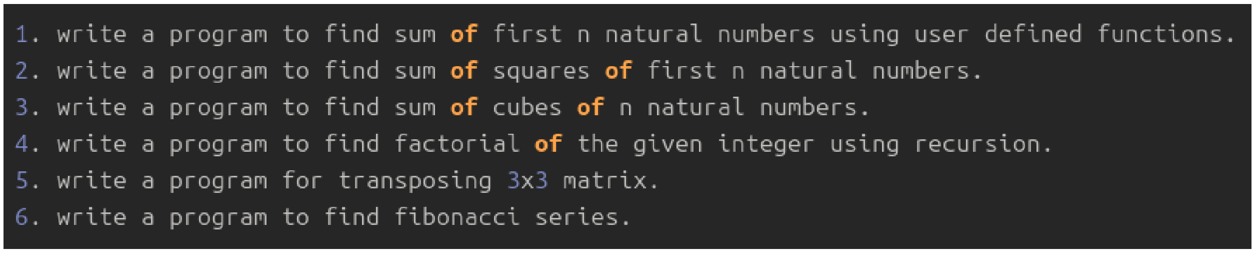
Name : Chepoori Sai Vivek

Roll no : CH.SC.U4CSE24108

Subject : DAA Lab -1 Experiment

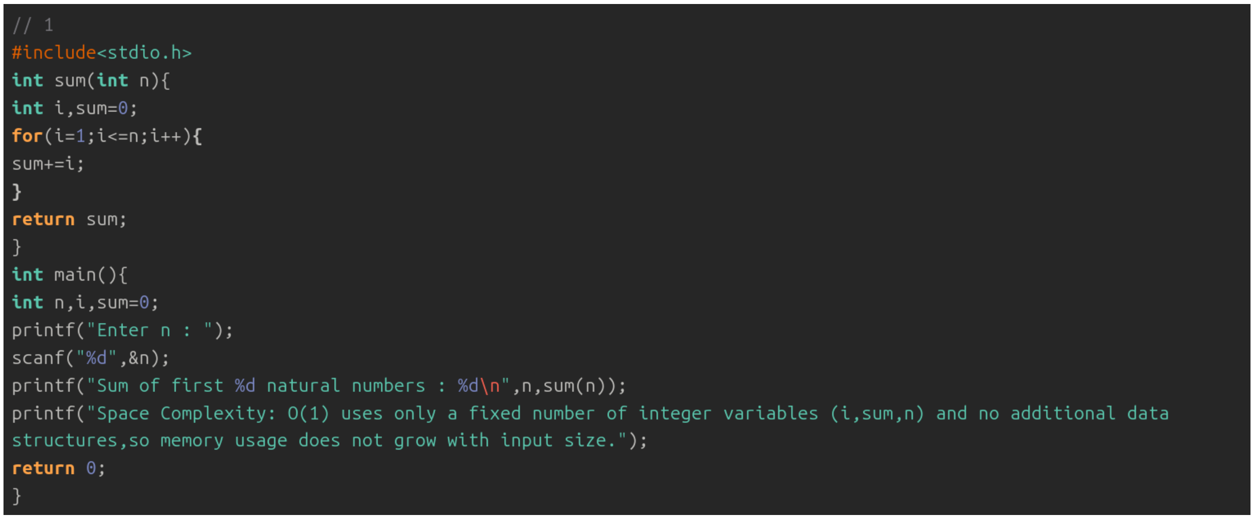
Date : 27th November 2025

Questions :

