# (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 12

Выполнил:

Колпаков А.С.

K3139

Проверил:

Афанасьев А.В

Санкт-Петербург 2024 г.

# Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка слиянием	3
Задача №3. Число инверсий	7
Задача №8. Умножение полиномов	10
Дополнительные задачи	14
Задача №4. Бинарный поиск	14
Задача №5. Представитель большинства	17
Задача №7. Поиск максимального подмассива за линейное время	20
Вывод	23

# 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 рандомных массивов, подходящих под параметры:
  - Формат входного файла (input.txt). В первой строке входного файла содержится число n ( $1 \le n \le 2 \cdot 10^4$ ) 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.

```
def merge(A, p, q, r):
  n1 = q - p + 1
  n2 = r - q

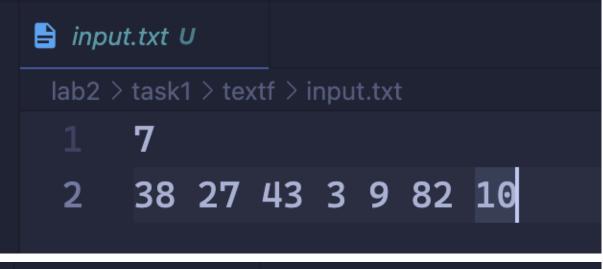
L = [0] * (n1+1)
  R = [0] * (n2+1)
```

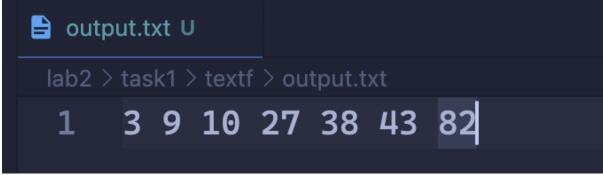
```
for i in range(n1):
  L[i] = A[p+i]
for j in range(n2):
  R[j] = A[q+j+1]
i, j, k = 0, 0, p
while i < n1 and j < n2:
   if L[i] <= R[j]:</pre>
    A[k] = L[i]
    i += 1
   else:
    A[k] = R[j]
    j += 1
  k += 1
while i < n1:
  A[k] = L[i]
  i += 1
  k += 1
while j < n2:
  A[k] = R[j]
  j += 1
  k += 1
def merge sort(A,p,r):
if p < r:
     q = (p+r) // 2
```

```
merge_sort(A, p, q)
merge_sort(A, q+1, r)
merge(A,p,q,r)
return A
```

- 1. Задаем функцию merge, которая дублирует псевдокод презентации.
- 2. Только теперь, сигналом к остановке служит нехватка элементов из какого-либо массива. В таком случае, все оставшиеся элементы из большего массива записываются в общий массив.
- 3. Далее создаем функцию merge-sort, в которой делим один массив на два подмассива, при этом каждый раз вызывая merge-sort с подмассивами. В конце объединяем все массивы функцией merge и возвращаем результат.

# Результат работы кода:





Тест примера

Время работы: 0.0008473340003547492 секунд

Память: 0.013241767883300781 МБ

Худший случай

Время работы: 0.29432408299908275 секунд

Память: 0.07659149169921875 МБ

Средний случай

Время работы: 0.28810362500007614 секунд

Память: 0.07659149169921875 МБ

Лучший случай

Время работы: 0.3383603329984908 секунд

Память: 0.07659149169921875 МБ

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.3383603329984908 секунд	0.07659149169921875 МБ
Пример из задачи	0.0008473340003547492 секунд	0.01324176788330078 1 МБ
Медиана диапазона значений входных данных из текста задачи	0.28810362500007614 секунд	0.07659149169921875 МБ
Верхняя граница диапазона значений входных данных из текста задачи	0.29432408299908275 секунд	0.07659149169921875 МБ

#### Вывод по задаче:

Программа успешно реализует алгоритм сортировки слиянием. Тестирование показало правильную работу алгоритма.

