Задача A. Сортировка через один

Будем добавлять элементы с чётными индексами в один дополнительный массив, а элементы с нечётными индексами — в другой дополнительный массив. Сортируем первый массив по убыванию, второй — по возрастанию. В конце выводим по очереди элементы из массивов: первый элемент из нечётного, первый из чётного, второй элемент из нечётного и так далее.

Задача B. Сортировка половин

Давайте разделим массив на две части. Отсортируем каждую отдельно. Вторую половину после сортировки нужно будет развернуть, чтобы она стала отсортированной по убыванию.

Задача C. Упаковка Груза

Для решения первой подзадачи необходимо перебрать все подмножества коробок, проверить каждое из них на корректность и найти корректное подмножество максимального размера. Реализовать это можно следующим образом. Для начала отсортируем коробки в порядке неубывания размера. Далее реализуем рекурсивный (или нерекурсивыный) перебор всех подмножеств множества коробок. Пусть в подмножество вошли коробки с номерами $i_1, i_2, \ldots, i_k$. Тогда для того, чтобы проверить набор на корректность, необходимо проверить что для любого $j \geqslant 2$ верно, что $\frac{s_{i_j}}{s_{i_{j-1}}} \geqslant K$ (данной проверки достаточно, так как коробки были отсортированы).

Асимптотика: $O(2^N \cdot N)$.

Рассмотрим решение второй подзадачи. В ней $K = 10^9$ — максимально возможное значение. Нетрудно понять, что при таком значении K ответ не бывает больше двух (размеры 1 и 10^9). Поэтому в случае, если есть хотя бы одна коробка размера 1 и хотя бы одна коробка размера 10^9, ответ будет равен двум. Во всех остальных случая ответ будет равен одному.

Асимптотика: $O(N)$.

Рассмотрим полное решение данной задачи. Отсортируем коробки по неубыванию размера. Далее будем набирать коробки жадно, в порядке сортировки. Нетрудно показать, что данный алгоритм оптимален. То есть, для начала возьмем в набор коробку минимального размера. После этого будем пропускать коробки больших размеров до тех пор, пока мы не сможем взять следующую коробку. Берем первую следующую подходящую коробку, после чего продолжаем алгоритм.

Асимптотика: $O(N \log N)$.

Пример решения на языке C++:

```
#include <cstdio>
#include <algorithm>

using namespace std;

const int N = 100100;

int a[N];

int main() {
    int n, k;
    scanf("%d%d", &n, &k);

    for (int i = 0; i < n; ++i) {
        scanf("%d", &a[i]);
    }

    sort(a, a + n);
    reverse(a, a + n);
```

```
    int ans = 1;
    int last = a[0];

    for (int i = 1; i < n; ++i) {
        if (1ll * a[i] * k <= last) {
            last = a[i];
            ++ans;
        }
    }

    printf("%d\n", ans);

    return 0;
}
```

Задача D. Олимпиада

Для начала вычислим, сколько задач мы успеем решить до конца тура. Так как до конца тура осталось T единиц времени, а на одну задачу уходит C единиц времени, то это количество равно: $M = \min\left(N, \left\lfloor \frac{T}{C} \right\rfloor\right)$.

Теперь, так как мы хотим набрать как можно больше баллов, нужно решать задачи, которые стоят как можно больше баллов. Для этого отсортируем баллы по убыванию, после чего просуммируем первые M чисел в порядке убывания.

Для решения первых двух групп можно воспользоваться любой квадратичной сортировкой, однако для получения полного балла необходимо воспользоваться любой быстрой сортировкой.

Асимптотика: $O(N \log N)$.

Пример решения на языке C++:

```
#include <cstdio>
#include <algorithm>

using namespace std;

const int N = 100100;

int a[N];

int main() {
    int t, c, n;
    scanf("%d%d%d", &t, &c, &n);

    for (int i = 0; i < n; ++i) {
        scanf("%d", &a[i]);
    }

    sort(a, a + n);
    reverse(a, a + n);

    int ans = 0;
    for (int i = 0; i < std::min(t / c, n); ++i) {
        ans += a[i];
    }
```

```
    printf("%d\n", ans);

    return 0;
}
```

Задача E. Дипломы в папках

Поймем, что если нам не повезло и то, пришлось просмотреть все стопки. Но можно не просматривать последнюю стопку, так как мы точно знаем, что диплом в ней, если он не встретился нам ранее.

Нам может постоянно не везти и придется просмотреть все кроме одной, тогда давайте сделаем её максимально возможной. А значит нам нужно отсортировать по размеру и просмотреть все кроме последней. Считаем сумму массива без максимального значения.

Задача F. Лука и локальная сеть динозавров

Решение заключается в том, чтобы отсортировать фигурки по координате x, в случае равенства сортируем по координате y. Теперь надо последовательно соединять динозавров, соседствующих в упорядоченном списке. Так как всего в списке n элементов, то проводов понадобится $n - 1$.

Пример решения на языке Python.

```
n = int(input())

points = []

for i in range(n):
    x = int(input())
    y = int(input())
    points.append((x, y, i + 1))

points.sort()
print(n - 1)
for i in range(n - 1):
    print(points[i][2], points[i + 1][2])
```

Задача G. Университетская команда

Для получения 25 баллов можно было просто перебрать все наборы из k чисел.