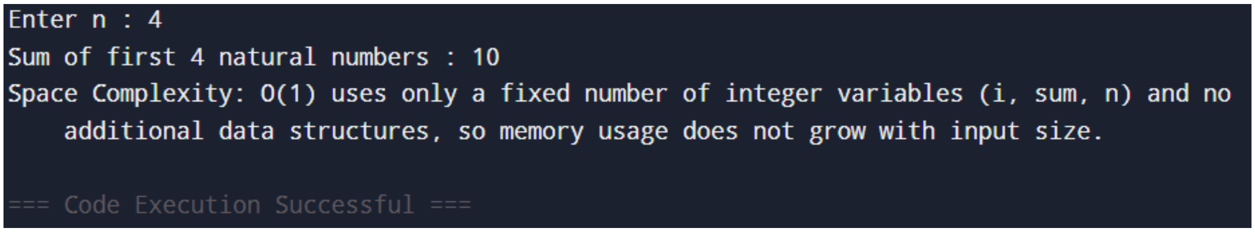


```
1. write a program to find sum of first n natural numbers using user defined functions.
2. write a program to find sum of squares of first n natural numbers.
3. write a program to find sum of cubes of n natural numbers.
4. write a program to find factorial of the given integer using recursion.
5. write a program for transposing 3x3 matrix.
6. write a program to find fibonacci series.
```

Solutions :



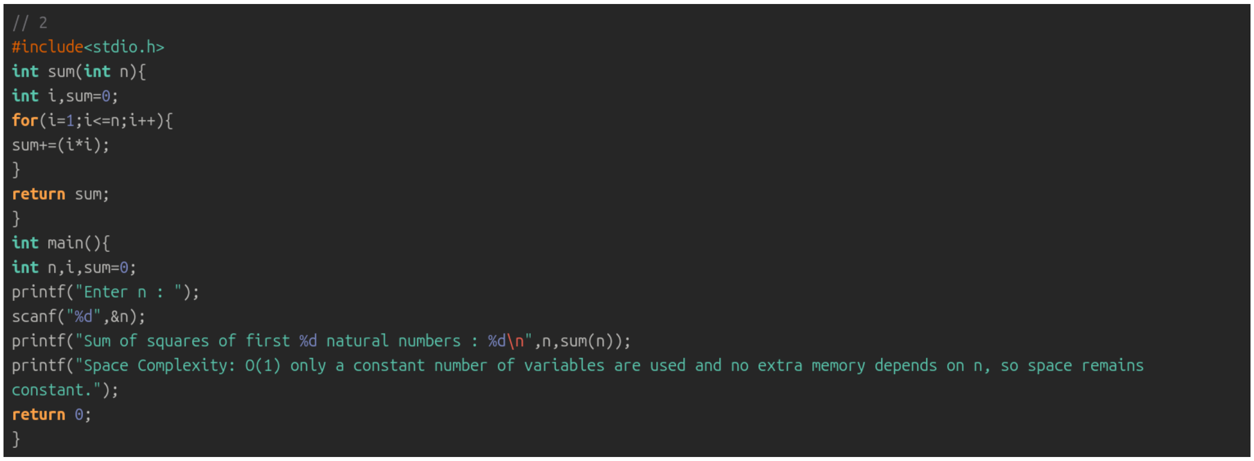
```c
// 1
#include<stdio.h>
int sum(int n){
int i,sum=0;
for(i=1;i<=n;i++){
sum+=i;
}
return sum;
}
int main(){
int n,i,sum=0;
printf("Enter n : ");
scanf("%d",&n);
printf("Sum of first %d natural numbers : %d\n",n,sum(n));
printf("Space Complexity: O(1) uses only a fixed number of integer variables (i,sum,n) and no additional data
structures,so memory usage does not grow with input size.");
return 0;
}
```



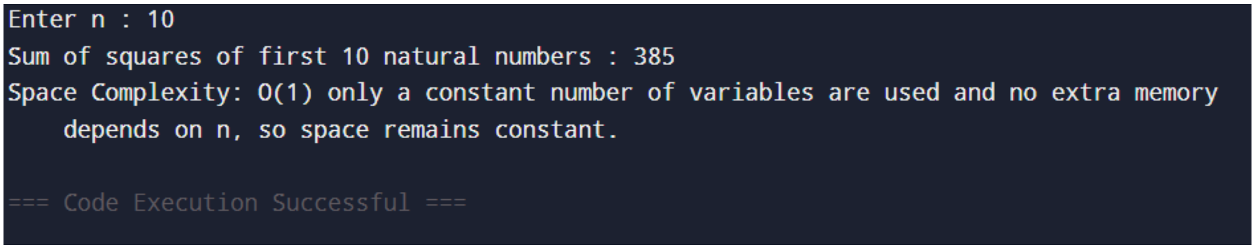
```
Enter n : 4
Sum of first 4 natural numbers : 10
Space Complexity: O(1) uses only a fixed number of integer variables (i, sum, n) and no
    additional data structures, so memory usage does not grow with input size.

=== Code Execution Successful ===
```

Space Complexity: O(1) uses only a fixed number of integer variables (i, sum, n) and no additional data structures, so memory usage does not grow with input size.