# Задача №3. Число инверсий

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

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем. Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

```
def merge(A, p, q, r, inversion count):
 n1 = q - p + 1
n2 = r - q
 L = [0] * (n1+1)
 R = [0] * (n2+1)
 for i in range(n1):
  L[i] = A[p+i]
 for j in range(n2):
   R[j] = A[q+j+1]
 i, j, k = 0, 0, p
 while i < n1 and j < n2:
   if L[i] <= R[j]:
     A[k] = L[i]
     i += 1
   else:
     A[k] = R[j]
     j += 1
     inversion count += (n1 - i)
```

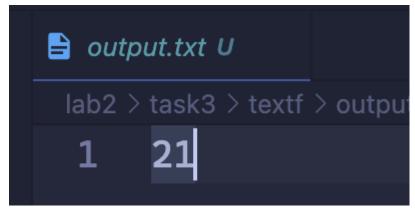
```
k += 1
while i < n1:
  A[k] = L[i]
  i += 1
  k += 1
while j < n2:
  A[k] = R[j]
  j += 1
  k += 1
return inversion count
def merge sort(A,p,r, inversion count):
if p < r:
    q = (p+r) // 2
           inversion count = merge sort(A, p,
inversion count)
        inversion count = merge sort(A, q + 1,
inversion count)
           inversion count = merge(A, p, q,
inversion count)
return inversion count
```

- 1. Тело алгоритма аналогично алгоритму из задания 1, за исключением того, что теперь в функцию merge необходимо передавать переменную inversion\_count.
- 2. Переменная inversion\_count обновляется в том случае, если элемент из левого подмассива оказался больше элемента из правого.

3. В функции merge\_sort мы каждый раз обновляем переменную inversion count и возвращаем ее в результате выполнения алгоритма.

Результат работы кода на примерах из текста задачи:





Тест примера Время работы: 0.0006604999998671701 секунд Память: 0.013310432434082031 МБ

	Время выполнения	Затраты памяти
Пример из задачи	0.000660499999867170 1 секунд	0.013310432434082031 МБ

#### Вывод по задаче:

Программа успешно реализует модифицированный алгоритм сортировки слиянием, отслеживающий количество инверсий в массиве.

#### Задача №8. Умножение многочленов

- Формат входного файла (input.txt). В первой строке число n порядок многочленов A и B. Во второй строке коэффициенты многочлена A через пробел. В третьей строке коэффициенты многочлена B через пробел.
- Формат выходного файла (output.txt). Ответ одна строка, коэффициенты многочлена C(x) = A(x)B(x) через пробел.
- Нужно использовать метод "Разделяй и властвуй". Подсказка: любой многочлен A(x) можно разделить на 2 части, например,  $A(x) = 4x^3 + 3x^2 + 2x + 1$  разделим на  $A_1 = 4x + 3$  и  $A_2 = 2x + 1$ . И многочлен  $B(x) = x^3 + 2x^2 + 3x + 4$  разделим на 2 части:  $B_1 = x + 2$ ,  $B_2 = 3x + 4$ . Тогда произведение  $C = A(x) * B(x) = (A_1B_1)x^n + (A_1B_2 + A_2B_1)x^{n/2} + A_2B_2$  требуется 4 произведения (проверьте правильность данной формулы). Можно использовать формулу Гаусса и обойтись всего тремя произведениями.

```
def add_polynomials(A, B):
    return [A[i] + B[i] for i in range(len(A))]

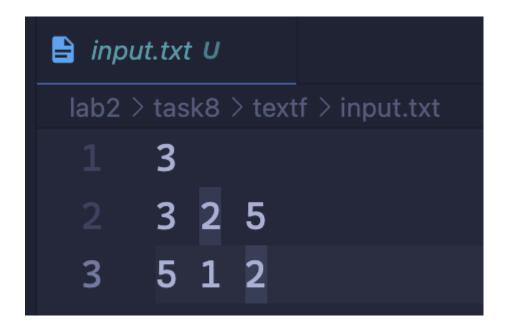
def subtract_polynomials(A, B):
    n = max(len(A), len(B))
    A += [0] * (n - len(A))
    B += [0] * (n - len(B))
    return [A[i] - B[i] for i in range(len(A))]

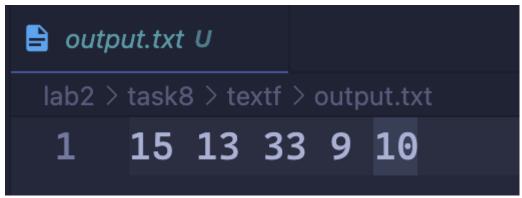
def multiply_polynomials(A, B, n):
    if len(A) == 1:
        return [A[0] * b for b in B]
    if len(B) == 1:
        return [B[0] * a for a in A]
```

```
if len(A) % 2 != 0:
  A.append(0)
  B.append(0)
 n = len(A)
A1, A2 = A[:n//2], A[n//2:]
B1, B2 = B[:n//2], B[n//2:]
P1 = multiply polynomials(A1, B1, n)
P2 = multiply polynomials(A2, B2, n)
A1 plus A2 = add polynomials (A1, A2)
B1 plus B2 = add polynomials(B1, B2)
P3 = multiply polynomials (A1 plus A2, B1 plus B2,
n)
middle term =
subtract polynomials(subtract polynomials(P3, P1),
P2)
result = [0] * (2 * n - 1)
 for i in range(len(P1)):
     result[i] += P1[i]
 for i in range(len(middle term)):
     result[i + n//2] += middle term[i]
for i in range(len(P2)):
     result[i + n] += P2[i]
while len(result) > 1 and result[-1] == 0:
    result.pop()
return result
```