Далее заметим, что для фиксированного набора минимальная слабость команды достигается в том случае, если набор отсортирован по возрастанию или по убыванию. Тогда для получения 50 баллов можно любой квадратичной сортировкой отсортировать массив a, после чего перебрать все подотрезки длины k. Такое решение будет работать за $\mathcal{O}(n^2)$.

Теперь научимся быстро считать слабость на подотрезке. Заметим, что $|a_1 - a_2| + \ldots + |a_{k-1} - a_k| = (a_2 - a_1) + \ldots + (a_k - a_{k-1}) = a_k - a_1$. Таким образом можно за $\mathcal{O}(1)$ посчитать слабость на подотрезке.

Если использовать сортировку подсчетом, то решение будет работать за $\mathcal{O}(n + A)$, где A — максимальное число в массиве a. Такое решение набирает не менее 50 баллов.

Для получения решения на 100 баллов достаточно отсортировать массив любой быстрой сортировкой.

```
n = int(input())
k = int(input())
a = [int(input()) for i in range(n)]
a.sort()
```

```
ans = 10**9
for i in range(k - 1, n):
    ans = min(ans, a[i] - a[i - k + 1])
print(ans)
```

Задача H. Удаление данных

Для решения первой подзадачи можно было написать простейший перебор с возвратом. Такое решение набирало не менее 20 баллов.

Для решения второй подзадачи требовалось заметить следующий факт: **выгодно удалять элемент, соседний с одним из минимальных элементов**.

Таким образом, решение второй подзадачи выглядит следующий образом:

- Если в массиве остался один элемент, завершаем алгоритм.
- Если в массиве хотя бы два элемента, находим минимальный. Пусть он равен a_{min} (если их несколько, выбирем любой), и удаляем одного из его соседей. При этом к ответу добавляется a_{min}.

Чтобы решить задачу на 100%, нужно было заметить следующий факт: **При удалении элемента, соседнего с минимальным, минимум массива не меняется.** Это значит, что на каждой итерации цикла из предыдущего листинга к ответу будет прибавляться одно и то же число, равное минимальному элементу изначального массива a. Поскольку всего будет ровно $n - 1$ интерация, то ответ равен $a_{min} \cdot (n - 1)$.

Таким образом, решение на 100 баллов имеет следующий вид:

```
n = int(input())

mn = 10 ** 9
for i in range(n):
        cur = int(input())
        mn = min(mn, cur)

print(mn * (n - 1))
```

Задача I. Юные следопыты

Отсортируем всех следопытов по неубыванию неопытности. Пусть мы сформировали какую-то группу, как проверить, что эта группа валидная? Неопытности всех следопытов в группе должны быть не больше размера группы. Но мы отсортировали всех следопытов, поэтому самая большая неопытность у последнего следопыта из группы. Следовательно, чтобы проверить группу на валидность необходимо и достаточно проверить, что неопытность последнего следопыта в группе не превышает размер группы.

Можно заметить, что мы вообще не обращаем внимание на всех следопытов, кроме последнего, нам важно только их количество. Фактически, распределение на группы можно организовать следующим образом: сначала выберем следопытов, которые будут последними в своих группах, а потом выдадим им нужное количество других следопытов. Никогда нет смысла давать больше следопытов, чем нужно, если что, лишних можно оставить в лагере.

Как же оптимально выбрать последних следопытов? Мы хотим сформировать побольше групп, поэтому сами группы должны быть поменьше... Возникает идея следующего жадного алгоритма: будем каждый раз жадно брать самого левого (а значит с самым маленьким требуемым размером группы) следопыта, слева от которого достаточно следопытов, чтобы выдать ему валидную группу. Идея в том, что мы тратим минимальное количество следопытов, и оставляем максимальное количество потенциальных последних следопытов. Докажем эту жадность строго:

Решение задаётся позициями последних следопытов $1 \leqslant p_1 < p_2 < \ldots < p_k \leqslant n$. Заметим, что решение валидно тогда и только тогда, когда $e_{p_1} + e_{p_2} + \ldots + e_{p_i} \leqslant p_i$ для всех $1 \leqslant i \leqslant k$ (нам всегда хватает следопытов, чтобы сформировать первые i групп).

Пусть $1 \leqslant p_1 < p_2 < \ldots < p_k \leqslant n$ — жадное решение, $1 \leqslant q_1 < q_2 < \ldots < q_m \leqslant n$ — оптимальное решение с максимальным общим префиксом с жадным решением. Пусть t — позиция первого отличия. $t \leqslant k$, так как иначе жадное решение не смогло добавить $(k+1)$-ю группу, хотя это было возможно. $p_t < q_t$, так как иначе жадное решение взяло бы следопыта q_t вместо следопыта p_t. Так как следопыты отсортированы, из этого следует $e_{p_t} \leqslant e_{q_t}$. Но тогда легко понять, что $1 \leqslant q_1 < q_2 < \ldots < q_{t-1} < p_t < q_{t+1} < \ldots < q_m \leqslant n$ — это валидное оптимальное решение, и у него общий префикс с жадным строго больше, что противоречит выбору оптимального решения.

Чтобы реализовать это решение, достаточно отсортировать следопытов по неубыванию неопытности, потом идти слева направо и поддерживать количество неиспользованных следопытов. Как только мы видимо возможность создать новую группу, мы её создаём.

Задача J. Фокусы

Рассмотрим два фокуса с номерами i и j. Если $b_i \geqslant 0$ и $b_j \geqslant 0$, то выгодно сначала показать фокус, у которого значение a меньше. В противном случае выгодно сначала показать тот фокус, у которого значение b больше.