157 156 155 154 153 152 151 150 149 148 147 146 145 144 143 142 141 140 139 138 137 136 135 134 133 132 131 130 129 128 127 126 125 124 123 122 121 120 119 118 117 116 115 114 113 112 111 110 109 108 107 106 105 104 103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

output.txt x
1 500500 ✓

```

```

Дополнительные задачи\input.txt x
1 3
2 1 2 3

output.txt x
1 0

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи		
Пример из задачи	0.140625 sec	17.537109375 KB
Верхняя граница диапазона значений входных данных из	0.109375 sec	96.921875 KB



Вывод по задаче: в задаче я немного изменила алгоритм сортировки слиянием, чтобы посчитать количество инверсий.

### Задача №7. Поиск максимального подмассива за линейное время

Можно найти максимальный подмассив за линейное время, воспользовавшись следующими идеями. Начните с левого конца массива и двигайтесь вправо, отслеживая найденный к данному моменту максимальный подмассив. Зная максимальный подмассив массива  $A[1..j]$ , распространите ответ на поиск максимального подмассива, заканчивающегося индексом  $j + 1$ , воспользовавшись следующим наблюдением: максимальный подмассив массива  $A[1..j + 1]$  представляет собой либо максимальный подмассив массива  $A[1..j]$ , либо подмассив  $A[i..j + 1]$  для некоторого  $1 \leq i \leq j + 1$ . Определите максимальный подмассив вида  $A[i..j + 1]$  за константное время, зная максимальный подмассив, заканчивающийся индексом  $j$ .

В этом случае у вас возможны 2 варианта тестирования: первый предполагает создание рандомного массива чисел, аналогично задаче №1 (в этом случае формат входного и выходного файла смотрите там). Второй вариант - взять любые данные по акциям какой-либо компании, аналогично задаче №6.

```
from time import process_time
from tracemalloc import start, get_traced_memory
```

```
def maxSubArray(arr):
    idx = 0
    idx_len = 0
    summ = 0
    max_sum = 0
    for i in range(len(arr)):
        if summ < 0:
            summ = 0
        if summ == 0:
            idx = i
        summ += arr[i]
        if summ > max_sum:
```

```

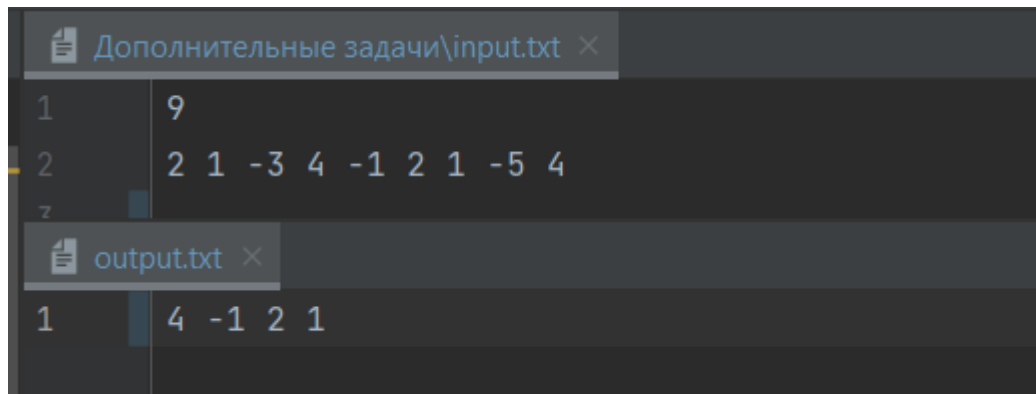
        max_sum = summ
        idx_len = i
        if max_sum == 0:
            idx = -1
            idx_len = -1
        return arr[idx: idx_len + 1]

if __name__ == '__main__':
    start()
    with open('input.txt', 'r') as f:
        n = f.readline()
        a = [int(i) for i in f.readline().split()]
        res = maxSubArray(a)
        with open('output.txt', 'w+') as g:
            g.write(' '.join([str(i) for i in res]))
        print('Time:', str(process_time()), 'sec')
        print('Memory usage:', str(get_traced_memory()[1] / 1024),
'KB')

```

Проходимся по всем элементам списка один раз и смотрим, не стала ли сумма элементов, включая следующий, меньше, чем была (если стала, прекращаем поиск и запоминаем этот подмассив).

Результат работы кода на примерах из текста задачи:



```

Дополнительные задачи\input.txt ×
1 9
2 2 1 -3 4 -1 2 1 -5 4
output.txt ×
1 4 -1 2 1

```

Результат работы кода на максимальных и минимальных значениях:

```

Дополнительные задачи\input.txt
1 1000
2 25 9 11 0 15 27 28 6 8 12 20 20 12 11 16 15 24 19 26 17 20 24
output.txt
1 25 9 11 0 15 27 28 6 8 12 20 20 12 11 16 15 24 19 26 17 20

```

```

Дополнительные задачи\input.txt
1 3
2 1 -1 3
output.txt
1 3

```

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

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.140625 sec	17.560546875 KB
Пример из задачи	0.125 sec	17.244140625 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.109375 sec	92.58984375 KB

Вывод по задаче: в этой задаче я использовала алгоритм нахождения максимального подмассива за линейное время.

### Вывод:

В работе я вспомнила метод “разделяй и властвуй” и некоторые алгоритмы, работающие за логарифмическое время.