

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 23

Выполнила:
Федорова М. В.
К3139

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка слиянием	3
Задача №6. Поиск максимальной прибыли	5
Задача №7. Поиск максимального подмассива за линейное время	7
Дополнительные задачи	9
Задача №3. Число инверсий	9
Задача №4. Бинарный поиск	13
Задача №5. Представитель большинства	15
Вывод	17

Задачи по варианту

Задача №1. Сортировка слиянием

1. Используя *псевдокод* процедур Merge и Merge-sort из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python и проверьте сортировку, создав несколько случайных массивов, подходящих под параметры:
 - **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 2 \cdot 10^4$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .
 - **Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
 - Ограничение по времени. 2сек.
 - Ограничение по памяти. 256 мб.
 2. Для проверки можно выбрать наихудший случай, когда сортируется массив размера 1000, 10^4 , 10^5 чисел порядка 10^9 , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний. Сравните, например, с сортировкой вставкой на этих же данных.
 3. Перепишите процедуру Merge так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива L или R скопированы обратно в массив A , после чего в этот массив копируются элементы, оставшиеся в непустом массиве.
- или перепишите процедуру Merge (и, соответственно, Merge-sort) так, чтобы в ней не использовались значения границ и середины - p , r и q .

```

1  def merge(left_arr, right_arr): 1 usage
2      sorted_arr = []
3      left_arr_index = right_arr_index = 0
4      left_arr_length, right_arr_length = len(left_arr), len(right_arr)
5      for _ in range(len(left_arr) + len(right_arr)):
6          if left_arr_index < len(left_arr) and right_arr_index < len(right_arr):
7              if left_arr[left_arr_index] <= right_arr[right_arr_index]:
8                  sorted_arr.append(left_arr[left_arr_index])
9                  left_arr_index += 1
10             else:
11                 sorted_arr.append(right_arr[right_arr_index])
12                 right_arr_index += 1
13             elif left_arr_index == left_arr_length:
14                 sorted_arr.append(right_arr[right_arr_index])
15                 right_arr_index += 1
16             elif right_arr_index == right_arr_length:
17                 sorted_arr.append(left_arr[left_arr_index])
18                 left_arr_index += 1
19     return sorted_arr
20
21
22  def merge_sort(arr): 5 usages
23      if len(arr) <= 1:
24          return arr
25      m = len(arr) // 2
26      left_arr = merge_sort(arr[:m])
27      right_arr = merge_sort(arr[m:])
28      return merge(left_arr, right_arr)
29

```

```

30
31  if __name__ == "__main__":
32      with open("input.txt", "r") as file:
33          n = int(file.readline())
34          arr = [int(i) for i in file.readline().split()]
35      with open("output.txt", "w") as file:
36          if 0 <= n <= 2 * 10 ** 4 and min(arr) >= -10 ** 9 and max(arr) < 10 ** 9:
37              sorted_arr = merge_sort(arr)
38              file.write(" ".join(str(a) for a in sorted_arr))
39          else:
40              print("Число в массиве по модулю превосходит 10^9 или количество элементов не соответствует ограничениям")

```

1. Создаём функцию *merge()*, принимая в ней переменные *left_arr* и *right_arr*
2. Merge:
 - Создаем пустой массив *sorted_arr*

- Устанавливаем индексы *left_arr_index* и *right_arr_index*, которые будут использоваться для отслеживания текущих элементов в левых и правых массивах соответственно.
- Определяются длины массивов *left_arr_length* и *right_arr_length*.
- Цикл выполняется $\text{len}(\text{left_arr}) + \text{len}(\text{right_arr})$ раз, что соответствует общей длине двух массивов.
- В каждой итерации проверяем три условия:
1) Сравнение элементов:

Если индексы обоих массивов меньше их длин, сравниваются текущие элементы (по индексам *left_arr_index* и *right_arr_index*).

Меньший элемент добавляется в *sorted_arr*, а соответствующий индекс увеличивается на 1.

2) Элементы в *left_arr* закончились:

Если все элементы в *left_arr* были добавлены ($\text{left_arr_index} == \text{left_arr_length}$), оставшиеся элементы из *right_arr* добавляются в *sorted_arr*, и индекс *right_arr_index* увеличивается.

