Day 1 : 24 Feb' 25

Factorial :

C++ :

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

int factorial(int n) {
    if(n == 0) {
        return 1;
    } else if(n == 1) {
        return 1;
    } else {
        return n*factorial(n-1);
    }
}

int main() {
    int n;
    cin >> n;
    cout << factorial(n);
    return 0;
}
```

Python :

```
def factorial(n):
    if n == 0:
        return 1
    elif n == 1:
        return 1
    else:
        return n*factorial(n-1)

n = int(input())
print(factorial(n))
```

Printing 1 to n backwards :

C++ :

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

int backwards(int n) {
    if(n == 1) {
```

```
        return 1;
    } else {
        cout << n << " ";
        return backwards(n-1);
    }
}

int main() {
    int n;
    cin >> n;
    cout << backwards(n);
    return 0;
}
```

Python :

```
def backwards(n):
    if n == 1:
        return 1
    else:
        print(n, end=" ")
        return backwards(n-1)

t = int(input())
while(t > 0):
    n = int(input())
    print(backwards(n))
    t -= 1
```

Towers of Hanoi :

C++ :

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

void TowersOfHanoi(int n, char from_disc, char to_disc, char auxi) {
    if(n == 1) {
        cout << "Move disk 1 from disc " << from_disc << " to " <<
to_disc << endl;
        return;
    }
    TowersOfHanoi(n-1, from_disc, auxi, to_disc);
    cout << "Move disk " << n << " from disc " << from_disc << " to "
<< to_disc << endl;
    TowersOfHanoi(n-1, auxi, to_disc, from_disc);
}
```

```
int main() {
    int n;
    cin >> n;
    TowersOfHanoi(n, 'A', 'B', 'C');
    return 0;
}
```

Python :

```
def TowersOfHanoi(n, from_disc, to_disc, auxi):
    if n == 1:
        print(f"Move disc 1 from disc {from_disc} to {to_disc}")
        return
    TowersOfHanoi(n-1, from_disc, auxi, to_disc)
    print(f"Move disk {n} from disc {from_disc} to {to_disc}")
    TowersOfHanoi(n-1, auxi, to_disc, from_disc)

n = int(input())
TowersOfHanoi(n, 'A', 'B', 'C')
```

Checking whether an array is sorted using recursion :

C++ :

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

int check(vector<int> A, int n) {
    if(n == 1) {
        return 1;
    }
    if(A[n-2] > A[n-1]) {
        return 0;
    } else {
        return check(A, n-1);
    }
}

int main() {
    int n;
    cin >> n;
    vector<int> num(n);
    for(int i = 0; i < n; i++) {
        cin >> num[i];
    }
    if(check(num, n) == 1) {
```

```
        cout << "YES";
    } else {
        cout << "NO";
    }
    return 0;
}
```

Python :

```
def check(arr, n):
    if n <= 1:
        return True
    if arr[n - 2] > arr[n - 1]:
        return False
    return check(arr, n - 1)

n = int(input())
arr = [int(input()) for _ in range(n)]
if check(arr, n):
    print("YES")
else:
    print("NO")
```

All binary strings of length n :

C++ :

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

void generate_binary_strings(int n, string current) {
    if (current.length() == n) {
        cout << current << endl;
        return;
    }
    generate_binary_strings(n, current + "0");
    generate_binary_strings(n, current + "1");
}

int main() {
    int n;
    cin >> n;
    generate_binary_strings(n, "");
    return 0;
}
```

Python :

```
def generate_binary_strings(n, current=""):
    if(len(current) == n):
        print(current)
        return
    generate_binary_strings(n, current + "0")
    generate_binary_strings(n, current + "1")

n = int(input())
generate_binary_strings(n)
```

All strings of length n drawn from 0 … k-1

C++ :

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

void generate_strings(int n, int k, string current) {
    if (current.length() == n) {
        cout << current << endl;
        return;
    }
    for (int i = 0; i < k; i++) {
        generate_strings(n, k, current + to_string(i));
    }
}

int main() {
    int n, k;
    cin >> n >> k;
    generate_strings(n, k, "");
    return 0;
}
```