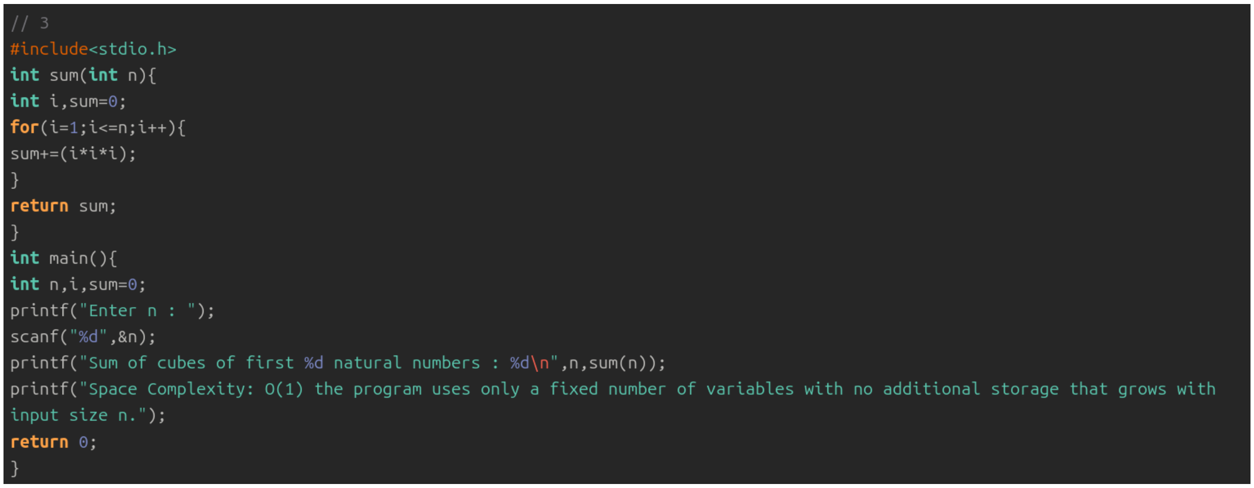
```c
// 2
#include<stdio.h>
int sum(int n){
int i,sum=0;
for(i=1;i<=n;i++){
sum+=(i*i);
}
return sum;
}
int main(){
int n,i,sum=0;
printf("Enter n : ");
scanf("%d",&n);
printf("Sum of squares of first %d natural numbers : %d\n",n,sum(n));
printf("Space Complexity: O(1) only a constant number of variables are used and no extra memory depends on n, so space remains
constant.");
return 0;
}
```



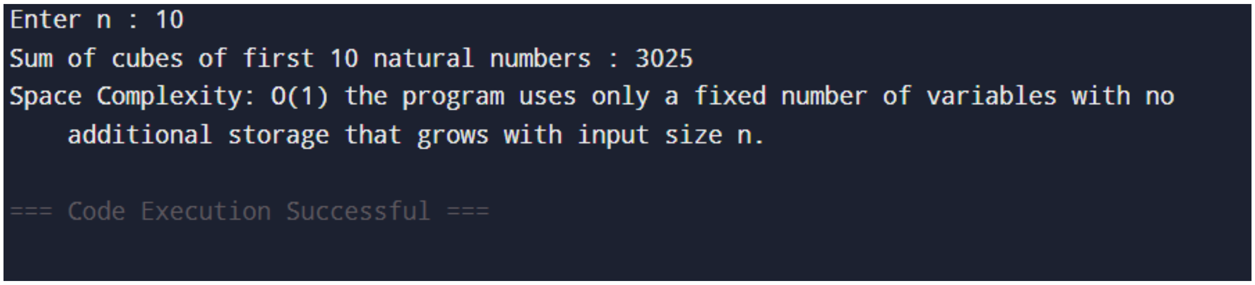
```
Enter n : 10
Sum of squares of first 10 natural numbers : 385
Space Complexity: O(1) only a constant number of variables are used and no extra memory
    depends on n, so space remains constant.

=== Code Execution Successful ===
```

Space Complexity: O(1) only a constant number of variables are used and no extra memory depends on n, so space remains constant.