3) Элементы в *right_arr* закончились:

Аналогично, если все элементы в *right_arr* были добавлены ($\text{right_arr_index} == \text{right_arr_length}$), оставшиеся элементы из *left_arr* добавляются в *sorted_arr*, и индекс *left_arr_index* увеличивается.

- После завершения цикла функция возвращает *sorted_arr*, который содержит все элементы из *left_arr* и *right_arr*, отсортированные по возрастанию.

3. Merge_sort:

- Если массив пуст или кол-во элементов равно одному, то возвращаем сам массив
- Иначе создаем переменную m , которая будет отвечать за индекс середины массива
- Разделяем массив на две части и возвращаем их

4. Далее открываем файл input.txt, считываем сам массив, затем открываем файл output.txt и заносим туда уже отсортированный массив, если он подходит под ограничения

5. Результат:

1	5
2	23 1 45 1000 34
1	1 23 34 45 1000

6. Пишем тесты:

```
1 import psutil
2 import time
3 from alg.alg_lab2.task1.src.index_1 import merge_sort
4 t_start = time.perf_counter()
5 arr_1=[ 1,2,3,4,5]
6 arr_2=[12,45,2,100,67,122]
7 arr_3=[int(i) for i in range(10**9 - 10**4 + 1, 10**9 + 1)][::-1]
8 print(merge_sort(arr_3))
9 print("Время работы: %s секунд " % (time.perf_counter() - t_start))
10 print(f"Память: {psutil.Process().memory_info().rss / 1024 ** 2:.2f} МБ")
```

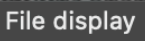
	Время выполнения, s	Затраты памяти, МВ
Нижняя граница диапазона значений входных данных из текста задачи	0.0005	12.61
Пример из задачи	0.00051	12.72
Верхняя граница диапазона значений входных данных из текста задачи	0.022	13.81

Вывод по задаче: В ходе выполнения задания была разработана программа для сортировки массива чисел с использованием алгоритма слияния. Также реализована функция для измерения времени работы программы и контроля за потреблением оперативной памяти, что позволяет оценить эффективность реализации.

Задача №6. Поиск максимальной прибыли

Используя *псевдокод* процедур Find Maximum Subarray и Find Max Crossing Subarray из презентации к Лекции 2 (страницы 25-26), напишите программу поиска максимального подмассива.

Примените ваш алгоритм для ответа на следующий вопрос. Допустим, у нас есть данные по акциям какой-либо фирмы за последний месяц (год, или иной срок).

Проанализируйте  срок и выдайте ответ, в какой из дней при покупке единицы акции данной фирмы, и в какой из дней продажи, вы бы получили максимальную прибыль? Выдайте дату покупки, дату продажи и максимальную прибыль.

Вы можете использовать любые данные для своего анализа. Например, я набрала в Google "акции" и мне поиск выдал акции Газпрома, [тут](#) - можно скачать информацию по стоимости акций за любой период. (Перейдя по ссылке, нажмите на вкладку "Настройки" → "Скачать")

Соответственно, вам нужно только выбрать данные, посчитать *изменение цены* и применить алгоритм поиска максимального подмассива.

- **Формат входного файла** в данном случае на ваше усмотрение.
- **Формат выходного файла (output.txt).** Выведите название фирмы, рассматриваемый вами срок изменения акций, дату покупки и дату продажи единицы акции, чтобы получилась максимальная выгода; и сумма этой прибыли.

```

1  def find_maximum_subarray(prices): 1 usage new *
2      max_profit = 0
3      current_profit = 0
4      start = 0
5      end = 0
6      s = 0
7      for i in range(1, len(prices)):
8          current_profit += prices[i] - prices[i - 1]
9          if current_profit < 0:
10             current_profit = 0
11             s = i
12             if current_profit > max_profit:
13                 max_profit = current_profit
14                 start = s
15                 end = i
16
17      return max_profit, start, end

