

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 вставками, выбором, пузырьковая
Вариант 27

Выполнил:
Филиппов А.Э.
К3139

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3-5
Задача №1. Сортировка вставкой	3-5
Задача №8. Секретарь своп	5-9
Задача №9. Сложение двоичных чисел	9-12
Дополнительные задачи	12
Задача №2. Сортировка вставкой +	12-14
Задача №3. Сортировка вставкой по убыванию	14-16
Задача №4. Линейный поиск	16-18
Задача №5. Сортировка выбором	18-21
Задача №6. Пузырьковая сортировка	21-23
Задача №7. Знакомство с жителями Сортлэнда	23-27
Задача №10. Палиндром	27-30
Вывод	31

Задачи по варианту

Задача №1. Сортировка вставкой

Используя код процедуры Insertion-sort, напишите программу и проверьте сортировку массива $A = \{31, 41, 59, 26, 41, 58\}$.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 103$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 109.
- **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.

Выберите любой набор данных, подходящих по формату, и протестируйте алгоритм.

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()

def insertion_sort():
    for i in range(1, len(a)):
        j = i
        while a[j-1] > a[j] and j != 0:
            a[j-1], a[j] = a[j], a[j-1]
            j -= 1
n = int(lines[0])
a = list(map(int, lines[1].split()))
insertion_sort()
open("output.txt", 'w').write(" ".join(map(str, a)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
      " байт")
tracemalloc.stop()
```

В функции `insertion_sort` описан алгоритм сортировки вставкой для массива `a`. Программа читает ввод из файла, запускает функцию и записывает вывод в файл.

Результаты работы кода:

```
1 6
2 31 41 59 26 41 58

input.txt
26 31 41 41 58 59

NORMAL output.txt
```

```
1 1
2 1

NORMAL input.txt
1 1

output.txt
```

```
0000 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 9
70 969 968 967 966 965 964 963 962 961 960 959 958 957 956 955 954 953 952 951 950 949 948 947 946 945 944 943 942 941 940
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 9
09 908 907 906 905 904 903 902 901 900 899 898 897 896 895 894 893 892 891 890 889 888 887 886 885 884 883 882 881 880 879
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 8
48 847 846 845 844 843 842 841 840 839 838 837 836 835 834 833 832 831 830 829 828 827 826 825 824 823 822 821 820 819 818
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 7
87 786 785 784 783 782 781 780 779 778 777 776 775 774 773 772 771 770 769 768 767 766 765 764 763 762 761 760 759 758 757
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 7
26 725 724 723 722 721 720 719 718 717 716 715 714 713 712 711 710 709 708 707 706 705 704 703 702 701 700 699 698 697 696
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 6
65 664 663 662 661 660 659 658 657 656 655 654 653 652 651 650 649 648 647 646 645 644 643 642 641 640 639 638 637 636 635
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 6
04 603 602 601 600 599 598 597 596 595 594 593 592 591 590 589 588 587 586 585 584 583 582 581 580 579 578 577 576 575 574
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 5
RMAL input.txt text utf-8[unix] 1,001 words 100% ln:2/2=1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 8
5 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 18
0 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 24
1 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 30
2 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332
333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 36
3 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393
394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 42
4 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
tput.txt text utf-8[!EOL][unix] 1,000 words 100% ln:1/1=1
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00025519600058032665	13939
Пример из задачи	0.0004998770000383956	13955
Верхняя граница диапазона значений входных данных из текста задачи	0.2963638310002352	109026

Вывод по задаче: Ввиду $O(n^2)$ сложности алгоритма, сортировка большого количества чисел может занять много времени. Сортировка вставками один из простых алгоритмов, позволяющих легко и быстро решить несложные задачи в области сортировки с маленьким количеством чисел. Однако в условиях больших нагрузок использовать такой алгоритм не рационально.

Задача №8. Секретарь своп

Дан массив, состоящий из n целых чисел. Вам необходимо его отсортировать по неубыванию. Но делать это нужно так же, как это делает мистер Своп — то есть, каждое действие должно быть взаимной перестановкой пары элементов. Вам также придется записать все, что Вы делали, в файл, чтобы мистер Своп смог проверить Вашу работу.

- **Формат входного файла (input.txt).** В первой строке входного файла со держится число n ($3 \leq n \leq 5000$) — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 109. Числа могут совпадать друг с другом.
- **Формат выходного файла (output.txt).** В первых нескольких строках вы ведите осуществленные Вами операции перестановки элементов. Каждая строка должна иметь следующий формат:

Swap elements at indices X and Y.

Здесь X и Y — различные индексы массива, элементы на которых нужно переставить ($1 \leq X, Y \leq n$). Мистер Своя любит порядок, поэтому сделайте так, чтобы $X < Y$.

После того, как все нужные перестановки выведены, выведите следующую фразу:

No more swaps needed.

• Пример:

input.txt	output.txt
5 3 1 4 2 2	Swap elements at indices 1 and 2. Swap elements at indices 2 and 4. Swap elements at indices 3 and 5. No more swaps needed.

5

- Ограничение по времени. 1 сек.
- Ограничение по памяти. 256 мб.

Семья секретаря Своя занималась сортировками массивов, и именно с помощью перестановок пар элементов, как минимум с XII века, поэтому все Свои владеют этим искусством в совершенстве. Мы не просим Вас произвести мини мальную последовательность перестановок, приводящую к правильному ответу. Однако учтите, что для вывода слишком длинной последовательности у Вашего алгоритма может не хватить времени (или памяти — если выводимые строки хранятся в памяти перед выводом). Подумайте, что с этим можно сделать. Решение существует!