Python :

```
def generate_strings(n, k, current=""):
    if len(current) == n:
        print(current)
        return
    for i in range(k):
        generate_strings(n, k, current + str(i))

n, k = map(int, input().split())
generate_strings(n, k)
```

Finding the length of connected cells of 1s in a matrix of 0s and 1s :

C++ :

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

int getVal(vector<vector<int>> &A, int i, int j, int L, int H) {
    if (i < 0 || i >= L || j < 0 || j >= H)
        return 0;
    else
        return A[i][j];
}

void findMaxBlock(vector<vector<int>> &A, int r, int c, int L, int H,
int size, vector<vector<bool>> &cntarr, int &maxsize) {
    if (r >= L || r < 0 || c >= H || c < 0)
        return;
    cntarr[r][c] = true;
    size++;
    if (size > maxsize)
        maxsize = size;
    int direction[8][2] = {{1, 0}, {0, -1}, {-1, 0}, {0, 1}, {1, -1},
{-1, -1}, {1, 1}, {-1, 1}};
    for (int i = 0; i < 8; i++) {
        int newi = r + direction[i][0];
        int newj = c + direction[i][1];
        int val = getVal(A, newi, newj, L, H);
        if (val > 0 && cntarr[newi][newj] == false) {
            findMaxBlock(A, newi, newj, L, H, size, cntarr, maxsize);
        }
    }
    cntarr[r][c] = false;
}

int getMaxOnes(vector<vector<int>> &A, int rmax, int colmax) {
    int maxsize = 0;
    int size = 0;
    vector<vector<bool>> cntarr(rmax, vector<bool>(colmax, false));
    for (int i = 0; i < rmax; i++) {
        for (int j = 0; j < colmax; j++) {
            if (A[i][j] == 1) {
                findMaxBlock(A, i, j, rmax, colmax, 0, cntarr,
maxsize);
```

```
            }
        }
    }
    return maxsize;
}

int main() {
    int rmax, colmax;
    cin >> rmax >> colmax;
    vector<vector<int>> A(rmax, vector<int>(colmax));
    for (int i = 0; i < rmax; i++) {
        for (int j = 0; j < colmax; j++) {
            cin >> A[i][j];
        }
    }
    cout << getMaxOnes(A, rmax, colmax) << endl;
    return 0;
}
```

Python :

```
def get_val(A, i, j, L, H):
    if i < 0 or i >= L or j < 0 or j >= H:
        return 0
    else:
        return A[i][j]

def find_max_block(A, r, c, L, H, size, cntarr, maxsize):
    if r >= L or r < 0 or c >= H or c < 0:
        return
    cntarr[r][c] = True
    size += 1
    if size > maxsize[0]:
        maxsize[0] = size
    direction = [(1, 0), (0, -1), (-1, 0), (0, 1), (1, -1), (-1, -1),
(1, 1), (-1, 1)]
    for i in range(8):
        newi = r + direction[i][0]
        newj = c + direction[i][1]
        val = get_val(A, newi, newj, L, H)
        if val > 0 and not cntarr[newi][newj]:
            find_max_block(A, newi, newj, L, H, size, cntarr,
maxsize)
    cntarr[r][c] = False
```

```
def get_max_ones(A, rmax, colmax):
    maxsize = [0]
    size = 0
    cntarr = [[False for _ in range(colmax)] for _ in range(rmax)]
    for i in range(rmax):
        for j in range(colmax):
            if A[i][j] == 1:
                find_max_block(A, i, j, rmax, colmax, 0, cntarr,
maxsize)
    return maxsize[0]

rmax, colmax = map(int, input().split())
A = [list(map(int, input().split())) for _ in range(rmax)]
print(get_max_ones(A, rmax, colmax))
```