- 1. Задаем функцию add\_polynomials и substract\_polynomials, которые соответственно складывают значения коэффициентов полиномов и вычитают их.
- 2. Далее задаем функцию multiply\_polynomials. Первые строчки функцию описывают ситуацию, когда полином равен константе, в таком случае мы просто перемножаем все значения одного полинома на эту константу.
- 3. Далее, если количество коэффициентов полинома нечетно, то добавляем нули.
- 4. Далее пользуемся идеей "Разделяй и властвуй", разделяя полиномы на два подмножества.
- 5. Далее рекурсивно умножаем эти подмножества между собой и потом складываем их.
- 6. После чего пользуемся формулой Гаусса для умножения многочленов и запишем все в общую переменную результата, удалив лишние нули в конце.

Результат работы кода на примерах из текста задачи:





r i i i i i i i i i i i i i py ci i o i i casito, ceses, eese, py

Тест примера

Время работы: 0.0006110840004112106 секунд

Память: 0.013337135314941406 МБ

	Время выполнения	Затраты памяти
Пример из задачи	0.000611084000411210 6 секунд	0.013337135314941406 МБ

# Вывод по задаче:

Программа эффективно, пользуясь формулой Гаусса для умножения полиномов, производит вычисления коэффициентов заданных полиномов.

#### Дополнительные задачи

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

```
def bin_search(A, target, 1, r):
   if 1 > r:
       return -1

mid =(1+r)//2

if A[mid] == target:
      return mid

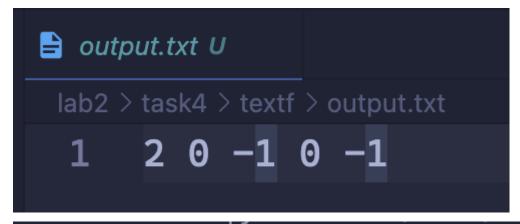
if target > A[mid]:
    return bin_search(A, target, mid+1, r)

if target < A[mid]:
   return bin_search(A, target, l, mid-1)</pre>
```

```
def find_ind(A, B, n):
  res = []
  for num in B:
    res.append(bin_search(A, num, 0, n - 1))
    return res
```

- 1. Задаем функцию bin\_search, которая реализует бинарный поиск в заланном массиве.
- 2. Задаем условие, при котором, если левый индекс больше правого, значит нужного элемента в массиве нет и тогда алгоритм возвращает -1.
- 3. Потом разделяем массива на два подмассива и смотрим, нужный элемент больше среднего в массиве, если да, то возвращаем функцию bin\_search с левым индексом равным среднему элементу массива + 1, в ином случае возвращаем функцию с правым индексом равным среднему элементу массива -1.
- 4. В функции find\_ind описываем с помощью цикла поиск индекса каждого элемента заданного массива и возвращаем результат.

Результат работы кода на примерах из текста задачи:



Тест примера

Время работы: 0.0007560000012745149 секунд

Память: 0.013357162475585938 МБ

	Время выполнения	Затраты памяти
Пример из задачи	0.000756000001274514 9 секунд	0.013357162475585938 МБ

# Вывод по задаче:

Алгоритм бинарного поиска эффективно справляется с поиском индекса заданного элемента в отсортированном массиве чисел.

# Задача №5. Представитель большинства

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

- Формат входного файла (input.txt). В первой строке входного файла содержится число n ( $1 \le n \le 10^5$ ) число элементов в массиве. Во второй строке находятся n положительных целых чисел, по модулю не превосходящих  $10^9$ ,  $0 \le a_i \le 10^9$ .
- **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.

```
def count_majority(A, target, 1, r):
    count = 0
    for i in range(1, r+1):
        if A[i] == target:
            count += 1
    return count

def majority(A, 1, r):
    mid = (1 + r) // 2

if 1 == r:
    return A[1]
    al = majority(A, 1, mid)
    ar = majority(A, mid+1, r)

if al == ar:
```