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", 'w')

def swap(i, j):
    d = a[i]
    a[i] = a[j]
    a[j] = d
def quick_sort(l, r):
    i = l
    j = r
    pivot = a[(i+j)//2]
    while i <= j:
        while a[i] < pivot:
            i+=1
        while a[j] > pivot:
            j-=1
        if i <= j:
            a[i], a[j] = a[j], a[i]
            out.write(f"Swap elements at indices {i} and {j}. ")
            i += 1
            j -= 1
    if j > l:
        quick_sort(l, j)
    if i < r:
        quick_sort(i, r)

n = int(lines[0])
a = list(map(int, lines[1].split()))
quick_sort(0, n-1)

out.write("No more swaps needed")

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
      " байт")
tracemalloc.stop()
```

Решение работает через алгоритм быстрой сортировки. Его средняя сложность $O(n \cdot \log(n))$

Результат работы кода:

```

Swap elements at indices 2 and 4. Swap elements at indices 0 and 1. Swap elements at indices 1 and 3. Swap elements at indices 2 and 2. No more swaps needed

```

```
5000
@
@
@
@
@
@
@
@
@
@
put.txt                                     text utf-8[unix] 5,001 words 50% ln:1/236
Swap elements at indices 0 and 4999. Swap elements at indices 1 and 4998. Swap elements at indices 2 and 4997. Swap elements at indices 3 and 4996. Swap elements at indices 4 and 4995. Swap elements at indices 5 and 4994. Swap elements at indices 6 and 4993. Swap elements at indices 7 and 4992. Swap elements at indices 8 and 4991. Swap elements at indices 9 and 4990. Swap elements at indices 10 and 4989. Swap elements at indices 11 and 4988. Swap elements at indices 12 and 4987. Swap elements at indices 13 and 4986. Swap elements at indices 14 and 4985. Swap elements at indices 15 and 4984. Swap elements at indices 16 and 4983. Swap elements at indices 17 and 4982. Swap elements at indices 18 and 4981. Swap elements at indices 19 and 4980. Swap elements at indices 20 and 4979. Swap elements at indices 21 and 4978. Swap elements at indices 22 and 4977. Swap elements at indices 23 and 4976. Swap elements at indices 24 and 4975. Swap elements at indices 25 and 4974. Swap elements at indices 26 and 4973. Swap elements at indices 27 and 4972. Swap elements at indices 28 and 4971. Swap elements at indices 29 and 4970. Swap elements at indices 30 and 4969. Swap elements at indices 31 and 4968. Swap elements at indices 32 and 4967. Swap elements at indices 33 and 4966. Swap elements at indices 34 and 4965. Swap elements at indices 35 and 4964. Swap elements at indices 36 and 4963. Swap elements at indices 37 and 4962. Swap elements at indices 38 and 4961. Swap elements at indices 39 and 4960. Swap elements at indices 40 and 4959. Swap elements at indices 41 and 4958. Swap elements at indices 42 and 4957. Swap elements at indices 43 and 4956. Swap elements at indices 44 and 4955. Swap
```

```
Swap elements at indices 1 and 1. No more swaps needed
```


	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00017191200095112436	13876
Пример из задачи	0.00022218999947654083	13880
Верхняя граница диапазона значений входных данных из текста задачи	0.03396921999956248	514884

Вывод по задаче: Алгоритмы сортировки с квадратичной сложностью не эффективны. В вопросе эффективности хорошо себя проявляет алгоритм quicksort.

Задача №9. Сложение двоичных чисел

Рассмотрим задачу сложения двух n -битовых двоичных целых чисел, хранящихся в n -элементных массивах A и B . Сумму этих двух чисел необходимо занести в двоичной форме в $(n + 1)$ -элементный массив C . Напишите скрипт для сложения этих двух чисел.

- **Формат входного файла (input.txt).** В одной строке содержится два n битовых двоичных числа, записанные через пробел ($1 \leq n \leq 103$)
- **Формат выходного файла (output.txt).** Одна строка - двоичное число, которое является суммой двух чисел из входного файла.
- Оцените асимптотическое время выполнения вашего алгоритма.

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", 'w')

n = int(lines[0].rstrip())
a = list(map(int, lines[1].rstrip()))
b = list(map(int, lines[2].rstrip()))
c = [0] * (n+1)
mem = False
for i in range(n-1, -1, -1):
    num = a[i] + b[i]
    if mem:
        num += 1
    mem = False
    match num:
        case 0:
            c[i+1] = 0
        case 1:
            c[i+1] = 1
        case _:
            mem = True
            c[i+1] = 0
if mem:
    c[0] = 1
    out.write("".join(map(str, c)))
else:
    out.write("".join(map(str, c[1:])))

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
      " байт")
tracemalloc.stop()
```

Так как оба числа одинаковой длины n , то задачу можно решить циклом от 0 до n . Сложность алгоритма $O(n)$, однако в худшем случае учитывая вывод сложность составляет $O(2*n)$

Результат работы кода:

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00017971499983104877	13990
Верхняя граница диапазона значений входных данных из текста задачи	0.00040449900188832544	20896

Вывод по задаче: Если побитово складывать числа, то сложение можно выполнить за $O(n)$. Это довольно медленный способ для современных вычислительных систем.

Дополнительные задачи

Задача №2. Сортировка вставкой +

Измените процедуру Insertion-sort для сортировки таким образом, чтобы в выходном файле отображалось в первой строке n чисел, которые обозначают новый индекс элемента массива после обработки.

- **Формат выходного файла (input.txt).** В первой строке выходного файла выведите n чисел. При этом i -ое число равно индексу, на который, в момент обработки его сортировкой вставками, был перемещен i -ый элемент исходного массива. Индексы нумеруются, начиная с единицы. Между любыми двумя числами должен стоять ровно один пробел.

Пример.

input.txt	output.txt
10 1 8 4 2 3 7 5 6 9 0	1 2 2 2 3 5 5 6 9 1 0 1 2 3 4 5 6 7 8 9

В примере сортировка вставками работает следующим образом:

- Первый элемент остается на своем месте, поэтому первое число в ответе — единица. Отсортированная часть массива: [1]
- Второй элемент больше первого, поэтому он тоже остается на своем месте, и второе число в ответе — двойка. [1 8]
- Четверка меньше восьмерки, поэтому занимает второе место. [1 4 8]
- Двойка занимает второе место. [1 2 4 8]
- Тройка занимает третье место. [1 2 3 4 8]
- Семерка занимает пятое место. [1 2 3 4 7 8]
- Пятерка занимает пятое место. [1 2 3 4 5 7 8]
- Шестерка занимает шестое место. [1 2 3 4 5 6 7 8]
- Девятка занимает девятое место. [1 2 3 4 5 6 7 8 9]
- Ноль занимает первое место. [0 1 2 3 4 5 6 7 8 9]

Код:

```
import time

import tracemalloc
lol = "1 "
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
def insertion_sort():
    global lol
    for i in range(1, len(a)):
        j = i
        while a[j-1] > a[j] and j != 0:
            a[j-1], a[j] = a[j], a[j-1]
            j -= 1
        lol += str(j+1) + " "
n = int(lines[0])
a = list(map(int, lines[1].split()))
insertion_sort()
open("output.txt", 'w').write(lol + " ".join(map(str, a)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) + " байт")
tracemalloc.stop()
```

В качестве основы для решения данной задачи используется код из задачи 1. Дополнительно был добавлен счетчик для вывода, удовлетворяющему условию.