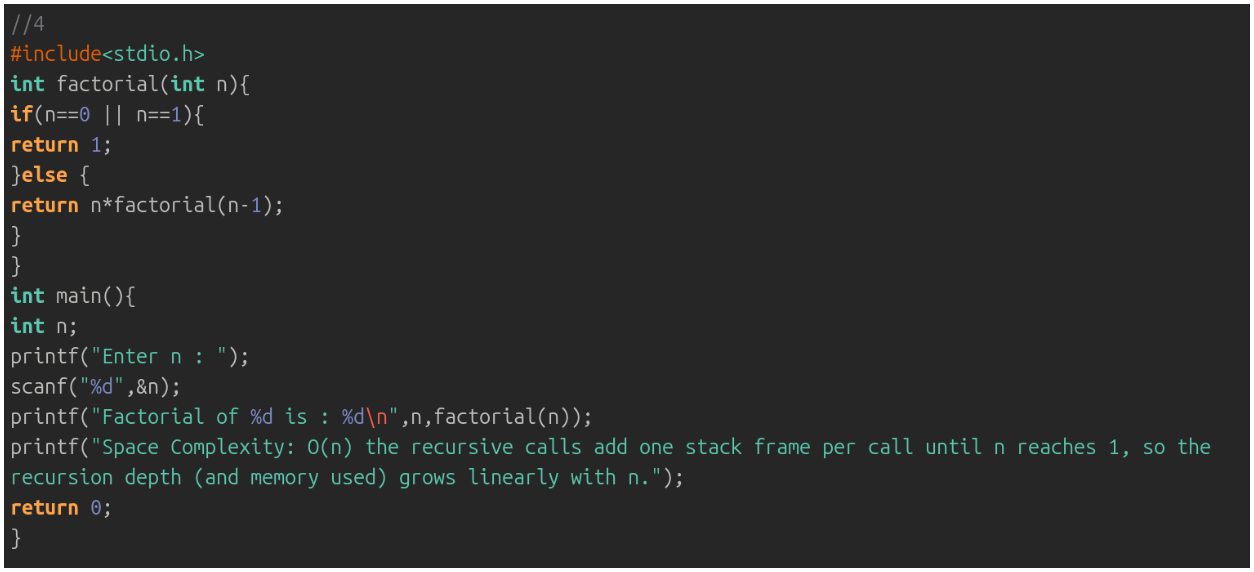
```c
// 3
#include<stdio.h>
int sum(int n){
int i,sum=0;
for(i=1;i<=n;i++){
sum+=(i*i*i);
}
return sum;
}
int main(){
int n,i,sum=0;
printf("Enter n : ");
scanf("%d",&n);
printf("Sum of cubes of first %d natural numbers : %d\n",n,sum(n));
printf("Space Complexity: O(1) the program uses only a fixed number of variables with no additional storage that grows with 
input size n.");
return 0;
}
```



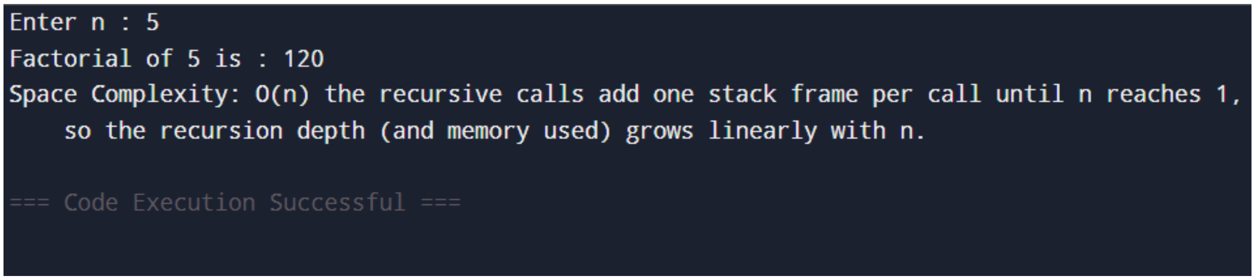
```
Enter n : 10
Sum of cubes of first 10 natural numbers : 3025
Space Complexity: O(1) the program uses only a fixed number of variables with no
    additional storage that grows with input size n.

=== Code Execution Successful ===
```

Space Complexity: O(1) the program uses only a fixed number of variables with no additional storage that grows with input size n.



