

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

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Сортировка слиянием. Метод декомпозиции
Вариант 20

Выполнила:
Толстухина Ксения Александровна
К3139

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка слиянием.	3
Задача №5. Представитель большинства	7
Задача №7. Поиск максимально подмаслила за линейное время	10
Дополнительные задачи	14
Задача №4. Бинарный поиск	14
Задача №3. Число инверсий	17
Задача №8. Умножение многочленов	21
Вывод	23

Задачи по варианту

Задача №1. Сортировка слиянием.

Текст задачи.

Напишите программу сортировки слиянием и проверьте ее, создав несколько случайных массивов.

Листинг кода.

```
def merge_sort(nums):  
    if len(nums) > 1:  
        mid = len(nums) // 2  
        left = nums[:mid]  
        right = nums[mid:]  
        merge_sort(left)  
        merge_sort(right)  
  
        i = j = k = 0  
        while i < len(left) and j < len(right):  
            if left[i] < right[j]:  
                nums[k] = left[i]  
                i += 1  
            else:  
                nums[k] = right[j]  
                j += 1  
            k += 1  
  
        while i < len(left):  
            nums[k] = left[i]  
            i += 1  
            k += 1  
        while j < len(right):  
            nums[k] = right[j]
```

```

        j += 1
        k += 1

    return nums

with open('output.txt', 'w') as f:
    file = open('input.txt')
    n = int(file.readline())
    nums = list(map(int, file.readline().split()))
    f.write(' '.join(map(str, merge_sort(nums))))

```

Текстовое объяснение решения.

Для начала я написала функцию - рекурсивный алгоритм сортировки вставками, для его реализации я делю список пополам, таким образом создаю два списка `right` и `left`. Затем я применяю этот же алгоритм для деления списков `right` и `left`, до тех пор, пока не получу список, длина которого равна 1. Дальше происходит слияние списков. Я ввожу три индекса, для правого, левого и изначального. После я перебираю элементы и правого, и левого списка, сравнивая каждый элемент между собой. Если число из списка `left` меньше, чем число из списка `right`, я вставляю его на позицию `k` и увеличиваю индексы левого и текущего массива, в обратном случае в итоговый список отправляется число из правого и увеличиваются нужные индексы на 1. Этот алгоритм занесен в цикл `while` и будет работать до тех пор, пока в любом из списков не кончатся элементы. На случай, если в левом или правом списке еще остались элементы, я прописываю еще два цикла `while`, в которых просто добавляю все оставшееся в нужный список. После этого я читаю данные из файла и записываю в результирующий файл результат работы функции.

Для проверки я генерирую рандомные списки.

