

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

Выполнил:

Дегтярь Г.С

К3141

Проверил(а):

Афанасьев А. В.

Санкт-Петербург

2024 г.

# Содержание отчета

## Содержание отчета

<b>Задачи по варианту</b>	<b>2</b>
Задача №1. Сортировка слиянием	
Задача №2. Сортировка слиянием+	6
Задача №9. Метод Штрассена для умножения матриц	17
<b>Дополнительные задачи:</b>	
Задача №3. Число инверсий	17
Задача №4. Бинарный поиск	19
<b>Вывод</b>	<b>21</b>

## Задачи по варианту

### Задача №1. Сортировка слиянием.

Текст задачи:

Используя *псевдокод* процедур Merge и Merge-sort из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = time.perf_counter()

def merge(L, R):
    res = []
    i, j = 0, 0
    n, m = len(L), len(R)
    while i < n and j < m:
        if L[i] < R[j]:
            res.append(L[i])
            i += 1
        else:
            res.append(R[j])
            j += 1
    while j < m:
        res.append(R[j])
        j += 1
    while i < n:
        res.append(L[i])
        i += 1
    return res

def merge_sort(A):
    if len(A) <= 1:
        return A
    q = len(A) // 2
    left = merge_sort(A[:q])
    right = merge_sort(A[q:])
    return merge(left, right)

with open("input.txt", "r") as f:
    n = int(f.readline())
    A1 = f.readline().split()
    A = [int(x) for x in A1]
    res = merge_sort(A)
```

```

with open("output.txt", "w") as f:
    res = [str(x) for x in res]
    s = " ".join(res)
    f.write(s)

end = time.perf_counter()

print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} МВ")
tracemalloc.stop()

```

### Текстовое объяснение решения:

- Сначала разберем принцип работы функции `merge`: ей на вход подаются 2 части массива, которые необходимо сравнить и объединить в 1 массив в правильном порядке. Затем мы инициализируем вспомогательный массив для хранения будущего результата (`res = []`), указатели (`i, j = 0, 0`) и находим длины подмассивов (`n, m = len(L), len(R)`). Затем в цикле `while` пока мы не дойдем до конца хотя бы одного подмассива, мы будем сравнивать их элементы и записывать подходящий в новый массив `res`, после нахождения подходящего элемента, указатель в этом массиве увеличивается на 1. Затем в еще двух циклов `while` мы проверяем прошли ли указатели каждый подмассив полностью и если нет — мы дописываем элементы из недопройденного массива в результирующий массив без проверки т.к каждый подмассив уже отсортирован. Возвращаем `res`. Функцию `merge` вызывает функция `mergt_sort`, давайте разберем ее: Сначала устанавливаем, что условием выхода из рекурсии будет то, что массив, который получает функция будет содержать 1 элемент. Затем находим индекс середины массива через целочисленное деление. Затем рекурсивно вызываем функцию `left = merge_sort(A[:q])`, подавая в аргументе только левую часть массива, а затем сразу же выполняем тот же алгоритм для правой части: `right = merge_sort(A[q:])`. Таким образом после достижения максимальной глубины рекурсии, нам останется лишь объединить все подмассивы которые мы получили: `return merge(left, right)`

## Результат работы кода на примерах задачи:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\6leb\Лаб\АмСД\algorithms-and-data-structures\lab2\task1\merge_sort.py
1000
1758 1725 9457 8738 8248 1345 6998 8572 188 379 5959 8784 8093 8465 5569 5682 8432 670 445 9229 5956 4152 5987 2389 4713 4100 8517 8507 6
Время работы: 0.00493840000000003425 секунд
Пиковая память: 0.27 MB
```

```
1 8 33 34 36 42 43 44 66 73 89 90 107 123 139 162 188 190 205 218 240 242 243 250 250 259 272 293 307 334 335 339 358 379
```



```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\6leb\Лаб\АмСД\algorithms-and-data-structures\lab2\task1\merge_sort.py
1000000
72370 66638 2512 30802 26700 3387 62444 85990 55644 1935 80447 78165 89494 56319 47991 58427 91841 38063 61274 59126 74092 51264 81740 99409
Время работы: 0.0049460000000000339 секунда
Пиковая память: 0.27 MB

Process finished with exit code 0
```

```
117 254 273 383 403 545 812 830 966 1111 1368 1426 1669 1877 1883 1935 2048 2315 2512 2529 2921 3069 3265 3359 3387 3424
```