```

```

18 def main(): 1 usage new *
19     dates = []
20     prices = []
21     with open("input.txt", 'r') as file:
22         name_firm = file.readline().strip()
23         while (line := file.readline().strip()) != "":
24             arr_file = line.split(" ")
25             date = arr_file[0]
26             price_str = arr_file[1]
27             price = int(price_str)
28             dates.append(date)
29             prices.append(price)
30     profit, buy_date, sell_date = find_maximum_subarray(prices)
31     print(profit, buy_date, sell_date)
32     period = "Сентябрь 2024"
33     with open('output.txt', 'w') as f:
34         f.write(f"Фирма: {name_firm}\n")
35         f.write(f"Срок: {period}\n")
36         f.write(f"Дата покупки: {dates[buy_date]}\n")
37         f.write(f"Дата продажи: {dates[sell_date]}\n")
38         f.write(f"Максимальная прибыль: {profit}\n")
39
40     if __name__ == "__main__":
41         main()

```

1. Создаём функцию *find_maximum_subarray(prices)*:

Эта функция предназначена для нахождения максимальной прибыли от акций, используя подход, основанный на динамическом программировании.

- Переменные:

max_profit — это максимальная прибыль, которую мы можем получить.

current_profit — это текущая прибыль, которую мы рассчитываем на основе изменений цен.

start и *end* — индексы, которые указывают на начало и конец подмассива, который дает максимальную прибыль.

s — временный индекс, который используется для отслеживания начала текущего подмассива.

- Цикл по ценам:

Мы проходим по массиву цен, начиная с первого элемента. На каждой итерации мы добавляем разницу между текущей и предыдущей ценой к *current_profit*. Это позволяет нам отслеживать изменения прибыли.

- Обработка отрицательной текущей прибыли:

Если *current_profit* становится отрицательным, это значит, что дальнейшие продажи не принесут прибыли. Поэтому мы сбрасываем *current_profit* до нуля и устанавливаем *s* в текущий индекс *i*, чтобы начать новый расчет прибыли.

- Обработка максимальной текущей прибыли:

Если текущая прибыль превышает *max_profit*, мы обновляем *max_profit* и индексы *start* и *end*.

- Возврат результата:

Функция возвращает максимальную прибыль и индексы начала и конца подмассива, который дает эту прибыль.

2. Создаем функцию *main()*:

Мы открываем файл *input.txt*, считываем название фирмы и затем извлекаем даты и цены акций. Эти данные сохраняются в списках

dates и *prices*. Мы вызываем функцию *find_maximum_subarray*, чтобы получить максимальную прибыль и соответствующие индексы дат покупки и продажи. Результаты записываются в файл *output.txt*, включая название фирмы, период, даты покупки и продажи, а также максимальную прибыль.

Результат:

1	<u>Gazprom</u>
2	12.09.2024 132
3	13.09.2024 128
4	14.09.2024 144
5	15.09.2024 146
6	16.09.2024 143
7	17.09.2024 155
8	18.09.2024 167
9	19.09.2024 132
10	20.09.2024 140
11	21.09.2024 141
12	22.09.2024 110
13	23.09.2024 160
14	24.09.2024 158
15	25.09.2024 154
16	26.09.2024 155
17	27.09.2024 158
18	28.09.2024 161
19	29.09.2024 162
20	30.09.2024 166

1	<u>Фирма: Gazprom</u>
2	<u>Срок: Сентябрь 2024</u>
3	<u>Дата покупки: 22.09.2024</u>
4	<u>Дата продажи: 30.09.2024</u>
5	<u>Максимальная прибыль: 56</u>

