**To Compile:**

to compile: g++ -o exe test.cpp

please copy the code to test.cpp, then run g++ command

**Steps to submit:**

calculate complexity of all functions and submit the results in GitHub (your Github)

**Deadline:** Tuesday, Sep 7th, 23:59

#include <stdio.h>

#include <stdlib.h>

#include <cassert>

#include <time.h>

int\*\* create\_matrix(int n)

{

int\*\* m = new int\*[n];

for (int i = 0; i < n; ++i) {

m[i] = new int[n];

}

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

m[i][j] = 0;

}

}

return m;

}

As there is a “for” loop with another one inside the complexity would be O(n^2)

void delete\_matrix(int\*\* m, int n)

{

for (int i = 0; i < n; ++i) {

delete [] m[i];

}

delete [] m;

}

As there is a “for” loop the complexity would be O(n)

void fill\_matrix\_with\_random\_elements(int\*\* m, int n)

{

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

int r = rand() % 2;

m[i][j] = r;

}

}

}

As there is a “for” loop with another one inside the complexity would be O(n^2)

void print\_matrix(int\*\* m, int n)

{

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

printf("%d ", m[i][j]);

}

printf("\n");

}

}

As there is a “for” loop with another one inside the complexity would be O(n^2)

bool check\_all\_elements\_in\_row\_are\_zero(int\*\* a, int n, int i)

{

assert(i >= 0);

assert(i < n);

for (int j = 0; j < n; ++j) {

if (a[i][j] != 0) {

return false;

}

}

return true;

}

As there is a “for” loop the complexity would be O(n)

void check\_for\_zero(int\*\* m, int n)

{

for (int i = 0; i < n; ++i) {

bool b = check\_all\_elements\_in\_row\_are\_zero(m, n, i);

bool x = check\_all\_elements\_in\_column\_are\_zero(m, n, i);

if (b) {

printf("all elements in row %d are zero\n", i);

break;

}

if (x) {

printf("all elements in column %d are zero\n", i);

break;

}

}

}

As there is a “for” loop the complexity would be O(n)

int main()

{

int n = 0;

// Use current time as seed for random generator

srand(time(0));

printf("Enter array size: ");

scanf("%d", &n);

if (n <= 0) {

printf("Invalid input, exiting...\n");

return EXIT\_FAILURE;

}

int\*\* matrix = create\_matrix(n);

fill\_matrix\_with\_random\_elements(matrix, n);

print\_matrix(matrix, n);

check\_for\_zero(matrix, n);

return EXIT\_SUCCESS;

}

The complexity would be 11, as there are only actions that are counted as 1.