Результат работы кода на примерах из текста задачи:

```
input.txt x      output.txt x
1 12565 ✓        1 15 167 223 324 376 391 542 550 586 596, ✓
2 783747 554672 639792 625107 868760 ✓    663 811 921 926 972 977 988 997 1040 ✓
  765368 451362 921828 134099 557592,    1130 1164 1191 1222 1422 1438 1469 ✓
  844254 942025 734168 220661 ✓    1557 1676 1729 1757 1841 2112 2144 ✓
  350241 72971 841010 284873 900834 ✓    2209 2369 2475 2500 2538 2675 2726 ✓
  490128 805438 127498 802431 589607,    2796 2800 2805 2835 2838 2840 2942 ✓
  916171 541239 772191 208439 ✓    2949 2987 3006 3115 3119 3127 3152 ✓
  842940 109297 175150 571501 147928,    3234 3389 3434 3453 3507 3597 3638 ✓
  860667 20665 713252 572050 74475 ✓    3671 3754 3873 3893 4204 4488 4522 ✓
  630766 227520 572347 64585 963753 ✓    4553 4564 4592 4595 4715 4852 5001 ✓
  488971 485701 688121 786417 485357,    5018 5024 5062 5087 5089 5102 5125 ✓
  680712 399771 404414 72810 477358,    5206 5327 5353 5457 5470 5558 5630 ✓
  37588 510215 833627 999986 49670 ✓    5695 5754 5783 6132 6144 6570 6576 ✓
  284981 267708 758108 630125 226980,    6628 6774 6782 6811 6978 7260 7291 ✓
  306752 376 254741 308502 288206 ✓    7394 7625 7628 7751 7752 7977 8095 ✓
  426016 11224 474900 401719 104457 ✓    8159 8245 8266 8306 8339 8354 8376 ✓
  592926 234984 701900 68701 320160 ✓    8434 8524 8557 8559 8628 8693 9106 ✓
  116324 408505 791634 107249 754712,    9350 9384 9402 9407 9439 9511 9583 ✓
  952326 618599 608121 319157 ✓    9688 9743 9832 9834 9875 9877 9886 ✓
  352290 398496 615372 862717 294929,    9922 10036 10286 10344 10663 10744 ✓
  998068 549223 656107 88506 731976,    10759 10782 10792 10810 10902 10953 ✓
  810262 747030 372136 241538 ✓    10995 11089 11095 11200 11224 11229 ✓
  466091 330834 812791 673482 660992,    11332 11355 11454 11632 11638 11673 ✓
  497438 722274 954964 988493 ✓    11491 11857 12016 12057 12106 12128 ✓

Run test 1 x
/usr/local/bin/python3.12 /Users/kseniatoIstuhina/PycharmProjects/Algo_labs/lab2
/task1/test 1.py
0.14203095802804455c
1.320887565612793 MB
Process finished with exit code 0
```

```
input.txt x      output.txt x
1 18018 ✓        1 6 10 48 54 55 295 329 416 435 484 513 ✓
2 590254 13280 588086 840115 721688 ✓    562 610 726 781 903 1009 1069 1138 ✓
  679505 32001 200965 148408 513955 ✓    1197 1209 1217 1224 1232 1253 1266 ✓
  238564 297905 935197 753659 904152,    1468 1479 1501 1545 1689 1694 1697 ✓
  668387 566246 272303 292229 ✓    1823 1863 1906 1990 2124 2204 2215 ✓
  883499 361686 371796 189807 191474,    2238 2349 2399 2423 2513 2530 2664 ✓
  625450 945221 731746 62537 198256,    2906 2912 2956 2988 3027 3034 3082 ✓
  370145 883539 460197 201223 ✓    3188 3211 3368 3381 3420 3469 3473 ✓
  779949 884432 7791 528082 884466 ✓    3490 3504 3622 3627 3728 3734 3761 ✓
  411844 719363 129168 474399 258663,    3785 3830 3847 3899 4066 4114 4162 ✓
  317815 77651 414913 8466 166243 ✓    4198 4249 4355 4449 4504 4576 4631 ✓
  691104 757480 648838 100589 36175 ✓    4772 4906 4908 4957 4977 5027 5031 ✓
  944923 885473 103877 762920 423091,    5046 5085 5120 5141 5175 5391 5461 ✓
  267943 189744 60838 501781 593907,    5511 5619 5620 5693 5703 5890 5894 ✓
  207038 119539 928339 60475 280645,    5901 5909 5974 6126 6133 6181 6199 ✓
  55018 168539 205951 642316 219155,    6212 6258 6334 6403 6432 6522 6565 ✓
  171079 787109 964249 44088 902510,    6572 6578 6610 6639 6661 6663 6715 ✓
  560801 125759 970490 189583 ✓    6760 6834 6893 6895 6960 6973 7057 ✓
  226411 714310 385482 960451 249276,    7086 7093 7104 7137 7255 7387 7631 ✓
  611667 297705 820737 144922 ✓    7633 7664 7695 7701 7731 7749 7775 ✓
  279738 200985 882958 535550 313144,    7791 7808 7851 7952 7957 8031 8072 ✓
  587609 793442 270661 961670 ✓    8242 8272 8275 8436 8466 8586 8716 ✓
  634331 861369 517923 577356 393807,    8747 8750 8764 8805 8929 9012 9049 ✓
  201859 599462 154332 222777 ✓    9084 9114 9278 9301 9360 9465 9538 ✓

Run test 1 x
/usr/local/bin/python3.12 /Users/kseniatoIstuhina/PycharmProjects/Algo_labs/lab2
/task1/test 1.py
0.3264776249998249c
1.7098779678344727 MB
Process finished with exit code 0
```