Результат работы кода:

```
10
1 8 4 2 3 7 5 6 9 0

out.txt
1 2 2 2 3 5 5 6 9 1 0 1 2 3 4 5 6 7 8 9
```

	Время выполнения (с)	Затраты памяти (байт)
Пример из задачи	0.000388245993235614 15	13960

Вывод по задаче: Простые алгоритмы могут иметь преимущества перед более эффективными, но сложными. Одним из преимуществ является наглядность работы самого алгоритма.

Задача №3. Сортировка вставкой по убыванию

Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедуры Swap. Формат входного и выходного файла и ограничения - как в задаче 1. *Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?*

Код:

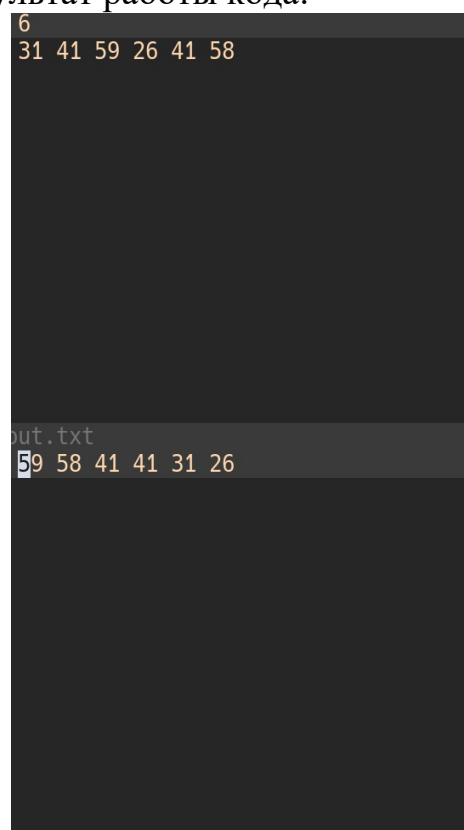
```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
def swap(i, j):
    d = a[i]
    a[i] = a[j]
    a[j] = d

def insertion_sort():
    for i in range(1, len(a)):
        j = i
        while a[j-1] < a[j] and j != 0:
            swap(j-1, j)
            j -= 1
n = int(lines[0])
a = list(map(int, lines[1].split()))
insertion_sort()
open("output.txt", 'w').write(" ".join(map(str, a)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
      " байт")
tracemalloc.stop()
```

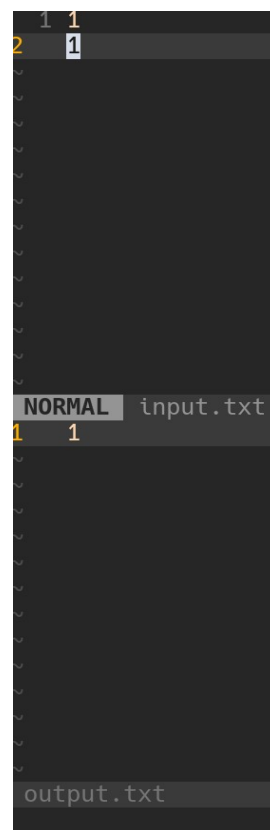
Для того, чтобы поменять порядок достаточно изменить знак с больше на меньше. Была написана функция swap.

Результат работы кода:



```
6
31 41 59 26 41 58

out.txt
59 58 41 41 31 26
```



```
1 1
2 1

NORMAL input.txt
1 1

output.txt
```

```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 8
5 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 18
0 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 24
1 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 30
2 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332
333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 36
3 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393
394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 42
4 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 48
RMAL <ellorok/projects/labs/sem1/1_alg/Дополнительные задачи/3/input.txt text utf-8[unix] | 1,001 words 100% ln :2/236.4
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 9
70 969 968 967 966 965 964 963 962 961 960 959 958 957 956 955 954 953 952 951 950 949 948 947 946 945 944 943 942 941 940
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 9
09 908 907 906 905 904 903 902 901 900 899 898 897 896 895 894 893 892 891 890 889 888 887 886 885 884 883 882 881 880 879
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 8
48 847 846 845 844 843 842 841 840 839 838 837 836 835 834 833 832 831 830 829 828 827 826 825 824 823 822 821 820 819 818
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 7
87 786 785 784 783 782 781 780 779 778 777 776 775 774 773 772 771 770 769 768 767 766 765 764 763 762 761 760 759 758 757
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 7
26 725 724 723 722 721 720 719 718 717 716 715 714 713 712 711 710 709 708 707 706 705 704 703 702 701 700 699 698 697 696
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 6
65 664 663 662 661 660 659 658 657 656 655 654 653 652 651 650 649 648 647 646 645 644 643 642 641 640 639 638 637 636 635
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 6
04 603 602 601 600 599 598 597 596 595 594 593 592 591 590 589 588 587 586 585 584 583 582 581 580 579 578 577 576 575 574

```

	Время выполнения(с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00027802800468634814	13939
Пример из задачи	0.0002642250037752092	13955
Верхняя граница диапазона значений входных данных из текста задачи	0.282430967999997147	109314

Вывод по задаче: Порядок в алгоритме сортировки выборкой можно легко поменять. Иногда удобнее использовать функцию swar.

Задача №4. Линейный поиск

Рассмотрим задачу поиска.