## Результат работы кода на максимальных и минимальных значениях:

```
1000000
35735 675877 191409 283196 503858 2915 846545 423784 225988 501062 949724 397287 485878 527246 735380 32332 609398 31553
<698396 816302 381165 229523 217122 906132 959563 845461 325282 373594 391258 885657 126217 381912 675512 56209 799999 6
<612666 271737 205087 958199 142127 201758 931274 239138 802184 974093 137130 977931 250573 264845 979624 515242 975012
<543758 501656 103790 382074 373947 711201 613304 17455 54376 742611 601341 243333 178459 626224 730774 472461 55835 691
< 917385 305823 938642 2087 779314 869624 897167 211768 177602 436734 408926 918915 556157 673228 219160 727732 454163 6
<969734 636287 384822 307566 455370 567053 500464 86119 167987 147664 406780 407383 350619 202768 428804 659473 20744 48
<208858 603897 346539 399088 271482 532424 372588 669145 39130 55777 448910 874146 505442 929008 347967 477749 787303 34
<148135 936406 595622 697788 661671 223949 92318 426315 665495 43741 513889 80383 17244 530994 928730 373197 397400 1005
<202506 784676 81524 706536 418773 82765 612456 791888 351675 219106 102602 433511 939537 350283 275979 60632 490730 762
<537707 473155 216404 512489 926913 342467 360194 755732 30014 555719 426898 76163 4989 906218 245135 869824 375124 3464
<591878 370488 851004 732978 188934 958602 836798 165403 486577 865466 511023 257721 240147 523824 185366 292421 343119
<266491 471503 297329 802253 556468 853034 188932 641561 199920 733356 905053 331558 699359 294628 755337 197880 5458 27
14 22 27 28 38 39 44 72 73 83 88 95 97 108 114 138 147 148 149 156 159 159 162 164 175 192 205 213 223 223 233 234 237 2
< 351 353 357 365 374 378 380 388 390 393 393 409 412 415 432 436 439 440 464 478 483 492 492 502 506 518 527 528 529 53
<568 586 600 605 606 607 608 608 614 620 629 633 638 648 650 652 657 666 673 694 697 702 702 702 703 723 724 727 733 734
<839 840 848 872 885 887 898 915 922 929 946 953 957 958 968 971 973 976 981 982 987 994 1012 1015 1020 1020 1022 1025 1
< 1119 1119 1124 1147 1148 1148 1153 1154 1159 1171 1187 1189 1205 1207 1211 1224 1230 1254 1258 1268 1268 1288 1294 130
<1388 1396 1417 1433 1442 1451 1466 1479 1495 1505 1516 1516 1518 1521 1521 1527 1527 1540 1545 1551 1554 1557 1561 1563
<1639 1640 1641 1652 1657 1661 1663 1663 1664 1694 1725 1729 1729 1732 1739 1740 1756 1761 1767 1777 1787 1795 1801 1807
<1899 1902 1906 1914 1917 1921 1922 1928 1938 1945 1946 1952 1956 1966 1968 1980 1981 1987 1989 1991 2003 2020 2046 2046
<2129 2134 2137 2152 2161 2168 2180 2186 2186 2197 2209 2215 2217 2218 2235 2241 2242 2259 2272 2280 2283 2288 2306 2311
<2400 2401 2403 2407 2413 2419 2429 2431 2433 2438 2467 2474 2491 2503 2515 2521 2530 2533 2536 2540 2559 2561 2578 2581
<2679 2682 2693 2697 2697 2707 2718 2719 2723 2740 2742 2745 2748 2750 2765 2770 2795 2808 2815 2820 2821 2823 2831 2836
<2889 2895 2900 2906 2915 2919 2940 2952 2965 2968 2984 2984 3012 3022 3023 3039 3043 3048 3049 3064 3069 3069 3073 3077
<3139 3141 3143 3144 3145 3146 3155 3167 3174 3189 3190 3195 3195 3203 3208 3209 3222 3231 3245 3252 3254 3267 3283 3286
<3387 3392 3396 3399 3406 3423 3424 3427 3432 3450 3452 3455 3464 3475 3481 3483 3484 3501 3505 3505 3513 3523 3524 3525
<3586 3590 3590 3591 3599 3605 3612 3631 3640 3642 3642 3644 3645 3654 3663 3674 3676 3684 3689 3701 3712 3715 3724 3734
```