Пишем тесты:

```

1 import unittest
2 from alg.alg_lab2.task6.src.index_6 import find_maximum_subarray
3 class CaesarTestCase(unittest.TestCase):
4     def test_one(self):
5         dates = []
6         prices = []
7         with open("test_input1.txt", 'r') as file:
8             name_firm = file.readline()
9             while (line := file.readline()) != "":
10                 arr_file = line.split(" ")
11                 print(arr_file)
12                 date = arr_file[0]
13                 price_str = arr_file[1]
14                 price = int(price_str)
15                 dates.append(date)
16                 prices.append(price)
17         profit, buy_date, sell_date = find_maximum_subarray(prices)
18         res = [profit, buy_date, sell_date]
19         self.assertEqual(res, second: [46,8,18])
20
21     def test_two(self):
22         dates = []
23         prices = []
24         with open("test_input2.txt", 'r') as file:
25             name_firm = file.readline()
26             while (line := file.readline()) != "":
27                 arr_file = line.split(" ")
28                 print(arr_file)
29                 date = arr_file[0]
30                 price_str = arr_file[1]
31                 price = int(price_str)
32                 dates.append(date)
33                 prices.append(price)
34         profit, buy_date, sell_date = find_maximum_subarray(prices)
35         res = [profit, buy_date, sell_date]
36         self.assertEqual(res, second: [56,8,18])

```

```

37     def test_three(self):
38         dates = []
39         prices = []
40         with open("test_input3.txt", 'r') as file:
41             name_firm = file.readline()
42             while (line := file.readline()) != "":
43                 arr_file = line.split(" ")
44                 print(arr_file)
45                 date = arr_file[0]
46                 price_str = arr_file[1]
47                 price = int(price_str)
48                 dates.append(date)
49                 prices.append(price)
50         profit, buy_date, sell_date = find_maximum_subarray(prices)
51         res = [profit, buy_date, sell_date]
52         self.assertEqual(res, second: [14,5,13])

```

Все тесты проходят:

```
Ran 3 tests in 0.002s

OK

Process finished with exit code 0
```

Вывод по задаче: В ходе выполнения задания была разработана программа, которая эффективно вычисляет максимальную прибыль от покупки и продажи акций, используя метод, основанный на отслеживании изменений цен.

Задача №7. Поиск максимального подмассива за линейное время

File display

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

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

```

1  def max_subarray(arr): 3 usages new *
2      max_current = arr[0]
3      max_global = arr[0]
4      start_index = 0
5      end_index = 0
6      temp_start_index = 0
7      for i in range(1, len(arr)):
8          if arr[i] > max_current + arr[i]:
9              max_current = arr[i]
10             temp_start_index = i
11         else:
12             max_current += arr[i]
13             if max_current > max_global:
14                 max_global = max_current
15                 start_index = temp_start_index
16                 end_index = i
17     return arr[start_index:end_index + 1]
18
19
20  if __name__ == "__main__":
21      with open("input.txt", "r") as file:
22          n = int(file.readline())
23          arr = [int(i) for i in file.readline().split()]
24      with open("output.txt", "w") as file:
25          if 0 <= n <= 2 * 10 ** 4 and min(arr) >= -10 ** 9 and max(arr) < 10 ** 9:
26              max_subarr = max_subarray(arr)
27              file.write(" ".join(str(a) for a in max_subarr))
28          else:
29              print("Число в массиве по модулю превосходит 10^9 или количество элементов не со
30

```

1) Создаем функцию *max_subarray*:

- *max_current* и *max_global* инициализируются первым элементом массива, представляя текущую максимальную сумму и общую максимальную сумму, найденную до сих пор.
- *start_index*, *end_index* и *temp_start_index* инициализируются для отслеживания границ максимального подмассива.
- Создаем цикл, который начинается со второго элемента:

Для каждого элемента $arr[i]$ алгоритм проверяет, больше ли текущий элемент сам по себе, чем сумма $max_current$ и $arr[i]$. Если это так, $max_current$ сбрасывается на $arr[i]$, а $temp_start_index$ обновляется до текущего индекса.

В противном случае $arr[i]$ добавляется к $max_current$.

- Если $max_current$ превышает max_global , обновляется max_global , а $start_index$ и end_index устанавливаются на индексы нового максимального подмассива.
- Наконец, функция возвращает подмассив от $start_index$ до end_index , включая их, который представляет собой непрерывный подмассив с максимальной суммой.