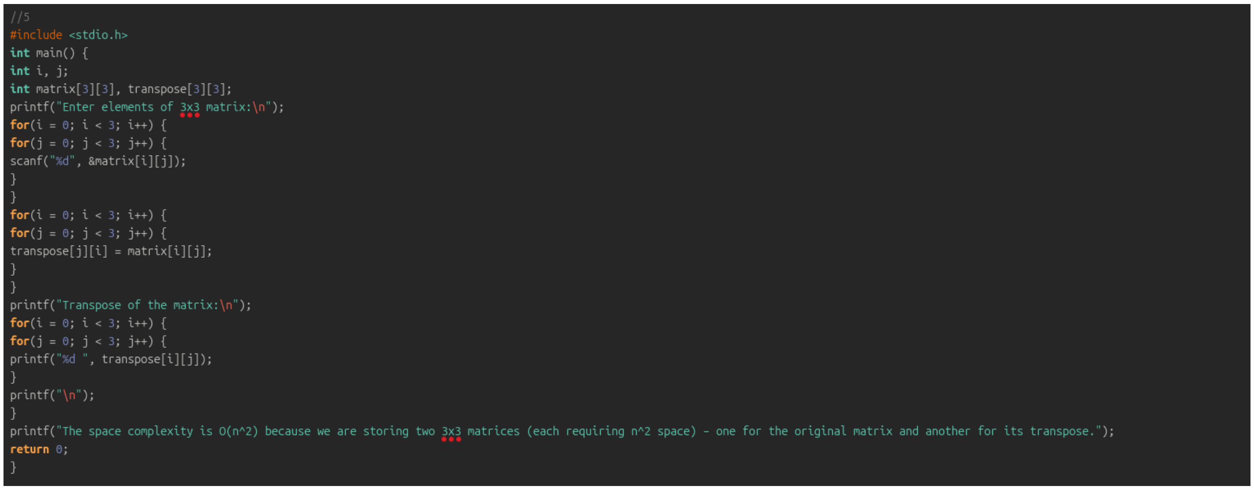
```c
//4
#include<stdio.h>
int factorial(int n){
if(n==0 || n==1){
return 1;
}else {
return n*factorial(n-1);
}
}
int main(){
int n;
printf("Enter n : ");
scanf("%d",&n);
printf("Factorial of %d is : %d\n",n,factorial(n));
printf("Space Complexity: O(n) the recursive calls add one stack frame per call until n reaches 1, so the recursion depth (and memory used) grows linearly with n.");
return 0;
}
```



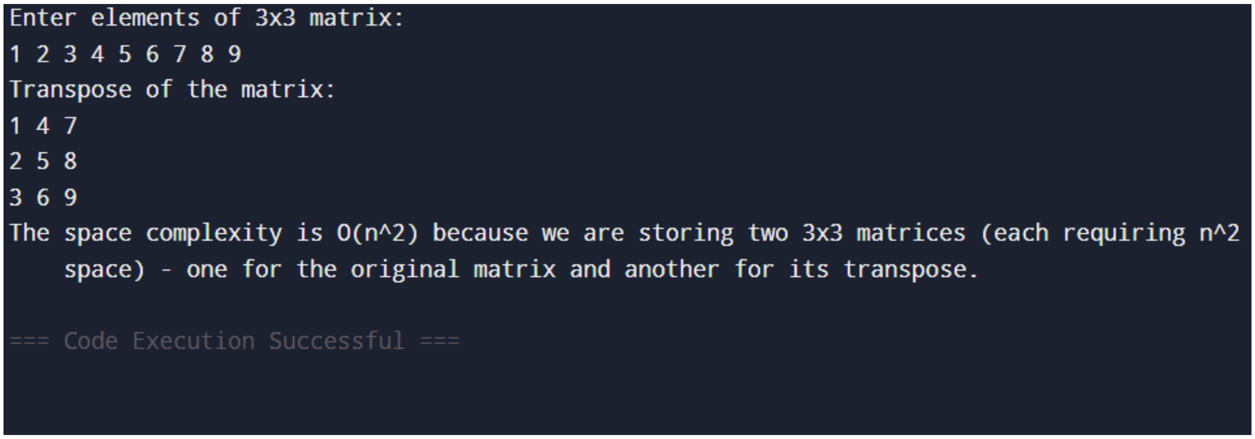
```
Enter n : 5
Factorial of 5 is : 120
Space Complexity: O(n) the recursive calls add one stack frame per call until n reaches 1,
    so the recursion depth (and memory used) grows linearly with n.

=== Code Execution Successful ===
```