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\6leb\Лаб\АмСД\algorithms-and-data-structures\lab2\task1\merge_sort.py
1
12
Время работы: 0.0008252999999998067 секунд
Пиковая память: 0.04 MB

Process finished with exit code 0
```

```
12
```



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0008252999999998067 секунд	0.04 MB
Пример 1	0.00027679999999996596 секунд	0.04 MB
Пример 2	0.00026571257242557242 секунд	0.04 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.9125724999999996 секунд	34.92 MB

Вывод о задаче:

Данная задача знакомит нас с относительно эффективным алгоритмом сортировки.

## 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.

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = time.perf_counter()

def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = merge_sort(dict(list(arr.items())[:mid]))
    right_half = merge_sort(dict(list(arr.items())[mid:]))

    return merge(left_half, right_half)

def merge(left, right):
```

```

sorted_array = []
i = j = 0

l1 = min(list(left.keys()))
r1 = max(list(right.keys()))

while i < len(left) and j < len(right):
    if list(left.values())[i] < list(right.values())[j]:
        sorted_array.append([list(left.keys())[i],
list(left.values())[i]])
        i += 1
    else:
        sorted_array.append([list(right.keys())[j],
list(right.values())[j]])
        j += 1

while i < len(left):
    sorted_array.append([list(left.keys())[i],
list(left.values())[i]])
    i += 1

while j < len(right):
    sorted_array.append([list(right.keys())[j],
list(right.values())[j]])
    j += 1

f.write(str(l1) + ' ' + str(r1) + ' ' + str(sorted_array[0][1])
+ ' ' + str(sorted_array[-1][1]))
f.write("\n")
return dict(sorted_array)

with open("input.txt", "r") as f:
    n = int(f.readline())
    arr1 = f.readline().split()
    arr = [int(x) for x in arr1]
    with open("output.txt", "w") as f:
        inp_arr = []
        for i in range(1, len(arr)+1):
            inp_arr.append([i, arr[i-1]])
        dict_arr = dict(inp_arr)
        sorted_arr = merge_sort(dict_arr)
        res = sorted_arr.values()
        res = [str(x) for x in sorted_arr.values()]
        f.write(" ".join(res))

```

Текстовое объяснение решения:

Так как данная задача – модификация предыдущей, я объясню только то, в чем она отличается. Начнем с вызова: После считывания данных из файла input мы создаем буферный массив inp\_arr = [], а затем в цикле for i in range(1, len(arr)+1) заполняем его массивам, состоящими из 2х элементов: индекса (начиная с 1) и значения исходного массива. Далее из получившейся конструкции мы создаем словарь dict\_arr, в котором теперь все значения исходного массива будут пронумерованы. Затем мы вызываем функцию merge\_sort(), которая и выполняет все вычисления, а затем записываем данный в файл output. Рассмотрим сначала функцию merge т.к

она вызывается функцией `merge_sort()`. Находим сначала минимальный ключ в словаре `l1 = min(list(left.keys()))`, а затем максимальный `r1 = max(list(right.keys()))`, она нам понадобятся для записи границ слияния. Затем аналогично прошлой задаче мы проверяем все значения подмассивов, однако в данной задаче, в результирующий массив мы записываем не только подходящие значения, но и их номера. Затем после того, как мы отсортировали подмассивы, записываем полученные результаты в файл `output` и выводим отсортированный массив, как словарь. Функция `merge_sort` аналогична функции в прошлой задаче, с поправкой на использование словарей.

Результат работы кода на примерах задачи:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\Gleb\Лабы\АиСД\
100
932306 121287 461523 700722 176829 986816 72844 817699 224364 507644 65196 10789
Время работы: 0.0052045999999999893 секунд
Пиковая память: 0.11 MB
Process finished with exit code 0
```

```
1 | 3 121287 461523
2 | 1 121287 932306
3 | 5 6 176829 986816
4 | 4 6 176829 986816
5 | 1 6 121287 986816
6 | 8 9 224364 817699
7 | 7 9 72844 817699
8 | 11 12 65196 107890
9 | 10 12 65196 507644
10 | 7 12 65196 817699
11 | 1 12 65196 986816
12 | 14 15 490411 906103
13 | 13 15 490411 906103
14 | 17 18 71215 258976
15 | 16 18 71215 930389
16 | 13 18 71215 930389
17 | 20 21 73053 978545
18 | 19 21 73053 978545
19 | 22 23 48021 241176
```

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\Gleb\Лабы\АиСД\
10
194280 868920 854595 458118 233766 888612 465673 152263 61821 744877
Время работы: 0.00092970000000003663 секунд
Пиковая память: 0.05 MB
```

```
1 | 2 194280 868920
4 | 5 233766 458118
3 | 5 233766 854595
1 | 5 194280 868920
6 | 7 465673 888612
9 | 10 61821 744877
8 | 10 61821 744877
6 | 10 61821 888612
1 | 10 61821 888612
61821 152263 194280 233766 458118 465673 744877 854595 868920 888612
```



