# Data Structure - Homework 2: Transpose

n is number of row
m is number of col k is number of none-zero elements

## Analysis Traditional matrix

```
void transpose() override {
    vector<vector<int>> transpose(m, vector<int>(n, 0));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            transpose[j][i] = mat[i][j];
        }
    }
    mat = transpose;
    swap(n, m);
}
```

For all element in mat.
Move $mat[i][j]$ to $mat[j][i]$
The complexity is $O(n*m)$

## spare matrix transpose

```
void transpose() override {
    vector<tuple<int, int, int>> transpose;
    for(int i=0;i<=m;++i){
        for(auto& [a, b, c]: mat){
            if(b == i){
                transpose.push_back({b, a, c});
            }
        }
    }
    mat = transpose;
    swap(n, m);
}
```

The total iteration of the inner loop is k.
The outer loop runs m+1 times.
The complexity is O(m*k).

## spare matrix fest transpose

```
    void transpose() override {
        vector<tuple<int, int, int>> transpose(mat.size());
        vector<int> count(m+1, 0);
        for(auto& [a, b, c]: mat){
            count[b]++;
        }
        vector<int> pos(m+1, 0);
        for(int i=1;i<=m;++i){
            pos[i] = pos[i-1] + count[i-1];
        }
        for(auto& [a, b, c]: mat){
            transpose[pos[b]++] = {b, a, c};
        }
        mat = transpose;
        swap(n, m);
    }
```

The number frequency in colum.The complexity is $O(k)$.
Initializing the position The complexity is $O(m)$.
Filling the transposed matrix is also $O(k)$.
Therefore, the overall time complexity of this method is O(k + m).