Результат работы кода на максимальных и минимальных значениях:

The screenshot displays the PyCharm IDE interface during a test run. The top section shows two files: `input.txt` and `output.txt`. `input.txt` contains a list of 20,000 numbers, starting with 20000 and ending with 994330. `output.txt` contains a list of 20,000 numbers, starting with 66 and ending with 7901. The Run console at the bottom shows the execution path: `/usr/local/bin/python3.12 /Users/kseniatoIstuhina/PycharmProjects/Algo_labs/Lab2 /task1/test 1.py`. The console also displays the memory usage: 0.22967508394503966c and 3.7833776473999023 MB. The process finished with exit code 0.

The screenshot displays the PyCharm IDE interface during a test run. The top section shows two files: `input.txt` and `output.txt`. `input.txt` contains 2 numbers: 1 and 984132. `output.txt` contains 1 number: 984132. The Run console at the bottom shows the execution path: `/usr/local/bin/python3.12 /Users/kseniatoIstuhina/PycharmProjects/Algo_labs/Lab2 /task1/test 1.py`. The console also displays the memory usage: 0.0017532500205561519c and 0.0196533203125 MB. The process finished with exit code 0.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0017532500205561519с	0.0196533203125 MB
Пример из задачи	0.14203095802804455с	1.320887565612793 MB
Пример из задачи	0.3264776249998249с	1.7098779678344727 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.22967508394503966с	3.7833776473999023 MB

Вывод по задаче: Алгоритм работает достаточно быстро, как в среднем случае, так и наихудшим случае.

Задача №5. Представитель большинства

Текст задачи.

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

Листинг кода.

```
def majority(a):
    dic = {}
    for i in a:
        dic[i] = dic.get(i, 0) + 1

    for k, v in dic.items():
        if v > len(a) // 2:
            return 1

    return 0

with open('output.txt', 'w') as f:
```

```

file = open('input.txt')
n = int(file.readline())
list_input = list(map(int, file.readline().split()))
f.write(str(majority(list_input)))

```

Текстовое объяснение решения.

Я использую словарь для записи количества чисел, потом прохожу по его элементам и если какое либо значение больше чем $n // 2$, то возвращаю 1, если таких значений нет, 0.

Результат работы кода на примерах из текста задачи:

input.txt ×			:	output.txt ×		
1	5	✓		1	1	
2	2 3 9 2 2					

input.txt ×			:	output.txt ×		
1	4	✓		1	0	
2	1 2 3 4					

Результат работы кода на максимальных и минимальных значениях:

input.txt ×			:	output.txt ×		
1	100000	✓		1	0	
2	881634047 482000441 825083704 494438025 ↵					
	↵ 925205948 947504844 214872216 14057450 ↵					
	↵ 861134385 258889502 956281025 797143593 ↵					
	↵ 740416070 934848414 132940737 217342417 ↵					
	↵ 962754919 315556848 922959082 839267417 ↵					
	↵ 3330840 916971828 834093175 180157239 ↵					
	↵ 761085647 320465851 783064575 688809236 ↵					
	↵ 424956935 26121041 512555153 837214272 ↵					
	↵ 415853776 268462479 101225994 621226272 ↵					
	↵ 791715610 576082966 340727475 899354034 ↵					

input.txt ×			:	output.txt ×	
1	1	✓		1	1
2	1				

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0002115829847753048 с	0.018624305725097656 МВ
Пример из задачи	0.00018554204143583775 с	0.01867961883544922 МВ
Пример из задачи	0.00028624996775761247 с	0.018675804138183594 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.1273922499967739 с	10.944957733154297 МВ

Вывод по задаче: данная программа работает за линейное время и не превышает лимит памяти.

Задача №7. Поиск максимального подмассива за линейное время

Текст задачи.