Результат работы кода на максимальных и минимальных значениях:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\Gleb\Лабы\АиСД\
1
5
Время работы: 0.0009553000000002143 секунд
Пиковая память: 0.04 MB

C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\Gleb\Лабы\АиСД\
100000
563309 178701 361638 880127 92738 20086 112921 663835 767092 861940 854801 393790
Время работы: 1.7334436000000001 секунд
Пиковая память: 4.19 MB

Process finished with exit code 0

2 178701 563309
3 361638 880127
1 4 178701 880127
5 6 20086 92738
7 8 112921 663835
9 10 767092 861940
11 12 393790 854801
13 14 356456 366492
15 16 582183 691063
17 18 249984 287280
19 20 249984 507901
21 22 301745 327683
23 24 286471 563002
25 26 286471 653896
27 28 286471 653896
29 30 249984 653896
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005006999999999096 секунд	0.04 MB
Пример 1	0.005204599999999893 секунд	0.11 MB
Пример 2	0.0009297000000003663 секунд	0.05 MB
Верхняя граница диапазона значений входных данных из текста задачи	1.7544864999999996 секунд	4.19 MB

Вывод о задаче:

Данная задача показывает, что устоявшиеся алгоритмы могут быть модифицированы для достижения новых целей.

### Задача № 3. Число инверсий

#### Текст задачи:

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

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

#### Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

with open("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = time.perf_counter()

def merge_and_count(arr, left, mid, right):
    left_part = arr[left:mid + 1]
    right_part = arr[mid + 1:right + 1]
    i = j = 0
    inversions = 0
    temp = []
    while i < len(left_part) and j < len(right_part):
        if left_part[i] <= right_part[j]:
            temp.append(left_part[i])
            i += 1
        else:
            temp.append(right_part[j])
            j += 1
            inversions += len(left_part) - i

    while i < len(left_part):
        temp.append(left_part[i])
        i += 1
    while j < len(right_part):
        temp.append(right_part[j])
        j += 1
    for i, val in enumerate(temp):
        arr[left + i] = val

    return inversions

def merge_sort_and_count(arr, left, right):
    inversions = 0
    if left < right:
        mid = (left + right) // 2
        inversions += merge_sort_and_count(arr, left, mid)
        inversions += merge_sort_and_count(arr, mid + 1, right)
        inversions += merge_and_count(arr, left, mid, right)
    return inversions

with open("input.txt", "r") as f:
```

```

n = int(f.readline())
arr = list(map(int, f.readline().split()))

inversions = merge_sort_and_count(arr, 0, n - 1)

with open("output.txt", "w") as f:
    f.write(str(inversions) + "\n")

```

Текстовое объяснение решения:

Данная задача так же является модификацией первой, соответственно ее принципы работы будут в целом похожи. Рассмотрим функцию `merge()`. Она работает подхоже на предыдущие варианты, однако в данной задаче в случае если элемент из левой части в сравнении с элементом из правой части меньше, то мы увеличиваем число инверсий на `inversions += len(left_part) - i` т.к все предыдущие элементы левой части тоже будут создавать инверсии (мы сортируем массив слева направо). Также отличием будет цикл `for i, val in enumerate(temp)`, который переопределяет значения исходного массива. Также давайте разберем функцию

Результат работы кода на примерах из текста задачи:

```

5
5 4 3 2 1
Время работы: 0.0008539999999999104 секунд
Пиковая память: 0.04 MB

```

10



```

C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe "C:\Gleb\Лабы\АиС
10
1 8 2 1 4 7 3 2 3 6
Время работы: 0.000646400000000008241 секунд
Пиковая память: 0.04 MB

```

17

C:\Gleb\Лабы\

Результаты работы алгоритма на максимальных и минимальных значениях:

```
673918 331214 505491 818243 628992 529842 25867 815398 176660 322757 89044 809543
121023 823684 878637 391539 827893 658284 195612 481561 845073 602584 928675 46457
304579 699523 406908 451385 403343 279682 163868 612678 786472 289354 227786 8938
567156 547063 7698 330896 61110 438908 880509 798099 830438 766250 974917 832919 2
809092 729293 361895 862775 237118 842496 592281 661293 638176 952363 579516 81691
868854 638989 56414 992967 434115 839217 930153 192446 192487 901951 443105 978589
350324 757648 893885 914998 26956 466145 570260 674613 881857 429094 545717 492313
680628 411449 882368 306456 936403 799012 591418 160062 692897 278339 745828 4741
964225 254665 821618 811047 795402 710221 508968 608195 781004 601648 962806 72231
405289 744198 189468 354287 809982 649113 771909 152894 181522 404922 289873 78786
Время работы: 1.9093571999999996 секунд
Пиковая память: 15.46 MB
```

```
2494548291
```

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.
1
1
Время работы: 0.00040579999999978966 секунд
Пиковая память: 0.04 MB
```

```
0
C:\Gleb\Лабы\АиС
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00040579999999978966 секунд	0.04 MB
Пример 1	0.0007782999999994544 секунд	0.04 MB
Пример 2	0.0006464000000008241 секунд	0.04 MB
Верхняя граница диапазона значений	1.9093571999999996 секунд	15.46 MB

Вывод о задаче:

Данная задача учит нас нестандартно подходить к уже привычным алгоритмам и методам для достижения новых целей.

## 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.

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n = input()
    arr = input().split()
    m = input()
    brr = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(arr))
    f.write("\n")
    f.write(m)
    f.write("\n")
    f.write(" ".join(brr))

def merge_sort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = merge_sort(dict(list(arr.items())[:mid]))
    right_half = merge_sort(dict(list(arr.items())[mid:]))

    return merge(left_half, right_half)

def merge(left, right):
    sorted_array = []
    i = j = 0
    while i < len(left) and j < len(right):
        if list(left.values())[i] < list(right.values())[j]:
            sorted_array.append([list(left.keys())[i],
list(left.values())[i]])
            i += 1
        else:
            sorted_array.append([list(right.keys())[j],
list(right.values())[j]])
            j += 1

    while i < len(left):
        sorted_array.append([list(left.keys())[i],
list(left.values())[i]])
```

```

        i += 1

    while j < len(right):
        sorted_array.append([list(right.keys())[j],
list(right.values())[j]])
        j += 1

    return dict(sorted_array)

def binary_search(arr_dict_sorted, n, b):
    if n == 0:
        return -1
    mid = n // 2
    if list(arr_dict_sorted.values())[mid] == b:
        return list(arr_dict_sorted.keys())[mid]
    if list(arr_dict_sorted.values())[mid] > b:
        res =
        binary_search(dict(list(arr_dict_sorted.items())[:mid]),
len(dict(list(arr_dict_sorted.items())[:mid])), b)
    if list(arr_dict_sorted.values())[mid] < b:
        res = binary_search(dict(list(arr_dict_sorted.items())[mid
+ 1:]), len(dict(list(arr_dict_sorted.items())[mid + 1:])), b)
    return res

with open("input.txt", "r") as f:
    n = int(f.readline())
    a1 = f.readline().split()
    arr = [int(x) for x in a1]
    m = int(f.readline())
    b1 = f.readline().split()
    brr = [int(x) for x in b1]
    arr_for_dict = []
    for i in range(len(arr)):
        arr_for_dict.append([i, arr[i]])
    arr_dict = dict(arr_for_dict)
    arr_dict_sorted = merge_sort(arr_dict)
    res = []
    for b in brr:
        res.append(binary_search(arr_dict_sorted, n, b))
        if res[-1] == None:
            res[-1] = -1
    print(res)

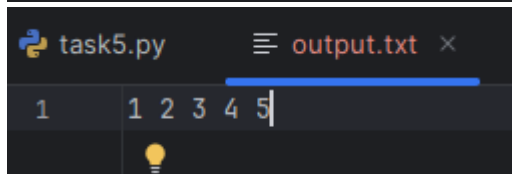
```

Текстовое объяснение решения:

Алгоритм сортировки выбором похож на алгоритм сортировки вставкой тем, что у нас так же есть отсортированная и неотсортированная части. Однако при данном методе мы заменяем число минимальным (либо максимальным) числом из неотсортированной части.