- **Формат входного файла.** Последовательность из n чисел $A = a_1, a_2, \dots, a_n$ в первой строке, числа разделены пробелом, и значение V во второй строке. Ограничения: $0 \leq n \leq 103, -103 \leq a_i, V \leq 103$
- **Формат выходного файла.** Одно число - индекс i , такой, что $V = A[i]$, или значение -1 , если V в отсутствует.
- Напишите код линейного поиска, при работе которого выполняется сканирование последовательности в поисках значения V .

- Если число встречается несколько раз, то выведите, сколько раз встречается число и все индексы i через запятую.
- Дополнительно: попробуйте найти свинью, как в лекции. Используйте во входном файле последовательность слов из лекции, и найдите соответствующий индекс.

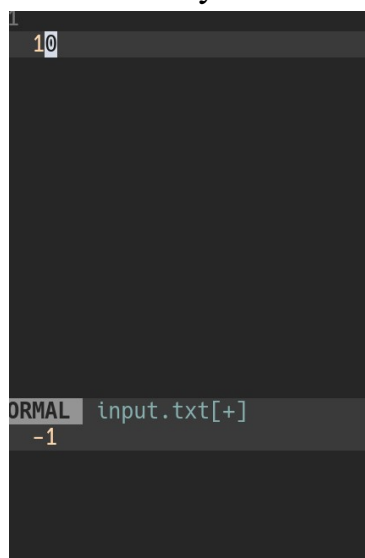
Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
nums = list(map(int, lines[0].split()))
v = int(lines[1])
out = open("output.txt", 'w')
occurrences = { 'count' :0, 'arr': []}

for i in range(len(nums)):
    if v == nums[i]:
        occurrences['count'] += 1
        occurrences['arr'].append(i)
if occurrences['count'] == 0:
    out.write("-1")
elif occurrences['count'] == 1:
    out.write(str(occurrences['arr'][0]))
else:
    out.write(str(occurrences['count']) + '\n')
    out.write(",".join(map(str, occurrences['arr'])))

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
" байт")
tracemalloc.stop()
```

Для решения данной задачи был использован словарь. В словарь записываются данные о том, сколько буква раз встречается в тексте. Далее выводится данные в соответствии с условием



```

10 3 6 4 10 4 10 7 3 4 3 2 3 10 2 7 6 9 2 10 6 1 9 10 5 6 4 10 8 2 2 8 9 7 5 2 5 2 3 6 5 1 1 6 7 8 6 2 8 5 7 9 6 4 9 1 10
1 9 2 5 7 4 4 5 7 7 8 5 9 10 7 6 1 3 1 1 6 9 1 1 4 10 10 1 3 1 9 3 4 2 5 8 9 9 4 9 6 7 10 9 7 4
10
out.txt text utf-8[unix] 104 words 33% ln :1/3
12
0,4,6,13,19,23,27,56,70,82,83,99

```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00013972300075693056	13888
Верхняя граница диапазона значений входных данных из текста задачи	0.000709216998075135	14156

Вывод по задаче: За линейное время можно найти число в несортированном массиве данных. Это может быть полезно

Задача №5. 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.

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()

def swap(i, j):
    d = a[i]
    a[i] = a[j]
    a[j] = d

def selection_sort():
    for i in range(n-1):
        mini = a[i]
        min_index = i
        for j in range(i+1, n):
            if a[j] < mini:
                mini = a[j]
                min_index = j
        swap(i, min_index)

n = int(lines[0])
a = list(map(int, lines[1].split()))
selection_sort()

open("output.txt", 'w').write(" ".join(map(str, a)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
      " байт")
tracemalloc.stop()
```

Сортировка выбором основывается на методе поиска минимального/максимального числа. Так, за n операций поиска можно отсортировать массив или список.

