Задача A. Сортировка слиянием с приколом

В задаче требовалось сделать сортировку слиянием, и обновлять количество инверсий при операции *merge*. Как это делать? Можно заметить, что мы сливаем "левый"и "правый"массивы, где любой элемент правого имел больший исходный индекс, чем любой элемент левого.

Это значит, что по определению инверсии, если левый и правый массивы уже были отсортированы рекурсивно (и мы учли все инверсии при той сортировке), у нас остались пары $i, j : \ i < j, a_i > a_j$ только такие, где i в левом массиве, а j в правом. Когда мы сливаем элемент из правого массива, можно представить себе, что это j, и тогда все возможные i находятся в левом массиве, и еще не были добавлены в слияние (так как они больше чем a_j). Значит, это еще не просмотренные элементы левого массива. Тогда требуется в merge сделать что-то такое:

```
if a[ptr1] <= b[ptr2]:
    res.append(a[ptr1])
    ptr1 += 1
else:
    res.append(b[ptr2])
    ptr2 += 1
    answer += len(a) - ptr1
```

Задача B. Anti-qsort test

В этой задаче нужно было понять, какой случай худший для алгоритма. Заметьте, что если мы сделали сравнение, то мы всегда после него двигаем указатель. Это происходит либо потому что цикл сделал итерацию, либо условие не выполнилось, но тогда мы потом сделаем swap. Если swap не делался, то мы сделаем выход из цикла и это сравнение можно не учитывать.

Таким образом, мы делаем один проход и в нем делаем $O(n)$ шагов. Значит, количество сравнений $C(n) = C(a)+C(b)+n$. Оптимальным для нас разбиением будет a и b как $1 : (n-1)$, когда получается квадратичное количество сравнений.

Тогда нам нужно сделать сведение задачи от n к $n-1$. Чтобы разбиение убрало один элемент, надо сделать так, чтобы центральным элементом стал максимум.

Идеальным вариантом будет рекурсивное сведение — чтобы мы сначала строили массив для $n-1$, и затем строили из него массив для n. Для этого достаточно приписать n к концу массива и поменять n с центральным элементом. Тогда линейный проход во время partition сделает обратный свап и запустится рекурсивно на массиве длины $n-1$.

```
def gen(n):
    if n == 1:
        a[0] = 1
        return
    if (n == 2)
        a[0] = 1
        a[1] = 2
        return
    gen(n - 1)
    a[n - 1] = n
    swap(a[n - 1], a[(n - 1) / 2])
```

Задача C. Число

Разбор

В этой задаче важно было понять, как правильно сортировать список с ”кусками” числа. Первое, что приходит в голову – сделать сортировку в лексикографическом порядке и просто склеить части. Однако тут возникает проблема: к примеру, части числа 4 и 4321 были бы отсортированы в неправильном порядке, т.к. лексикографически 4321 больше, чем 4

Можно было бы долго пытаться придумать правило вычисления ключа для сортировки с учетом реализации на Python, а можно было воспользоваться компараторами аналогично другим языкам. Поймём, какой компаратор мы бы хотели использовать. При заданных x, y мы хотим поставить x вперёд, если число xy больше, чем число yx.

Асимптотика решения: $O(N \times \log N)$

Решение

```
from functools import cmp_to_key
from sys import stdin

def cmp(x, y):
    return 1 if x + y > y + x else -1

v = []
for line in stdin:
    v.append(line.rstrip())
v.sort(key=cmp_to_key(cmp), reverse=True)
print(''.join(v))
```

Задача D. Зеркальный код

Предлагалось сделать подсчет символов в исходной строке, после чего группами по 2 собирать палиндром. В конце надо поставить в середину минимальный символ, оставшийся в количестве 1.

```
half = ''
for i in range(26):
    while cnt[i] >= 2:
        half += chr(ord('A') + i))
        cnt[i] -= 2
middle = ''
for i in range(26):
    if cnt[i] >= 1:
middle = chr(ord('A') + i))
print(half + middle + half[::-1]
```

Задача E. Что? Да! Пузырек

В этой задаче требовалось посмотреть, как меняется количество шагов сортировки в зависимости от массива. Если воспользоваться фактом, что пузырек после 1 прохода ставит максимум в конец, мы поймем, что количество проходов для сортировки это (количество 1) - (количество непрерывных 1 в конце массива) + 1.

Тогда надо придумать, как считать количество 1 в конце массива за $O(1)$ на запрос. Предлагалось сделать амортизированный алгоритм, который идет справа налево и уменьшает глобальный указатель, если он указывает на 1

```
zero = n - 1
for it in range(n):
    print(ans + 1, end = " ")
    ans += 1
    arr[p[it]] = 1
while zero > -1 and arr[zero] == 1:
    ans -= 1
    zero -= 1
print(1)
```