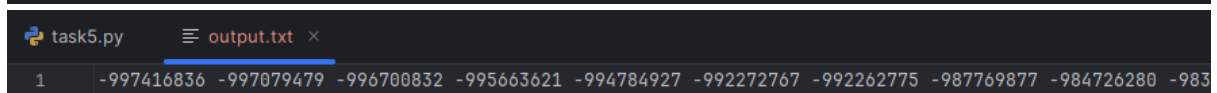
Результат работы кода на примерах из текста задачи:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\p
5
5 4 3 2 1
Время работы: 0.00019339999999967716 секунд
Пиковая память: 0.04 MB
```



Результат работы кода на максимальных и минимальных значениях:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\Gleb\Лабы\АисД\
1000
-356784546 224086878 631790692 583121661 -241719070 -745394775 345998592 -646450
Время работы: 0.110852600000000119 секунд
Пиковая память: 0.17 MB
```



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00018129999999993984 секунд	0.04 MB
Пример 1	0. 0.00027759999999995449	0.04 MB
Пример 2		
Верхняя граница диапазона значений входных данных из текста задачи	0.110852600000000119 секунд	0.17 MB

Вывод по задаче: Данная задача знакомит нас с методом сортировки выбором и учит использовать его.

## Задание № 6. Пузырьковая сортировка.

Текст задачи:

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.

Листинг кода:

```
import time  
import tracemalloc  
  
tracemalloc.start()  
  
with open ("input.txt", "w") as f:  
    n = input()  
    a = input().split()  
    f.write(n)  
    f.write("\n")  
    f.write(" ".join(a))  
  
start = time.perf_counter()  
  
with open ("input.txt", "r") as f:  
    n = int(f.readline())  
    a = f.readline().split()  
    a = [int(x) for x in a]
```



```

for i in range(0, n, 1):
    for j in range(0, n - 1, 1):
        if a[j] > a[j + 1]:
            a[j], a[j + 1] = a[j + 1], a[j]

end = time.perf_counter()

for q in range(n - 1):
    if a[q + 1] < a[q]:
        print("error: invalid sort")

with open ("output.txt", "w") as f:
    a = [str(x) for x in a]
    f.write(" ".join(a))

print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} MB")
tracemalloc.stop()

```

Текстовое объяснение решения:

Данная сортировка крайне неэффективна. В ней сортировка массива происходит за счет постоянной замены каждого числа, которое больше своего соседа справа (при сортировке по возрастанию) с этим самым соседом. С изменением шага цикла, мы бы так добрались до конца массива, постоянно меняя числа местами, а затем начали то же самое сначала  $n$  раз.

Результат работы кода на примерах задачи:

```

C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\6Leb\Лабы\АиСД\algorithms-and-da
1
10000
Время работы: 0.000421599999999996889 секунд
Пиковая память: 0.04 MB

```

```

task5.py  output.txt x
1 10000

```

```

C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe
5
5 4 3 2 1
Время работы: 0.000224699999999999666 секунд
Пиковая память: 0.04 MB

```

```

task5.py  output.txt x
1 1 2 3 4 5

```

Результаты работы кода на max:

```

C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe
1000
-356784546 224086878 631790692 583121661 -241719070 -745394775
Время работы: 0.1063608000000000092 секунд
Пиковая память: 0.17 MB

```

```
task5.py  output.txt x
1 -997416836 C:\Gleb\Лабы\АиСД\algorithms-and-data-structures\lab1\task5\output.txt 2775 -987769877 -984726280 -983578014 -982911285
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004215999999996889 секунд	0.04 MB
Пример из задачи	0.000493099999999913 секунд	0,04 MB
Пример из задачи		
Верхняя граница диапазона значений входных данных из текста задачи	0.106360800000000092 секунд	0.17 MB

Вывод по задаче:

Данная задача знакомит нас с пузырьковым методом сортировки данных и учит использовать его.

## Задача № 7. Знакомство с жителями Сортлэнда.

Текст задачи:

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со своими 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.

Листинг кода:

```
from time import *
import tracemalloc
```

```

tracemalloc.start()

with open ("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = perf_counter()

with open ("input.txt", "r") as f:
    n = int(f.readline())
    a = f.readline().split()
    a = [float(x) for x in a]
    b = a.copy()
    for i in range(1, n, 1):
        for j in range(0, i, 1):
            if a[i] < a[i - 1]:
                if a[j] > a[i]:
                    p = a[j]
                    a[j] = a[i]
                    a[i] = p

end = perf_counter()

for q in range(n - 1):
    if a[q + 1] < a[q]:
        print("error: invalid sort")

with open ("output.txt", "w") as f:
    f.write(str(b.index(a[0]) + 1) + " " + str(b.index(a[n // 2]) + 1) +
" " + str(b.index(a[-1]) + 1))
print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} МВ")
tracemalloc.stop()

```

Текстовое объяснение решения:

В данной задаче нам надо вывести самое маленькое, среднее и самое большое числа массива. Для сортировки я использую алгоритм сортировки вставкой из 1го задания. Так как в задаче необходимо вывести номера этих людей (чисел) относительно изначального (неотсортированного) списка, мы скопируем изначальный список в переменную b и будем выводить индексы относительно нее. Также раз нас просят вывести не индексы, а номера самого бедного, среднего и богатого человека, найдя индекс нужного числа, мы будем увеличивать его на 1, получая тем самым номер нужного человека.

Результат работы кода на примерах из текста задачи:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\61eb\Лабы\
3
2 4 5
Время работы: 0.00020549999999985857 секунд
Пиковая память: 0.04 MB
```

```
task7.py output.txt x
1 1 2 3
```

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\61eb\Лабы\
6
3 2 7 3 0 5
Время работы: 0.00046429999999997307 секунд
Пиковая память: 0.04 MB
```

```
task7.py output.txt x
1 5 1 3
```

Результат работы кода на максимальных и минимальных значениях:

Max:

```
C:\Users\tb\AppData\Local\Programs\Python\Python39\python.exe C:\61eb\Лабы\
9999
29236876 48336239 58543273 96836410 65420857 93255310 78291475 56514094 212
Время работы: 23.1053507000000002 секунд
Пиковая память: 1.37 MB
```

```
task7.py output.txt x
1 102 1357 1074
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004224999999999923 секунд	0.04 MB
Пример из задачи	0.00035030000000000256 секунд	0.04 MB
Пример из задачи		

Верхняя граница диапазона значений входных данных из текста задачи	23.105350700000002 секунд	1.37 MB
--	---------------------------	---------

Вывод о задаче: Данная задача хорошо закрепляет основные методы работы с сортировками в python и учит применять их на практике.

## Задание № 8. Секретарь Своя:

Текст задачи:

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

Листинг кода:

```
from time import *
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n = input()
    a = input().split()
    f.write(n)
    f.write("\n")
    f.write(" ".join(a))

start = perf_counter()

with open ("input.txt", "r") as f:
    with open ("output.txt", "w") as g:
        n = int(f.readline())
        b = f.readline().split()
        a = [int(x) for x in b]
        for i in range( n - 1):
            m = i
            for j in range(i + 1, n):
                if a[m] > a[j]:
                    m = j
            if m != i:
                p = min(m,i) + 1
                v = max(m,i) + 1
                g.write("Swap elements at indices " + str(p) + " and "
+ str(v) + ".")
                g.write("\n")
                a[m], a[i] = a[i], a[m]
            g.write("No more swaps needed.")

end = perf_counter()
```

```

for q in range(n - 1):
    if a[q + 1] < a[q]:
        print("error: invalid sort")

print("Время работы: ", end = start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} MB")
tracemalloc.stop()

```

Текстовое объяснение:

Данный алгоритм, подобно пузырьковой сортировке меняет соседние элементы массива, если они находятся в неправильном порядке, однако лишь до момента, когда они не окажутся на своей правильной позиции.

Результат работы:

```

C:\Users\tb\AppData\Local\Programs\Python\Python39\python
5
3 1 4 2 2
Время работы: 0.0005150000000000432 секунд
Пиковая память: 0.05 MB

```

```

task7.py task8.py output.txt x
1 Swap elements at indices 1 and 2.
2 Swap elements at indices 2 and 4.
3 Swap elements at indices 3 and 5.
4 No more swaps needed.