2) Открываем файл input.txt, считываем массив, далее открываем файл output.txt, если массив подходит под ограничения, то вызываем функцию `max_subarray` и заносим результат в файл

3) Пишем тесты:

```
1 import psutil
2 import time
3 import random
4 from alg.alg_lab2.task7.src.index_7 import max_subarray
5 t_start = time.perf_counter()
6 arr_1=[ 1,2,3,4,5]
7 arr_2=[12,-45,2,100,67,122]
8 arr_3=[random.randint(-10**9, 10**9) for _ in range(10**4)]
9 print(max_subarray(arr_2))
10 print("Время работы: %s секунд " % (time.perf_counter() - t_start))
11 print(f"Память: {psutil.Process().memory_info().rss / 1024 ** 2:.2f} МБ")
```

	Время выполнения, s	Затраты памяти, МВ
Нижняя граница диапазона значений входных данных из текста задачи	0.004	13.05
Пример из задачи	0.0047	13.11
Верхняя граница диапазона значений входных данных из текста задачи	0.01	13.48

Вывод по задаче: Данная программа эффективно находит максимальный подмассив и соответствует заданным ограничениям по входным данным.

Она подходит для работы с большими массивами благодаря использованию алгоритма с линейной временной сложностью $O(n)$.

Дополнительные задачи

Задача №3. Число инверсий

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

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

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

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9 .
- **Формат выходного файла (output.txt).** В выходной файл надо вывести число инверсий в массиве.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

```

1 def merge_and_count(arr, temp_arr, left, mid, right): 1 usage
2     i = left
3     j = mid + 1
4     k = left
5     inv_count = 0
6
7     while i <= mid and j <= right:
8         if arr[i] <= arr[j]:
9             temp_arr[k] = arr[i]
10            i += 1
11        else:
12            temp_arr[k] = arr[j]
13            inv_count += (mid - i + 1)
14            j += 1
15        k += 1
16
17    while i <= mid:
18        temp_arr[k] = arr[i]
19        i += 1
20        k += 1
21
22    while j <= right:
23        temp_arr[k] = arr[j]
24        j += 1
25        k += 1
26
27    for i in range(left, right + 1):
28        arr[i] = temp_arr[i]
29
30    return inv_count

```

```

32 def merge_sort_and_count(arr, temp_arr, left, right): 3 usages new *
33     inv_count = 0
34     if left < right:
35         mid = (left + right) // 2
36
37         inv_count += merge_sort_and_count(arr, temp_arr, left, mid)
38         inv_count += merge_sort_and_count(arr, temp_arr, mid + 1, right)
39         inv_count += merge_and_count(arr, temp_arr, left, mid, right)
40
41     return inv_count
42
43
44 def count_inversions(n, arr): 3 usages new *
45     temp_array = [0] * n
46     return merge_sort_and_count(arr, temp_array, left: 0, n - 1)
47
48
49 if __name__ == "__main__":
50     with open("input.txt", "r") as file:
51         n = int(file.readline())
52         arr = [int(i) for i in file.readline().split()]
53
54     with open("output.txt", "w") as file:
55         if 0 <= n <= 2 * 10 ** 4 and min(arr) >= -10 ** 9 and max(arr) < 10 ** 9:
56             inversions = count_inversions(n, arr)
57             file.write(str(inversions))
58         else:
59             print("Число в массиве по модулю превосходит 10^9 или количество элементов")

```


1. Создаём функцию *merge_and_count*:

- *i* указывает на начало первой половины (от *left* до *mid*).
- *j* указывает на начало второй половины (от *mid* + 1 до *right*).
- *k* используется для записи в *temp_arr*.
- *inv_count* инициализируется нулем и будет использоваться для подсчета инверсий.
- Вход в цикл *while* происходит, пока индексы *i* и *j* находятся в пределах своих половин.
- Если элемент в первой половине (*arr[i]*) меньше или равен элементу во второй половине (*arr[j]*), он копируется в *temp_arr*, и *i* увеличивается.
- Если *arr[i]* больше *arr[j]*, это означает, что все оставшиеся элементы в первой половине (от *i* до *mid*) образуют инверсии с *arr[j]*. Эти инверсии подсчитываются и добавляются к *inv_count*, затем элемент *arr[j]* копируется в *temp_arr*, и *j* увеличивается.
- После выхода из первого цикла два последующих цикла *while* копируют оставшиеся элементы из первой и второй половин в *temp_arr*.
- В финальном цикле все элементы из *temp_arr* копируются обратно в *arr* в диапазоне от *left* до *right*.
- Функция возвращает общее количество инверсий, найденных при слиянии.

