E. Team Building

time limit per test: 3 seconds
memory limit per test: 256 megabytes

Alice, the president of club FCB, wants to build a team for the new volleyball tournament. The team should consist of p players playing in p different positions. She also recognizes the importance of audience support, so she wants to select k people as part of the audience.

There are n people in Byteland. Alice needs to select exactly p players, one for each position, and exactly k members of the audience from this pool of n people. Her ultimate goal is to maximize the total strength of the club.

The i-th of the n persons has an integer a_i associated with him — the strength he adds to the club if he is selected as a member of the audience.

For each person i and for each position j, Alice knows $s_{i,j}$ — the strength added by the i-th person to the club if he is selected to play in the j-th position.

Each person can be selected at most once as a player or a member of the audience. You have to choose exactly one player for each position.

Since Alice is busy, she needs you to help her find the maximum possible strength of the club that can be achieved by an optimal choice of players and the audience.

Input

The first line contains 3 integers n, p, k ($2 \le n \le 10^5$, $1 \le p \le 7$, $1 \le k, p + k \le n$).

The second line contains n integers $a_1, a_2, \ldots, a_n$. ($1 \le a_i \le 10^9$).

The i-th of the next n lines contains p integers $s_{i,1}, s_{i,2}, \ldots, s_{i,p}$. ($1 \le s_{i,j} \le 10^8$)

Output

Print a single integer res — the maximum possible strength of the club.

Examples

input Copy

```
4 1 2
1 16 10 3
18
19
13
15
```

output Copy

```
44
```

→ **Problem tags**

bitmasks dp greedy sortings *2300

No tag edit access

Problem Breakdown

Lets understand the problem first

You have:

- n people (n ≤ 100,000)
- p positions (p ≤ 7) very small
- k audience members
- Each person can be:
  - A player in exactly one position, or
  - An audience member, or
  - Not selected

Each person:

- Gives a[i] strength if chosen as audience
- Gives s[i][j] strength if chosen as player for position j

You must:

- Pick exactly one person per position → total p players
- Pick exactly k audience members
- No person can be used twice
- Maximize total strength

- We CANNOT bitmask people (too many)
- We CAN bitmask positions

Therefore:

- Mask will represent which positions are already filled.

Example:

p = 3
mask = 101_2 → positions {0, 2} are filled, 1 is empty.

So total number of masks = 2^p ≤ 2^7 = 128

This is perfect for DP.

Each person has two independent possible contributions:

1. As audience → a[i]
2. As player for any ONE position → s[i][j]

Since:

- Audience has no position
- Player occupies exactly one position
- And p is small

We process people one by one and maintain:

For each mask of filled positions, the best score after processing first i people.

DP State Definition

```
dp[mask] = maximum strength achievable after processing some people,
           where positions filled are exactly 'mask'
```

Greedy Observation for Audience:

If we:

- Pick x players so far
- Then among the remaining processed people,
- The best audience members are simply the k highest ai values among those not used as players

So instead of manually tracking audience count inside DP with another dimension, we do this:

We sort people by a[i] in descending order.

Then for every prefix of people:

- The first t people we did not use as players give us audience strength automatically.
- Because they're already in decreasing a[i].

This removes the need for a separate audience DP dimension.

This is a standard compression trick in this problem.

DP Transition Logic

We process people one by one.

For each person i, we update the DP:

For each current mask:

We have two options:

Option 1: Make this person an audience member

But we can only take at most k audience members.

How do we know how many audience we already used?

We already assigned:

- popcount(mask) players
- Total processed people = i
- So number of audience taken so far = i - popcount(mask)

So audience still allowed if:

i - popcount(mask) < k

Then:

new_dp[mask] = max(new_dp[mask], dp[mask] + a[i])

Option 2: Make this person a player

If some position j is empty in mask:

if ((mask & (1<<j)) == 0)

Then:

newMask = mask | (1<<j)
new_dp[newMask] = max(new_dp[newMask], dp[mask] + s[i][j])

Code

```cpp
#include <bits/stdc++.h>
using namespace std;

static const long long NEG_INF = -(1LL << 60);

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int n, p, k;
    cin >> n >> p >> k;

    vector<long long> a(n);
    for (int i = 0; i < n; i++)
        cin >> a[i];

    vector<vector<long long>> s(n, vector<long long>(p));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < p; j++)
            cin >> s[i][j];

    // Sort people by decreasing audience strength
    vector<int> order(n);
    iota(order.begin(), order.end(), 0);
    sort(order.begin(), order.end(), [&](int x, int y) {
        return a[x] > a[y];
    });

    int MAX_MASK = 1 << p;
    vector<long long> dp(MAX_MASK, NEG_INF);
    dp[0] = 0;

    // Process each person
    for (int idx = 0; idx < n; idx++) {
        int i = order[idx];
        vector<long long> new_dp = dp;

        for (int mask = 0; mask < MAX_MASK; mask++) {
            if (dp[mask] == NEG_INF) continue;

            int players_used = __builtin_popcount(mask);
            int audience_used = idx - players_used;

            // Option 1: use as audience
            if (audience_used < k) {
                new_dp[mask] = max(new_dp[mask], dp[mask] + a[i]);
            }

            // Option 2: use as player
            for (int j = 0; j < p; j++) {
                if (!(mask & (1 << j))) {
                    int newMask = mask | (1 << j);
                    new_dp[newMask] = max(
                        new_dp[newMask],
                        dp[mask] + s[i][j]
                    );
                }
            }
        }
        dp.swap(new_dp);
    }

    int fullMask = (1 << p) - 1;
    cout << dp[fullMask] << "\n";

    return 0;
}
```