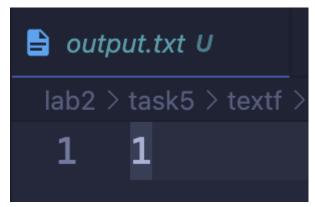
```
return al
al count = count majority(A, al, l, r)
ar count = count majority(A, ar, 1, r)
if al count > (r - 1) // 2:
   return al
elif ar count > (r - 1) // 2:
   return ar
return None
def has majority(A, n):
res = majority(A, 0, n-1)
if res is not None:
   count = count majority(A, res, 0, n-1)
  if count > n // 2:
     return 1
return 0
```

- 1. Задаем функцию count\_majority, которая считает количество заданного элемента в определенном промежутке массива A.
- 2. Далее описываем саму функцию поиска элемента, встречающегося больше n/2 pa3.
- 3. Если массив состоит из одного элемента, возвращаем его, в ином случае разделяем массива на подмассивы рекурсивно.
- 4. Если наиболее часто встречающиеся элементы подмассивов совпадают, то возвращаем этот элемент, в ином случае считаем количество этих элементов в заданном диапазоне.

- 5. Если количество одного из этих элементов больше чем половина количества элементов, заданных на диапазоне, то возвращаем один из них, если таких элементов в принципе нет, то возвращаем None.
- 6. В финальной функции has\_majority описываем, что, если элемент большинства есть и его количество в массиве больше чем n / 2, тогда возвращаем 1, в ином случае возвращаем 0.

Результат работы кода на примерах из текста задачи:





Тест примера

Время работы: 0.0005589579996012617 секунд

Память: 0.013225555419921875 МБ

	Время выполнения	Затраты памяти
Пример из задачи	0.000558957999601261 7 секунд	0.013225555419921875 МБ

#### Вывод по задаче:

Алгоритм поиска представителя большинства, используя метод "Разделяй и властвуй", эффективно ищет элемент, который встречается больше половины раз.

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

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

В этом случае у вас возможны 2 варианта тестирования: первый предполагает создание рандомного массива чисел, аналогично задаче  $\mathbb{N}^{0}1$  (в этом случае формат входного и выходного файла смотрите там). Второй вариант - взять любые данные по акциям какой-либо компании, аналогично задаче  $\mathbb{N}^{0}6$ .

```
def max_subarray(arr):
    max_sum = arr[0]
    current_sum = arr[0]
    st = end = 0
    temp_st = 0

for i in range(1, len(arr)):
    if arr[i] > current_sum + arr[i]:
        current_sum = arr[i]
        temp_st = i
    else:
```

```
current_sum += arr[i]

if current_sum > max_sum:
    max_sum = current_sum
    st = temp_st
    end = i

return arr[st:end + 1]
```

- 1. Задаем две переменные с суммами, а также две переменные с индексами начала и конца максимального подмассива, а также переменную индекса первого элемента текущего подмассива.
- 2. Далее с помощью цикла сравниваем, если текущий элемент больше суммы с этим элементом. если да, то перезаписываем текущую сумму. Иначе прибавляем в текущую сумму этот элемент.
- 3. Если текущая сумма больше максимальной, то перезаписываем ее в максимальную и обновляем переменные st и end.

Результат работы кода на примерах из текста задачи:



Тест примера

Время работы: 0.0006313329977274407 секунд

Память: 0.013316154479980469 МБ

Худший случай

Время работы: 0.012648916999751236 секунд

Память: 0.037754058837890625 МБ

Средний случай

Время работы: 0.012136375000409316 секунд

Память: 0.033367156982421875 МБ

Лучший случай

Время работы: 0.009892582998872967 секунд

Память: 0.039058685302734375 МБ

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.009892582998872967 секунд	0.03905868530273437 5 МБ
Пример из задачи	0.0006313329977274407 секунд	0.01331615447998046 9 МБ
Медиана диапазона значений входных данных из текста задачи	0.012136375000409316 секунд	0.03336715698242187 5 МБ
Верхняя граница диапазона значений входных данных	0.012648916999751236 секунд	0.03775405883789062 5 МБ

#### Вывод по задаче:

Данный алгоритм эффективно находит максимальный подмассив за линейное время.

#### Вывод

Рекурсивные алгоритмы, основанные на принципе "Разделяй и властвуй" можно использовать во многих задачах, легко задающихся правилам упрощения. Их плюсом является простота написания, при обозначенных правилах, но в случаях с большими значениями и глубокой рекурсией может потребоваться значительное количество памяти.