Результат работы кода:

```
312 444 906 876 103 351 181 423 307 693 104 524 818 292 497 395 94 418 835 929 375 27 624 243 453 759 759 621 368 74 236
03 108 776 198 486 617 340 705 71 423 422 188 563 945 583 201 194 848 279 1 56 450 326 103 392 364 703 858 887 829 32 484
7 68 62 390 96 99 986 231 661 157 990 235 889 307 867 43 114 380 333 619 396 855 221 341 32 77 836 414 790 322 521 929 39
64 796 42 743 391 56 626 430 244 983 882 915 706 998 230 286 50 518 295 990 344 89 658 729 528 378 774 249 25 255 464 51
530 19 681 268 962 640 73 60 437 403 347 20 66 819 730 446 155 635 475 771 189 839 741 17 950 675 26 364 743 90 270 167
93 498 919 280 534 559 514 655 871 342 66 9 733 591 17 297 356 870 865 256 933 777 477 469 363 577 422 416 192 726 145 79
368 490 116 977 222 836 648 213 627 929 946 966 34 238 452 638 980 79 564 857 458 825 293 703 730 597 483 586 720 589 24
974 675 961 651 131 568 900 252 73 996 568 991 477 416 920 519 978 860 200 334 932 873 909 898 714 360 2 43 200 600 724
09 267 990 485 601 969 416 34 621 60 774 662 330 896 614 450 616 28 472 525 409 920 894 691 700 90 598 1 353 864 392 7 46
231 543 614 847 9 642 802 762 204 650 839 865 805 317 255 829 47 109 676 790 777 416 67 695 344 12 327 400 233 794 282 9
6 522 544 807 524 964 133 852 230 261 445 16 760 728 632 237 952 128 967 115 81 859 403 666 919 327 213 637 842 839 465 4
603 685 569 66 356 618 272 488 459 596 609 542 559 232 493 228 518 230 252 363 697 978 522 758 954 721 693 445 943 393 8
9 704 477 914 638 883 146 251 597 200 183 349 93 710 721 959 590 381 167 289 504 853 316 706 585 496 357 90 621 393 647 9
8 984 24 638 729 328 745 719 555 30 162 512 58 503 129 841 424 576 236 88 868 2 686 416 858 88 56 664 820 967 336 822 500
665 179 273 600 533 183 994 783 463 93 143 198 838 161 747 136 388 708 555 428 294 149 207 460 614 69 299 315 726 598 612
MAL input.txt text utf-8[unix] 1,001 words 100% ln :2/2=6:1
1 1 2 2 2 2 4 7 7 9 9 12 13 16 16 16 17 17 17 17 19 20 24 25 26 27 28 28 29 29 30 31 32 32 32 32 33 34 34 36 38 39 39 40
1 42 42 43 43 44 46 47 47 49 50 51 56 56 56 58 58 60 60 60 62 64 66 66 66 66 66 67 68 69 71 72 73 73 74 74 77 77 79 81 81
82 87 88 88 88 89 93 93 93 93 91 91 92 93 93 92 98 92 97 926 925 924 923 922 921 920 919 918 917 916 915 914 913 912 911 910
4 115 115 116 120 126 126 127 128 129 131 133 133 135 136 137 138 141 143 145 145 145 146 147 149 152 154 154 155 157 159
161 162 162 166 167 167 167 172 173 175 177 178 179 180 181 183 183 184 185 188 189 189 190 191 192 193 194 197 198 1
8 198 200 200 200 201 202 203 204 204 204 205 207 208 208 209 212 213 213 216 219 221 221 222 222 224 228 228 228 230
230 230 231 231 232 233 233 234 234 234 234 235 236 236 237 237 238 241 241 243 244 244 245 247 248 249 249 251 252 2
2 255 255 256 256 257 258 258 259 261 262 265 266 266 267 267 268 270 271 271 272 273 274 276 277 279 280 282 283 283 284
286 288 289 291 292 292 293 293 293 294 295 295 295 297 299 299 300 303 307 307 307 310 311 312 315 315 316 317 317 318 3
9 320 322 322 322 322 325 326 327 327 327 327 327 327 328 328 330 332 333 333 334 334 334 335 336 336 336 336 338 340 340 341
341 342 342 342 344 344 346 347 347 349 349 349 350 351 353 356 357 357 360 360 360 361 363 363 363 364 364 364 36
6 366 368 368 368 369 369 372 373 374 375 375 376 377 378 379 379 380 380 381 381 386 386 388 388 390 390 391 392 392 393
393 395 396 399 400 400 400 403 403 405 407 407 408 409 409 414 414 416 416 416 416 416 416 416 418 418 422 422 423 423 424 4
4 424 424 425 426 427 428 429 430 430 430 436 437 438 441 444 445 445 445 446 446 450 450 451 452 453 454 456 456 458 458
```

```
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
70 969 968 967 966 965 964 963 962 961 960 959 958 957 956 955 954 953 952 951 950 949 948 947 946 945 944 943 942 941 94
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
09 908 907 906 905 904 903 902 901 900 899 898 897 896 895 894 893 892 891 890 889 888 887 886 885 884 883 882 881 880 87
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
48 847 846 845 844 843 842 841 840 839 838 837 836 835 834 833 832 831 830 829 828 827 826 825 824 823 822 821 820 819 81
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
87 786 785 784 783 782 781 780 779 778 777 776 775 774 773 772 771 770 769 768 767 766 765 764 763 762 761 760 759 758 75
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
26 725 724 723 722 721 720 719 718 717 716 715 714 713 712 711 710 709 708 707 706 705 704 703 702 701 700 699 698 697 69
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
65 664 663 662 661 660 659 658 657 656 655 654 653 652 651 650 649 648 647 646 645 644 643 642 641 640 639 638 637 636 63
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
04 603 602 601 600 599 598 597 596 595 594 593 592 591 590 589 588 587 586 585 584 583 582 581 580 579 578 577 576 575 57
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
MAL input.txt text utf-8[unix] 1,001 words 100% ln :2/2=6:1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 4
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
5 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 1
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 18
0 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 24
1 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 30
2 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332
333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 36
3 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393
394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 4
4 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
```

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 8
5 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 18
0 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 24
1 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 30
2 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332
333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 36
3 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393
394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 42
4 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 48
MAL input.txt text utf-8[unix] 1,001 words 100% ln :2/2=6:1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 8
5 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 11
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 18
0 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 24
1 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 30
2 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332
333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 36
3 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393
394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 42
4 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
```

	Время выполнения (с)	Затраты памяти (байт)
Наилучший случай	0.10575314200104913	109170
Средний случай	0.1094193440003437	109355
Наихудший случай	0.11533586500445381	109170

Вывод по задаче: Сортировка выбором сработала приблизительно в 2.5 раза быстрее чем сортировка вставкой при одинаковых данных. В среднем случае сортировка выбором приблизительно 2 раза быстрее.

Задача №6. Пузырьковая сортировка

Пузырьковая сортировка представляет собой популярный, но не очень эффективный алгоритм сортировки. В его основе лежит многократная перестановка соседних элементов, нарушающих порядок сортировки. Вот псевдокод этой сортировки:

```
Bubble_Sort(A):
  for i = 1 to A.length - 1
    for j = A.length downto i+1
      if A[j] < A[j-1]
        поменять A[j] и A[j-1] местами
```

Напишите код на Python и докажите корректность пузырьковой сортировки. Для доказательства корректности процедуры вам необходимо доказать, что она завершается и что $A'[1] \leq A'[2] \leq \dots \leq A'[n]$, где A' - выход процедуры Bubble_Sort, а n - длина массива A .

Определите время пузырьковой сортировки 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.

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()

def bubble_sort():
    for i in range(n-1):
        for j in range(n-i-1):
            if a[j] > a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
n = int(lines[0])
a = list(map(int, lines[1].split()))
bubble_sort()
open("output.txt", 'w').write(" ".join(map(str, a)))

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
      " байт")
tracemalloc.stop()
```

Пузырьковая сортировка работает. Это показывают тесты, во всех случаях массив отсортирован.