2. Функция *count_inversion*:

- Проверяет, что *left* меньше *right*. Это условие необходимо для того, чтобы убедиться, что в массиве есть более одного элемента для сортировки. Если условие выполняется, массив разделяется на две половины, а *mid* вычисляется как средний индекс.
- Функция рекурсивно вызывается для левой половины массива (*left* до *mid*). Инверсии из этой части добавляются к *inv_count*.
- Функция рекурсивно вызывается для правой половины массива (*mid* + 1 до *right*). Инверсии из этой части также добавляются к *inv_count*.

- После обработки обеих половин вызывается функция *merge_and_count*, которая объединяет две отсортированные части и считает инверсии между ними. Это количество инверсий добавляется к *inv_count*.
- В конце функция возвращает общее количество инверсий, найденных в текущем диапазоне массива.

3. Открываем файл *input.txt*, заносим наш массив в переменную *arr*.
Далее открываем файл *output.txt* проверяет условия на допустимые размеры входных данных, если они соответствуют заданным границам, то вызывается функция *count_inversion*, куда мы передаем наш массив

4. Результат:

1	5
2	23 1 45 1000 34

1	3
---	---

5. Пишем тесты:

```

1 import psutil
2 import time
3 from alg.alg_lab2.task3.src.index_3 import count_inversions
4 t_start = time.perf_counter()
5 arr_1=[ 1,2,3,4,5]
6 arr_2=[12,45,2,100,67,122]
7 arr_3=[int(i) for i in range(10**9 - 10**4 + 1, 10**9 + 1)][::-1]
8 print( count_inversions(len(arr_3),arr_3))
9 print ("Время работы: %s секунд " % (time.perf_counter () - t_start))
10 print(f"Память: {psutil.Process().memory_info().rss / 1024 ** 2:.2f} МБ")

```

Вывод по задаче: в ходе работы над задачей я создала алгоритм, который представляет из себя рекурсивный подход для сортировки массива и подсчета инверсий, разбивая массив на более мелкие подмассивы, а затем сливая их с подсчетом инверсий. Это обеспечивает общую временную сложность $O(n \log n)$, что делает алгоритм эффективным для обработки больших массивов

Задача №4. Бинарный поиск

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

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве, и последовательность $a_0 < a_1 < \dots < a_{n-1}$ из n **различных** положительных целых чисел в порядке возрастания, $1 \leq a_i \leq 10^9$ для всех $0 \leq i < n$. Следующая строка содержит число k , $1 \leq k \leq 10^5$ и k положительных целых чисел b_0, \dots, b_{k-1} , $1 \leq b_j \leq 10^9$ для всех $0 \leq j < k$.
- **Формат выходного файла (output.txt).** Для всех i от 0 до $k - 1$ вывести индекс $0 \leq j \leq n - 1$, такой что $a_i = b_j$ или -1, если такого числа в массиве нет.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
5	2 0 -1 0 -1
1 5 8 12 13	
5	
8 1 23 1 11	

В этом примере есть возрастающая последовательность из $a_0 = 1, a_1 = 5, a_2 = 8, a_3 = 12$ и $a_4 = 13$ длиной в $n = 5$ и пять чисел для поиска: 8 1 23 1 11. Видно, что $a_2 = 8$ и $a_0 = 1$, но чисел 23 и 11 нет в последовательности a , поэтому они имеют индекс -1. В итоге ответ: 2 0 -1 0 -1.

```

1  def binary_search(arr, target): 5 usages new *
2      left, right = 0, len(arr) - 1
3      while left <= right:
4          mid = left + (right - left) // 2
5          if arr[mid] == target:
6              return mid
7          elif arr[mid] < target:
8              left = mid + 1
9          else:
10             right = mid - 1
11     return -1
12 def insertionSort(n, arr): 5 usages new *
13     if n <= 1:
14         return
15     for i in range(1, n):
16         key = arr[i]
17         j = i-1
18         while j >= 0 and key < arr[j]:
19             arr[j+1] = arr[j]
20             j -= 1
21         arr[j+1] = key
22
23 if __name__ == "__main__":
24     with open("input.txt", "r") as file:
25         n1 = int(file.readline())
26         arr = [int(i) for i in file.readline().split()]
27         insertionSort(n1, arr)
28         n2 = int(file.readline())
29         search_elements = [int(i) for i in file.readline().split()]
30
31     with open("output.txt", "w") as file:
32         results = []
33         for element in search_elements:
34             result = binary_search(arr, element)
35             results.append(str(result))
36     file.write(" ".join(results))