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

Листинг кода.

```
def find_max(a):  
    max_sum, summ = 0, 0  
    for i in range(len(a)):  
        if summ == 0:  
            start_indx = i  
        summ += a[i]  
        if max_sum < summ:  
            max_sum = summ  
            end_indx = i  
        if summ < 0:  
            summ = 0  
    return a[start_indx:end_indx + 1]  
  
with open('output.txt', 'w') as file:  
    f_input = list(map(int,  
open('input.txt').readline().split()))  
    a = find_max(f_input)  
    file.write(' '.join(map(str, a)))
```

Текстовое объяснение решения.

Ввожу две переменные максимальную сумму и текущую. Прохожу по файлу и в случае, если текущая сумма равна нулю, то запоминаю начальный индекс(с которого будет в дальнейшем начинаться счет). Далее каждую итерацию я прибавляю к текущей сумме элемент. Далее идет проверка на максимальность, если максимальная сумма меньше текущей, то я обновляю данные макс суммы и запоминаю индекс, как конечный. Если сумма меньше нуля, то она не может быть наибольшей, поэтому я обнуляю текущую сумму и счет начинается сначала. В конце я вывожу массив от начального, до конечного индекса(с максимальной суммой)

Результат работы кода на примерах из текста задачи:

input.txt	output.txt
1 13 -3 -25 20 -3 -16 -23 18 20 -7 12 -5 -22 15 -4	1 18 20 -7 12

Результат работы кода на максимальных и минимальных значениях:

input.txt	output.txt
1 1	1 1

input.txt	output.txt
1 294147278 281464333 -199347849 -308762082 777271123 371817841 787331118 39202 192423698 172189120 -527362077 -552841434 260488689 239064608 206403243 843470828 -713897938 634465743 -724309451 846468361 403328018 592569932 497670145 182407035 7288119 516284560 12617016 -635848282 -905502842 500840438 475292864 -709721101 -534555936 -86879602 681710477 -912243950 786345073 311770172 964437477 -619580376 -844699471 -677078954 555361839 -349600358 369932286 -693223340 144166028 104335803 132420534 591822908 987445591 -711635589 803657346 313749951 580392122 592579477 -102943301 -374073439 704428982 22003665 -380591726 -209114631 949047756 -604391958 -503845252 983715032 140604514 487041635 755854540 964668988 426560747 -738010002 787564151 383576265 802108753 291758243 -405119477 -867551053 936577854 926361843 911477813 -2503474 582621836 138521816 -771987353 26428068 604653167 13044656 378553872 -455562200 -136928382 957198657 -444835713 171694221 204702262 222171862 884677273 -464707676 971668385 345422355 957475697 399290273 -452099761 723548938 -611193233 498827562 -883266672 310416989 517847070 883871056 464846476 61810223 -450597479 585206636 -156906114 577023313 -895521920 395791783 -684882999 663805229 -712311506 -592113732 -919002548 618172326 -719973023 -906041743 923720659 45438707 -964215414 -735645352 -274866902 552628320 -65439628 111734683 948399909 275324664 783261194 -945118816 -866798399 -130398847 671477257 -346651583 340661325 -978295312 605373478 617340330 474143689 -275463454 787497632 -505585848 499929117 -480398661	1 505772456 -170197022 -326549675 525369568 793986437 66872465 -246378532 -661737208 689434409 226438001 638367912 841304113 552573429 888156435 375021029 870302067 231203727 261188231 920225744 986916190 923837629 453068044 -640410188 910513862 611832736 57073878 129853480 -164186550 -171443766 351066368 398890583 357212846 663594975 928452923 668841612 -770521647 -230170458 -398737562 746934575 -761243867 -991551189 -788726861 682124590 -532308359 80404615 502804648 -98306520 210913415 -421833171 -944601408 -793952619 345362124 -254824057 972847662 -238474855 163319229 802422196 533746104 810050215 421110675 495613112 621355329 331046523 575767716 636757873 -910127736 658973022 -32813343 274371440 528930886 678969633 814231462 679281472 409421609 -334167698 -290392650 784238582 407511153 -539784058 -22290329 106573573 190452422 756323132 -167573128 -974693397 870042469 -470365184 -66150547 431625964 669980710 -388751470 -237259493 342901060 700729900 -974884947 -920901695 87070434 533005485 -581981844 -480429346 559710400 727702591 -497754060 -836710397 -737052917 915145451 -253400871 791908174 304651365 884671122 375122680 48683433 978463361 483171008 846136537 362216209 474247317 -378705707 547712682 -117579288 733796735 753268240 138591015 515806825 -893578890 -804428791 196040318 -288041813 261500411 725276027 272824999 920093069 778932730 667705707 330346281 977695093 142500302 -413862198 247098800 442027772 -799477454 467410774 950940761 804436354 326532633 151619472 401824087 128645459 -349211401 -271673416 973772354 -568111122 -323546259 597613501