Результат работы кода на примерах:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
5 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
0 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241
1 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302
2 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332
333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363
3 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393
394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424
4 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485
RMAL input.txt text utf-8[unix] 1,001 words 100% ln:2/236:1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
5 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
0 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241
1 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302
2 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332
333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363
3 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393
394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424
4 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
```

```

82 481 480 479 478 477 476 475 474 473 472 471 470 469 468 467 466 465 464 463 462 461 460 459 458 457 456 455 454 453 45
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
21 420 419 418 417 416 415 414 413 412 411 410 409 408 407 406 405 404 403 402 401 400 399 398 397 396 395 394 393 392 39
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
60 359 358 357 356 355 354 353 352 351 350 349 348 347 346 345 344 343 342 341 340 339 338 337 336 335 334 333 332 331 33
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
99 298 297 296 295 294 293 292 291 290 289 288 287 286 285 284 283 282 281 280 279 278 277 276 275 274 273 272 271 270 26
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
38 237 236 235 234 233 232 231 230 229 228 227 226 225 224 223 222 221 220 219 218 217 216 215 214 213 212 211 210 209 20
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
77 176 175 174 173 172 171 170 169 168 167 166 165 164 163 162 161 160 159 158 157 156 155 154 153 152 151 150 149 148 14
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
16 115 114 113 112 111 110 109 108 107 106 105 104 103 102 101 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82
1 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
out.txt      text  utf-8[unix]  1,001 words 100% ln :2/2≡3892
[ 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 4
45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84
5 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 1
9 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 1
0 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210
211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 2
1 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271
272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 3
2 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332
333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 3
3 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393
394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 4
4 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454
639 632 35 815 296 879 562 799 571 789 359 95 964 869 462 714 566 431 127 755 398 378 466 990 551 31 905 72 495 315 992 3
4 726 736 198 592 280 437 295 558 672 659 485 963 951 361 126 627 397 805 483 307 352 919 588 737 880 285 705 352 128 650
386 396 581 394 168 32 663 107 441 205 371 106 687 561 354 241 946 998 108 469 486 558 4 776 651 336 157 698 567 867 675
69 625 556 236 266 36 739 782 12 781 320 342 874 95 437 547 464 47 889 2 304 680 675 993 992 78 54 849 738 788 145 361 67
385 853 19 507 546 538 160 160 316 299 778 399 832 544 49 614 559 380 661 434 460 781 642 772 556 297 863 910 608 161 82
90 738 127 829 570 834 524 172 620 209 984 536 230 360 761 433 48 874 839 421 718 701 659 853 392 590 329 284 432 32 539
184 165 184 388 289 716 877 797 304 335 285 946 118 233 902 412 415 997 749 244 884 14 108 142 645 106 383 787 134 264 47
689 200 427 579 210 209 896 20 545 287 453 915 700 98 71 238 703 159 194 703 673 412 424 543 389 515 279 207 822 881 626
634 239 326 215 543 966 63 223 582 951 344 857 227 1 12 927 447 876 821 674 172 800 782 156 216 732 231 903 365 476 549 1
8 793 616 584 210 537 805 647 195 497 93 702 743 697 150 956 575 678 238 728 804 208 41 629 313 315 659 747 971 240 825 5
0 835 850 44 415 340 291 543 246 408 403 546 835 397 668 118 161 992 196 912 889 805 658 184 87 710 391 971 706 52 565 53
148 784 826 420 209 656 822 305 104 891 161 520 459 10 110 646 257 402 331 920 77 732 49 838 405 128 635 591 381 708 79
39 195 976 714 833 8 756 965 446 205 787 637 298 562 469 213 926 46 602 255 334 326 869 574 468 804 708 367 451 597 554 3
5 644 750 23 366 519 416 731 563 173 681 78 527 851 559 433 633 762 609 239 131 615 547 958 729 927 509 879 162 80 190 50
739 883 707 358 141 530 874 527 405 813 602 28 192 486 744 633 248 580 746 512 302 854 735 605 496 796 160 792 119 344 2
out.txt      text  utf-8[unix]  1,001 words 100% ln :2/2≡3
[ 2 3 4 4 5 6 8 8 9 10 11 12 12 12 12 14 18 19 20 22 23 27 27 27 28 31 32 32 33 35 35 36 36 39 40 40 41 42 44 44 44 45 46
47 48 49 49 50 52 52 54 54 57 60 62 63 63 63 69 71 72 72 73 74 75 77 78 78 79 80 80 86 87 87 90 91 93 94 94 95 95 95 98 9
99 100 100 101 102 103 104 105 106 106 107 108 108 108 109 109 110 114 115 117 118 118 119 123 125 126 126 127 127 128 1
8 128 131 134 134 141 142 142 145 147 148 148 148 149 149 150 150 153 155 155 155 156 156 157 159 160 160 160 160 161 161 161
162 162 163 165 168 168 168 171 172 172 173 176 180 181 181 181 184 184 184 188 188 189 190 190 191 192 193 194 195 195 1
6 196 196 198 200 201 203 205 205 207 207 208 209 209 209 209 210 210 212 213 213 213 215 216 222 222 223 224 225 226 227
229 230 231 231 232 233 236 238 238 238 238 239 239 239 240 240 241 241 241 243 244 246 246 246 248 253 255 256 257 257 2
8 259 261 263 264 264 264 266 266 267 269 269 273 274 276 279 279 280 280 282 283 284 284 285 285 285 285 287 289 289 290
291 292 293 295 295 295 296 297 297 298 298 299 302 303 304 304 304 305 305 307 307 310 311 312 313 315 315 316 316 3
7 320 320 324 325 326 326 327 328 328 329 331 331 331 333 334 335 336 337 339 340 342 344 344 347 349 352 352 353 353 354
355 358 359 360 360 361 361 363 365 365 365 366 367 367 371 373 375 377 378 380 380 381 381 383 385 386 388 388 3
9 390 390 391 392 394 395 395 396 397 397 397 398 398 399 399 402 403 403 403 405 405 406 407 407 408 408 409 412 412 412
412 413 414 414 414 414 415 415 416 416 417 417 418 420 420 421 424 425 425 427 427 428 429 429 431 432 432 433 433 4
4 434 434 434 435 437 437 438 440 441 443 446 447 451 452 452 453 453 454 456 458 459 460 462 462 462 462 462 464 465 466

```

	Время выполнения (с)	Затраты памяти (байт)
Наихудший случай	0.285529688000679	109026
Средний случай	0.22447535500396043	109650
Наилучший случай	0.1394668009961606	109026

Вывод по задаче: Пузырьковая сортировка оказалась немного эффективнее в наихудшем случае. Однако в среднем случае она оказалась немного медленнее

Задача №7. Знакомство с жителями Сортлэнда

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со сво ими подданными. Число жителей в графстве нечетно и составляет n ,

где n может быть достаточно велико, поэтому граф решил ограничиться знакомством с тремя представителями народонаселения: с самым бедным жителем, с жителем, обладающим средним достатком, и с самым богатым жителем.

Согласно традициям Сортлэнда, считается, что житель обладает средним достатком, если при сортировке жителей по сумме денежных сбережений он оказывается ровно посередине. Известно, что каждый житель графства имеет уникальный идентификационный номер, значение которого расположено в границах от единицы до n . Информация о размере денежных накоплений жителей хранится в массиве M таким образом, что сумма денежных накоплений жителя, обладающего идентификационным номером i , содержится в ячейке $M[i]$. Помогите секретарю графа мистеру Свопу вычислить идентификационные номера жителей, которые будут приглашены на встречу с графом.

- **Формат входного файла (input.txt).** Первая строка входного файла содержит число жителей n ($3 \leq n \leq 9999$, n нечетно). Вторая строка содержит описание массива M , состоящее из положительных вещественных чисел, разделенных пробелами. Гарантируется, что все элементы массива M различны, а их значения имеют точность не более двух знаков после запятой и не превышают 10^6 .
- **Формат выходного файла (output.txt).** В выходной файл выведите три целых положительных числа, разделенных пробелами — идентификационные номера беднейшего, среднего и самого богатого жителей Сортлэнда.

4

- Пример:

input.txt	output.txt
5 10.00 8.70 0.01 5.00 3.00	3 4 1

Если отсортировать жителей по их достатку, получится следующий

массив: $[0.01, 3][3.00, 5][5.00, 4][8.70, 2][10.00, 1]$

Здесь каждый житель указан в квадратных скобках, первое число — его достаток, второе число — его идентификационный номер. Таким образом, самый бедный житель имеет номер 3, самый богатый — номер 1, а средний — номер 4.

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

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", 'w')

n = int(lines[0])
a = list(enumerate(map(float, lines[1].split()))))

def quick_sort(l, r):
    i = l
    j = r
    pivot = a[(i+j)//2][1]
    while i <= j:
        while a[i][1] < pivot:
            i+=1
        while a[j][1] > pivot:
            j-=1
        if i <= j:
            a[i], a[j] = a[j], a[i]
            i += 1
            j -= 1
    if j > l:
        quick_sort(l, j)
    if i < r:
        quick_sort(i, r)

quick_sort(0, n-1)

out.write(f"{a[0][0]+1} {a[n//2][0]+1} {a[n-1][0]+1}")

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
      " байт")
tracemalloc.stop()
```