```

1. Создаём функцию *binary_search*:

- Цикл продолжается, пока левая граница не превышает правую. Это означает, что мы еще не просмотрели весь массив
- Вычисляем средний индекс *mid*. Это делается для предотвращения переполнения при больших значениях *left* и *right*.
- Если элемент в средней позиции равен *target*, мы возвращаем индекс *mid*, так как нашли элемент.

- Если элемент $arr[mid]$ больше $target$, значит, $target$ находится слева от середины, и мы сдвигаем правую границу влево: $right = mid - 1$.
- Если элемент $arr[mid]$ меньше $target$, это означает, что $target$ находится справа от середины, и мы сдвигаем левую границу вправо: $left = mid + 1$.
- Если мы вышли из цикла, это означает, что элемент не был найден в массиве, и мы возвращаем -1 .

2. Открываем файл `input.txt`, заносим данные в массив `arr` и `search_elements`, а потом применяем функцию `inersionSort` на массив `arr`. Далее открываем файл `output.txt`, создаем цикл `for`, перебираем каждый элемент массива `search_elements`, каждый раз вызывается функция `binary_search`, и результат ее выполнения заносится в массив, а дальше после завершения цикла, мы выводим этот массив.

3. Результат:

1	5
2	1 5 8 12 13
3	5
4	8 1 23 1 11

1	2 0 -1 0 -1
---	-------------

4. Пишем тесты:

```
1 import unittest
2 from alg.alg_lab2.task4.src.index_4 import binary_search
3 from alg.alg_lab2.task4.src.index_4 import insertionSort
4
5 arr_search=[8,5,10,-1,100]
6 class CaesarTestCase(unittest.TestCase): new *
7     def test_one(self): new *
8         results = []
9         for el in arr_search:
10             arr_1=[-1,5,3,10,8]
11             insertionSort(len(arr_1), arr_1)
12             res=binary_search(arr_1, el)
13             results.append(str(res))
14         str_res = " ".join(results)
15         self.assertEqual(str_res, second: "3 2 4 0 -1")
16
17     def test_two(self): new *
18         results = []
19         for el in arr_search:
20             arr_2= [6,100,5,0,10]
21             insertionSort(len(arr_2), arr_2)
22             res = binary_search(arr_2, el)
23             results.append(str(res))
24         str_res = " ".join(results)
25         self.assertEqual(str_res, second: "-1 1 3 -1 4")
26
27     def test_three(self): new *
28         results = []
29         for el in arr_search:
30             arr_3 = [0,0,0,0,0]
31             insertionSort(len(arr_3), arr_3)
32             res = binary_search(arr_3, el)
33             results.append(str(res))
34         str_res=" ".join(results)
35         self.assertEqual(str_res, second: "-1 -1 -1 -1 -1")
```

все тесты проходят

Вывод по задаче: в ходе работы над задачей я научилась создавать алгоритм бинарного поиска.

Задача №5. Представитель большинства

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

```
Majority(A):
for i from 1 to n:
    current_element = a[i]
    count = 0
    for j from 1 to n:
        if a[j] = current_element:
            count = count+1
    if count > n/2:
        return a[i]
return "нет элемента большинства"
```

Очевидно, время выполнения этого алгоритма квадратично. Ваша цель - использовать метод "Разделяй и властвуй" для разработки алгоритма проверки, содержится ли во входной последовательности элемент, который встречается больше половины раз, за время $O(n \log n)$.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве. Во второй строке находятся n положительных целых чисел, по модулю не превосходящих 10^9 , $0 \leq a_i \leq 10^9$.
- **Формат выходного файла (output.txt).** Выведите 1, если во входной последовательности есть элемент, который встречается строго больше половины раз; в противном случае - 0.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример 1:

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

Число "2" встречается больше $5/2$ раз.