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	5.8833000366576016e-05 секунд	0.005827903747558594 МВ
Пример из задачи	9.850000060396269e-05 секунд	0.006003379821777344 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.07804337500056135 секунд	3.817852020263672 МВ

Вывод по задаче: алгоритм работает на больших значениях достаточно быстро

Дополнительные задачи

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.

```
file = open('input.txt')

n, mass = int(file.readline()), list(map(int,
file.readline().split()))

k, mass_find = int(file.readline()), list(map(int,
file.readline().split()))

def binary_search(mas, what_find):
    l = 0
    r = len(mas) - 1
    while l <= r:
        mid = (l + r) // 2
        if what_find == mas[mid]:
            return mid
        elif what_find < mas[mid]:
            r = mid - 1
        else:
            l = mid + 1
    return -1
```

```
def play(mas, maas_find):
    return [binary_search(mas, maas_find[i]) for i in
range(len(maas_find))]

with open('output.txt', 'w') as f:
    f.write(' '.join(map(str, play(mass, mass_find))))
```

Текстовое объяснение решения.

Внутри функции ввожу две переменные - левая и правая граница. Далее, до тех пор пока мы не добрались до конца списка, мы находим середину списка, в случае, если элемент по середине равен искомому элементу, возвращаем индекс этого элемента. Если элемент посередине больше, чем искомый, то мы сдвигаем правую границу до серединного индекса, в обратном случае тоже самое с левой границей. Далее я в функции play перебираю элементы списка чисел, которые нужно найти и записываю в список найденные индексы, либо -1(если элемента в списке нет).

Результат работы кода на примерах из текста задачи:

input.txt ×		:	output.txt ×	
1	5	✓	1	2 0 -1 0 -1
2	1 5 8 12 13			
3	5			
4	8 1 23 1 11			

Результат работы кода на максимальных и минимальных значениях:

input.txt ×		:	output.txt ×	
1	1	✓	1	0
2	1			
3	1			
4	1			

Задача №3. Число инверсий

Текст задачи.

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

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

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

Листинг кода.

```
def search(mass, copy_mass, l, mid, r):

    i = k = l

    j = mid + 1

    count_inf = 0

    while i <= mid and j <= r:

        if mass[i] <= mass[j]:

            copy_mass[k] = mass[i]

            i += 1

        else:

            copy_mass[k] = mass[j]

            j += 1

            count_inf += (mid - i + 1)

        k += 1

    while i <= mid:

        copy_mass[k] = mass[i]

        k += 1
```

```

        i += 1

    for i in range(l, r + 1):
        mass[i] = copy_mass[i]
    return count_inf

def search_inversions(mass, copy_mass, l, r):
    if r <= l:
        return 0

    mid = (l + r) // 2
    count_inf = 0
    count_inf += search_inversions(mass, copy_mass, l, mid)
    count_inf += search_inversions(mass, copy_mass, mid + 1, r)
    count_inf += search(mass, copy_mass, l, mid, r)

    return count_inf

with open('output.txt', 'w') as f:
    file = open('input.txt')
    n = int(file.readline())
    nums = list(map(int, file.readline().split()))
    nums_copy = nums.copy()
    f.write(str(search_inversions(nums, nums_copy, 0, n -
1)))

```

Текстовое объяснение решения.

Функция `search` в которую я передаю массив, вспомогательный массив, левую границу, середину, правую границу. Далее, таким же алгоритмом, как в задаче 1 прохожу по массиву, сортирую его и в отдельную переменную записываю число инверсий на каждом шаге, если они есть. Циклом `for` переношу значения из вспомогательного массива в основной. Функция возвращает целое число - инверсий. Вторая функция делит массив на подмассивы и рекурсивно находит инверсии в каждом.