Для решения данной задачи был применен алгоритм быстрой сортировки.
 Для сохранения индексов в списке был использован тип данных tuple.
 Результат работы кода:

```

1 5
1 10.00 8.70 0.01 5.00 3.00

input.txt
1 3 4 1

NORMAL output.txt
    
```

```

3
3.08 4.09 2.34

out.txt
3 1 2

NORMAL output.txt
    
```

```

1666.52 4726.2 3899.87 8023.21 10.89 6799.72 6584.7 5411.01 5202.46 4908.4 564.51 2303.12 1699.88 1736.05 5648.14 1012.69
5985.62 5299.33 7351.37 2294.7 4793.1 6878.35 2297.45 5097.25 8941.26 7805.21 8667.26 2188.18 6496.63 764.04 1451.35 1600.
94 7012.67 13.36 5734.44 2747.89 4363.02 4651.45 7861.42 8718.99 3785.69 6984.81 5070.78 2564.26 2742.79 3757.24 5671.82 6
629.43 5242.58 2009.18 5816.04 8941.18 7585.25 1765.63 9179.33 9698.19 3954.75 2217.38 7333.21 1792.79 6723.48 7688.97 78.
12 4793.49 3520.26 993.11 7271.59 5295.14 1411.01 3175.5 9463.91 8192.51 8000.1 3411.89 460.62 4691.89 7106.75 2277.98 253
7.54 7728.17 4561.08 5787.79 2822.01 8102.92 1064.67 3254.52 4880.8 4797.11 192.5 6080.22 5207.42 3456.93 3974.89 6372.79
1089.16 854.1 2193.0 7392.81 4444.58 2886.62 1514.17 9154.86 4689.43 6658.07 5604.51 791.1 3140.04 7621.41 8223.74 8070.02
3507.1 9689.53 9010.27 2478.31 5174.82 4116.77 8930.71 6245.47 3121.66 4679.81 1170.81 3148.83 4259.27 7150.02 4727.43 73
81.96 9085.06 3342.76 6073.6 2960.85 3505.59 4062.03 2947.36 1528.95 6296.02 3949.24 5126.93 8263.43 5507.0 8745.62 8182.8
2 8121.0 3652.77 2913.6 2512.62 4809.92 3779.85 4390.44 1115.85 73.92 1406.41 4870.61 9812.68 8607.93 8620.1 8308.86 2851.
01 6191.9 2593.59 9412.81 3750.67 7923.98 1054.21 9344.03 9548.54 7298.76 6701.1 5200.91 7883.01 5663.99 2778.26 6053.8 66
9.91 8510.03 204.14 8300.78 8598.03 6506.6 5387.23 8488.27 4898.23 155.33 3384.14 9897.24 7259.56 2209.84 6564.32 8617.45
7981.55 7885.15 8456.17 1967.33 4637.91 3493.24 2782.29 7176.55 7803.81 1836.61 8456.62 9160.71 2159.57 1139.66 5547.48 19
09.87 1503.46 2088.46 1902.73 1621.5 1187.23 630.68 4571.45 8482.48 2861.84 3751.5 8757.79 8874.43 7233.68 6330.81 882.05
9880.91 6927.02 64.39 3196.55 8856.43 4323.55 2394.56 9952.41 9571.94 1320.53 6758.5 2074.28 3390.74 4004.19 6806.9 1094.0
out.txt
2593 666 4165

text utf-8[unix] 10,001 words 100% in :2/2=1
    
```

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.00016564099496463314	13952
Пример из задачи	0.000640764003037475	13964
Верхняя граница диапазона значений входных данных из текста задачи	0.0784491370068281	1888792

Вывод по задаче: С помощью быстрой сортировки можно быстро находить медианные, минимальные и максимальные значения в списке. Расчет будет выполняться медленно в случае сортировки квадратичной сложности

Задача №10. Палиндром

Палиндром - это строка, которая читается одинаково как справа налево, так и слева направо.

