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. 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 G. Два альбома

(*Фольклор.*)

Для каждого альбома определим массив, в котором будем хранить уникальные номера марок этого альбома. Пусть `a[n]` и `b[m]` — массивы из этих номеров. В задаче требуется найти количество общих элементов массивов `a[n]`, `b[m]` и вывести эти элементы.

Подзадача 1. Простым перебором элементов двух массивов можно набрать 30 баллов.

Подзадача 2. Для нахождения общих элементов можно сравнить каждое число массива `a[n]` с каждым числом массива `b[m]`. Алгоритмическая сложность такого решения — $O(n \cdot m)$, оно набирает 60 баллов.

Подзадача 3. Объединим массивы `a[n]` и `b[m]` в один массив `c[n + m]` и отсортируем его по возрастанию. Тогда общие элементы исходных массивов будут встречаться в массиве `c[n + m]` ровно два раза. Осталось пройти по всему массиву и подсчитать количество *соседних* повторяющихся элементов и сами эти элементы.

Другая возможность полного решения — использование функции `set_intersection` из библиотеки шаблонов <algorithm> языка `C++`. Для этого нужно сначала отсортировать массивы `a[n]` и `b[m]`, а затем к полученным после сортировки массивам применить указанную функцию.

Алгоритмическая сложность таких решений — $O(n \log n + m \log m)$, они набирают 100 баллов.

Приведём основной фрагмент кода на языке `C++`:

```
sort(c.begin(), c.end());

vector <int> v;
for (int i = 1; i < n + m; ++i)
    if (c[i] == c[i - 1])
        v.push_back(c[i]);

cout << v.size() << endl;
for (int i = 0; i < v.size(); ++i)
    cout << v[i] << "␣";
```

Задача H. Перекраска Одежды

Рассмотрим решение задачи для первой группы тестов. Для этого посчитаем, сколько есть футболок первого и второго цвета. Обозначим эти количества как c_1 и c_2. Если какое-то из этих чисел не меньше, чем k, то перекрашивать футболки не придется, и ответ на задачу равен нулю. В противном случае придется перекрасить некоторые футболки. Нетрудно понять, что придется перекрасить $\min(k - c_1, k - c_2)$ футболок.

Асимптотика: $O(N)$.

Для решений второй подзадачи можно использовать следующий алгоритм. Переберем цвет, в который мы будем перекрашивать футболки. Пройдем циклом по всем футболкам и посчитаем, сколько есть футболок выбранного цвета, пусть их количество равно c. Тогда придется перекрасить $\max(0, k - c)$ футболок. Найдем минимальное значение данного количества по всем цветам, это и будет ответ на задачу.

Асимптотика: $O(N \cdot C)$.

Теперь рассмотрим полное решение данной задачи. Из решения для второй подзадачи нетрудно понять, что чем больше количество футболок некоторого выбранного цвета, тем меньше придется купить краски. Заведем массив cnt, в i-й ячейке которого сохраним, сколько есть футболок i-го цвета. Это можно сделать при считывании данных. Теперь пройдем циклом по данному массиву и найдем цвет, который встречается чаще всего (найдем максимальное значение cnt_i). Пусть это значение равно $maxC$. Тогда ответ равен: $\max(0, k - maxC)$.

Асимптотика: $O(N)$.

Пример решения на языке C++:

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

using namespace std;

const int N = 100100;

int cnt[N];

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

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

    int mx = 0;
    for (int i = 0; i < N; ++i) {
        mx = max(mx, cnt[i]);
    }

    printf("%d\n", max(0, k - mx));

    return 0;
}
```

Задача I. Позитивный Настрой

В первой подзадаче можно перебрать все перестановки индексов, их всего 8! и для каждой проверить, сохранит ли последовательность позитивное отношение к жизни. В процессе проверки, возможно переполнение и на некоторых языках программирования стоит использовать 64-х битный тип данных.

В решении на полный балл, в задаче нужно отсортировать новости в порядки убывания уровня радости и проверить, что в любой момент чтения новостей уровень радости неотрицательный. После этого вывести, либо -1, либо последовательность индексов в отсортированном массиве.

Приведем доказательство. Предположим, что подходит другая последовательность новостей, а последовательность отсортированных не подходит. Рассмотрим префиксные суммы в обоих массивах. Тогда у нас совпадает какой-то префикс элементов, а в i различие. Пусть в отсортированной последовательсности стоит элемент a_i, а в рассматриваемой b_i. Из свойств отсортированной последовательности, мы знаем, что $a_i > b_i$. Тогда мы можем в рассматриваемой последовательности поменять местами b_i и a_i. Эта последовательность тоже подходит, т.к. массив префиксных сумм увеличится между элементами b_i и a_i. Такими действиями мы приведем рассматриваемую последовательность к отсортированной. Сл-во отсортированная тоже подходит. Мы пришли к противоречию. Значит отсортированная послездовательность подходит

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.