Результат работы кода на примерах из текста задачи:

input.txt			output.txt	
1	10	✓	1	17
2	1 8 2 1 4 7 3 2 3 6			

Результат работы кода на максимальных и минимальных значениях:

test_max_min.py	input.txt	test.py	output.txt
100000		✓	1 2497899061
986212577 613887437 174288863 841778834 327094351			
443404694 860865103 722073399 480718825 57915925			
631210778 203123056 228424687 560294288 333964204			
166766260 674338275 401436166 503545992 233385341			
361590313 98656786 238407668 780010629 525270931			
402199294 680524953 137556742 603653758 861939297			
122655410 916550524 378547771 19153412 959067938			
428669347 485657580 553257038 45999006 886529319			
603305370 985195518 325698735 809441693 382369894			
960113427 980308036 80472534 443150024 949297247			
537684826 547717350 390583802 978720957 669616714			
826197804 705760511 381400562 838593886 325076427			
339879056 927109112 779161698 711307519 428025012			
308330646 863062066 551920192 613657400 855839562			
768632822 753139094 429562863 379162406 379566469			
982666925 648686669 565102444 486216031 273860159			
8682641 875342311 243735175 789650556 811781827			
916666280 148171387 641321229 209699572 943806452			
977510746 401666264 102779363 606472316 889517814			
693285612 534618185 120088050 449112481 646013533			
20086086 158354514 593804472 732183353 531932398			
208710699 819249302 766761578 67153404 843898703			
843502886 857405644 643562473 622870794 363540936			
724740833 497816123 964288096 472819888 667899072			
314026599 339739354 786886641 769324549 973681690			
579227144 293774688 446223205 301906726 887813003			
582307624 297875493 819401425 443405159 447076268			
833846705 705126500 701318805 486492063 36537546			
688314700 456380244 452711634 449958954 23037710			
532864401 629006492 592031235 393998444 49555960			
801530694 32189955 925544472 745424114 547936021			
837244070 305041013 550321530 552024326 237774154			

input.txt		:	output.txt	
1	1	✓	1	0
2	1			

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000226374999328982 08 секунд	0.018799781799316406 MB
Пример из задачи	0.000113666999823180 96 секунд	0.006013870239257812 5 MB
Верхняя граница диапазона значений входных данных из текста задачи	2.17517908300033 секунд	8.967170715332031 MB

Вывод по задаче: алгоритм сортировки с использованием сортировки слиянием работает намного быстрее, чем алгоритм перебора элементов массива в двух циклах.

Задача №8. Умножение многочленов

Текст задачи.

Даны два многочлена. Нужно получить их произведение.

Листинг кода.

```
def mul_polynomials(a, b, n):  
    res = [0] * (n + n - 1)  
    for cof_a in range(n):  
        for cof_b in range(n):  
            res[cof_a + cof_b] += a[cof_b] * b[cof_a]  
    return res  
  
with open('output.txt', 'w') as f:  
    file = open('input.txt')  
    n = int(file.readline())  
    a, b = list(map(int, file.readline().split())),  
list(map(int, file.readline().split()))  
    f.write(' '.join(map(str, mul_polynomials(a, b, n))))
```

Текстовое объяснение решения.

Прохожу по коэффициентам и в результирующий список записываю нужные коэф(каждый член одного, на каждый член второго, затем складываю полученные произведения)

Результат работы кода на примерах из текста задачи:

≡ input.txt ×			:	≡ output.txt ×		
1	3	✓		1	15	13 33 9 10
2	3 2 5					
3	5 1 2					

	Время выполнения	Затраты памяти
Пример из задачи	9.604099977877922e-05 секунд	0.00600433349609375 МВ

Вывод по задаче: программа работает корректно

Вывод

В этой лабораторной работе я научилась писать различные алгоритмы, включая алгоритм сортировки слиянием и метод декомпозиции, так же вспомнила такие алгоритмы, как бинарный поиск или поиск максимального подмассива.