На вход программы поступает набор больших латинских букв (не обязательно различных). Разрешается переставлять буквы, а также удалять некоторые буквы. Требуется из данных букв по указанным правилам составить палиндром наибольшей длины, а если таких палиндромов несколько, то выбрать первый из них в алфавитном порядке.

- **Формат входного файла (input.txt).** В первой строке входных данных содержится число n ($1 \leq n \leq 100000$). Во второй строке задается последовательность из n больших латинских букв (буквы записаны без пробелов).
- **Формат выходного файла (output.txt).** В единственной строке выходных данных выдайте искомым палиндром.
- Пример:

input.txt	output.txt
5 AAB	ABA
6 QAZQAZ	AQZZQA
6 ABCDEF	A

- Ограничение по времени. 1сек. •

Ограничение по памяти. 64 мб.

Код:

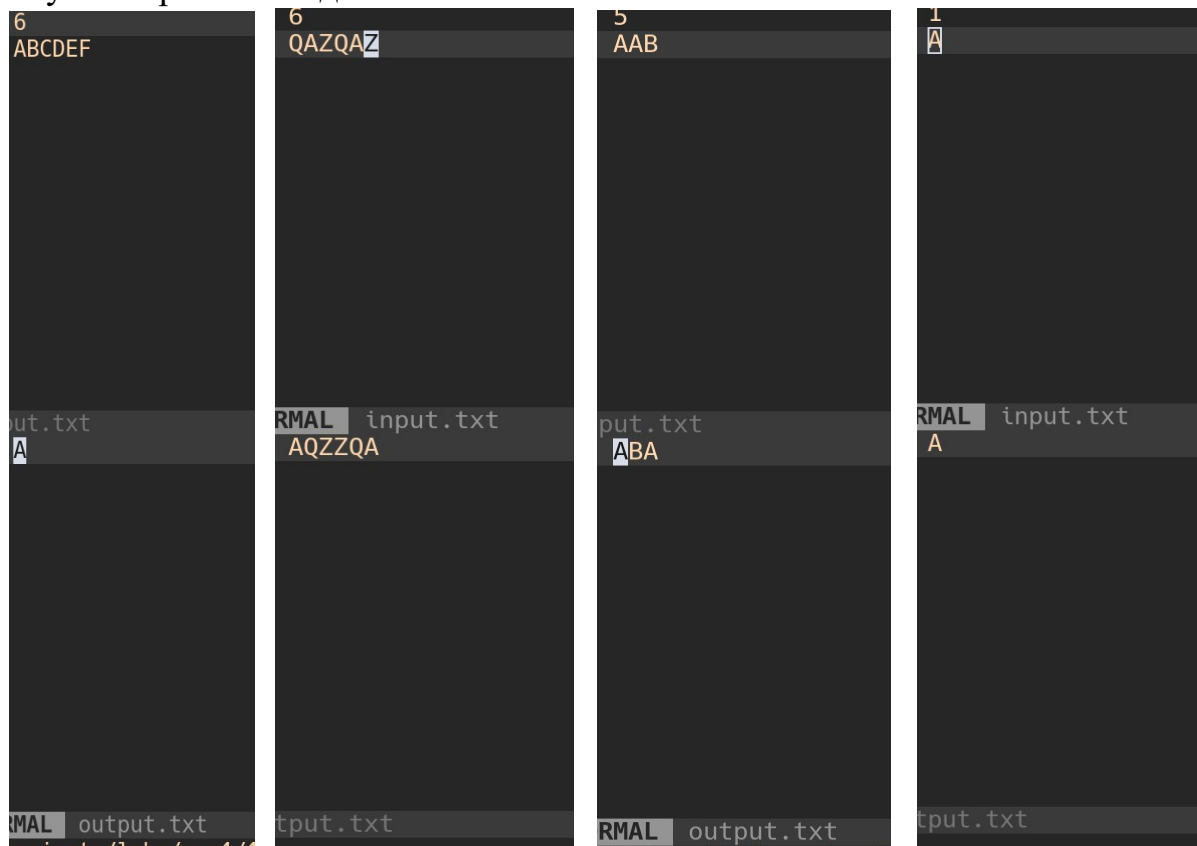
```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
file = open('input.txt')
lines = file.readlines()
out = open("output.txt", 'w')
n = int(lines[0].rstrip())
let = lines[1].rstrip()
occur = {}
for letter in let:
    if letter not in occur:
        occur[letter] = 1
    else:
        occur[letter] += 1
occur = dict(sorted(occur.items()))
nummer = ""
again = ""
for key in occur:
    if occur[key] % 2 == 1 and nummer == "":
        nummer = key
    num = occur[key] // 2
    if num:
        again += key * num
out.write(again + nummer + again[::-1])

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
```

```
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
" байт")
tracemalloc.stop()
```

Для решения данной задачи использовался словарь, где ключ — буква, а значение сколько раз она встречалась. Словарь сортируется по значению ключа в возрастающем порядке. Первая буква встречающаяся нечетное кол-во раз резервируется для центрального положения. Буква добавляется в строку целое число раз от деления на 2. Далее выводится: \$строка + буква + \$зеркальная_строка.

Результат работы кода:



Вывод

Существует множество алгоритмов сортировки данных. В данной лабораторной работе рассматривались главным образом алгоритмы сортировки с квадратичной сложностью. Такие методы сортировки могут быть применены в случае, когда имеет вес простота или скорость разработки, а в случае если приоритет имеет скорость выполнения, то квадратичные алгоритмы сортировки могут стать серьезной проблемой.