- Пример 2:

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

Нет элемента, встречающегося больше $n/2$ раз.

```

1 def majority_element(arr): 5 usages new *
2     def find_candidate(start, end): new *
3         if start == end:
4             return arr[start]
5         mid = (start + end) // 2
6         left_candidate = find_candidate(start, mid)
7         right_candidate = find_candidate(mid + 1, end)
8         if left_candidate == right_candidate:
9             return left_candidate
10        else:
11            return left_candidate if count_occurrences(left_candidate) > count_occurrences(
12                right_candidate) else right_candidate
13
14        def count_occurrences(cand): new *
15            count = sum([1 for x in arr if x == cand])
16            return count
17
18        candidate = find_candidate(start=0, len(arr) - 1)
19        occurrences_count = count_occurrences(candidate)
20        if occurrences_count > len(arr) // 2:
21            return 1
22        else:
23            return 0
24
25
26 if __name__ == "__main__":
27     with open('input.txt', 'r') as file:
28         n = file.readline()
29         array = [int(i) for i in file.readline().split()]
30
31     result = majority_element(array)
32
33     with open('output.txt', 'w') as file:
34         file.write(str(result))

```

1. Создаём функцию *majority_element*:

- Рекурсивно находит кандидата на мажоритарный элемент в подмассиве от *start* до *end*.
- Если диапазон сжимается до одного элемента, он возвращает этот элемент.
- Делит массив на две части и находит кандидатов для каждой половины.
- Сравнивает двух кандидатов и возвращает того, который встречается чаще (с помощью *count_occurrences*).
- функция *count_occurrences(cand)*:
Считает, сколько раз элемент *cand* встречается в массиве *arr*.

- Вызывает *find_candidate* для всего массива, чтобы получить кандидата.
- Проверяет количество вхождений кандидата с помощью *count_occurrences*.
- Если кандидат встречается более чем в половине массива, возвращает *1* (существует мажоритарный элемент), иначе возвращает *0*.

2. Открываем файл `input.txt`, считываем массив и заносим его в переменную *array*. Далее открываем файл `output.txt`, вызываем функцию *majority_element*, результат записываем в файл

3. Результат:

1	5
2	2 3 9 2 2

1	1
---	---

4. Пишем тесты:

```

1  import unittest
2  from alg.alg_lab2.task5.src.index_5 import majority_element
3
4
5  arr_search=[8,5,10,-1,100]
6  class CaesarTestCase(unittest.TestCase): new *
7  def test_one(self): new *
8      str_number="1 3 1 2 1 1"
9      arr=[int(i) for i in str_number.split()]
10     res= majority_element(arr)
11     self.assertEqual(res, second: 1)
12
13  def test_two(self): new *
14     str_number = "0 0 0 0 0"
15     arr = [int(i) for i in str_number.split()]
16     res = majority_element(arr)
17     self.assertEqual(res, second: 1)
18  def test_three(self): new *
19     str_number = "1 2 3 4 5 6"
20     arr = [int(i) for i in str_number.split()]
21     res = majority_element(arr)
22     self.assertEqual(res, second: 0)

```

все тесты проходят

Вывод по задаче: В ходе выполнения задания была разработана функция *majority_element*, которая определяет наличие мажоритарного элемента в массиве. Используя метод "разделяй и властвуй", функция эффективно ищет кандидата на мажоритарный

Вывод

В ходе работы были изучены алгоритмы сортировки слиянием, а также поиск максимальной прибыли и поиск максимального подмассива за линейное время. Также был реализован подсчет инверсий при сортировке массива, бинарный поиск и алгоритм нахождения мажоритарного элемента в массиве. Протестированы на максимальных и минимальных входных данных. Было проанализировано время работы каждого алгоритма и объем используемой памяти.