```

На максимальных значениях:

```

C:\Users\tb\AppData\Local\Programs\Python\Python39\py
5000
29236876 48336239 58543273 96836410 65420857 93255316
Время работы: 2.9171544000000001 секунд
Пиковая память: 0.84 MB

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005543999999999549 секунд	0.05 MB
Пример из задачи	0.00062609999999998518 секунд	0.05 MB

Пример из задачи		
Верхняя граница диапазона значений входных данных из текста задачи	2.9171544000000001 секунд	0.84 МВ

Вывод о задаче:

Данная задача очень интересна тем, что для ее решения необходимо пойти дальше стандартного алгоритма сортировки. Подобный подход развивает адаптивные способности программиста.

## Дополнительные Задачи:

### Задача №3.1. Сортировка вставкой через рекурсию.

Текст задачи:

*Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?*

Листинг задачи:

```
import time
from time import perf_counter
import tracemalloc

tracemalloc.start()

with open ("input.txt", "w") as f:
    n1 = input()
    a1 = input().split()
    f.write(n1)
    f.write("\n")
    f.write(" ".join(a1))

start = time.perf_counter()

with open ("input.txt", "r") as f:
    n = int(f.readline())
    arr = f.readline().split()
    arr = [int(x) for x in arr]

    def recursive_insertion_sort(n, arr):
        if n == 1:
            return arr

        arr = recursive_insertion_sort(n - 1, arr)
        key = arr[n - 1]
        j = n - 2

        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
```

```

        arr[j + 1] = key
        return arr

    a = recursive_insertion_sort(n, arr)
    for q in range(n - 1):
        if a[q + 1] < a[q]:
            print("error: invalid sort")
    a = [str(x) for x in a]

end = perf_counter()

with open ("output.txt", "w") as f:
    f.write(" ".join(a))

print("Время работы: ", end - start, "секунд")
current, peak = tracemalloc.get_traced_memory()
print(f"Пиковая память: {peak / 2**20:.2f} MB")
tracemalloc.stop()

```

Текстовое объяснение:

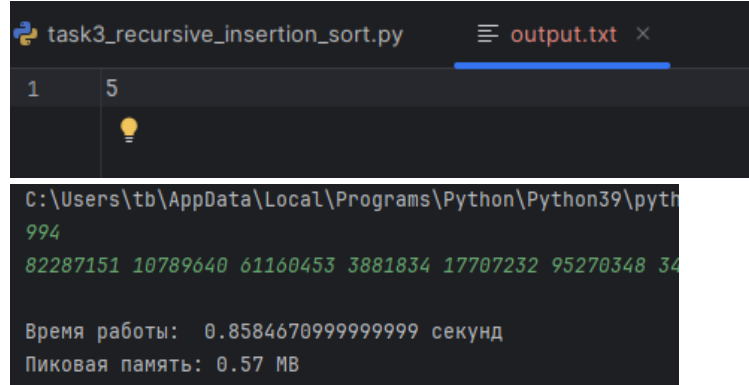
Это рекурсивный вариант сортировки вставкой. В функцию подаем два значения: кол-во элементов в массиве, который надо отсортировать и сам массив. Закручиваем нашу рекурсию повторным вызовом с новым значением  $n = n - 1$ . Благодаря данному действию после того, как  $n$  достигнет значения 1, функции начнут вызывать друг друга в обратном порядке и мы получим поочередный вызов функций со значениями  $n = 1, 2, 3 \dots 7$  и новыми массивами. Затем мы пробегаемся по самому массиву и сортируем его по принципу сортировки вставкой.

Примеры работы программы на максимальных и минимальных числах:

```

C:\Users\tb\AppData\Local\Programs\Python\Python39\py
1
5
Время работы: 0.0003883000000000081 секунд
Пиковая память: 0.04 MB

```



```

task3_recursive_insertion_sort.py  output.txt x
1  5
82287151 10789640 61160453 3881834 17707232 95270348 34
Время работы: 0.8584670999999999 секунд
Пиковая память: 0.57 MB

```



```
task3_recursive_insertion_sort.py  output.txt x
1 170820 207476 282078 373781 604227 605693 617620 778652 858140 920123 967002 1032380 1128074 1239659 1241246 1241481 130605
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00038830000000000081 секунд	0.04 МВ
Пример из задачи	0.0003234000000000008473 секунд	0.04 МВ
Пример из задачи		
Верхняя граница диапазона значений входных данных из текста задачи	0.8584670999999999 секунд	0.57 МВ

#### Вывод о задаче:

Данная задача наглядно показывает принципы работы рекурсивных функций и хорошо помогает разобраться в них.

#### Вывод о лабораторной работе:

Данная лабораторная работа направлена на изучение классических методов сортировки данных в python, тестирования программы, рекурсивных алгоритмов. Данная работа помогла мне освоить перечисленные навыки и вспомнить основные методы работы с языком программирования python.