Space Complexity: O(n) the recursive calls add one stack frame per call until n reaches 1, so the recursion depth (and memory used) grows linearly with n.



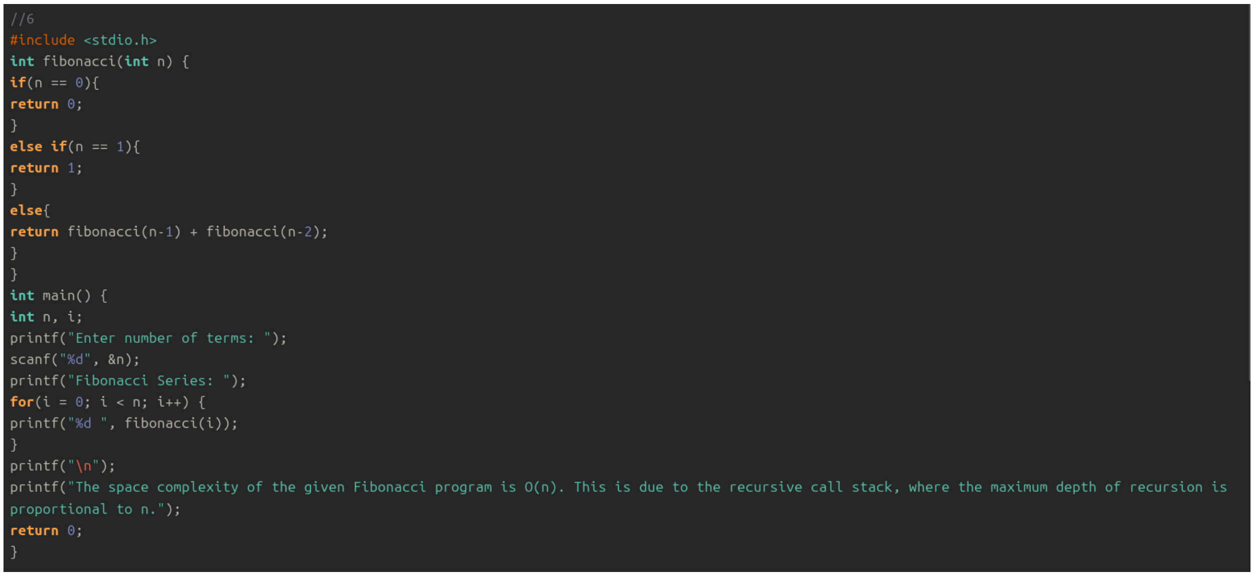
```c
//5
#include <stdio.h>
int main() {
int i, j;
int matrix[3][3], transpose[3][3];
printf("Enter elements of 3x3 matrix:\n");
for(i = 0; i < 3; i++) {
for(j = 0; j < 3; j++) {
scanf("%d", &matrix[i][j]);
}
}
for(i = 0; i < 3; i++) {
for(j = 0; j < 3; j++) {
transpose[j][i] = matrix[i][j];
}
}
printf("Transpose of the matrix:\n");
for(i = 0; i < 3; i++) {
for(j = 0; j < 3; j++) {
printf("%d ", transpose[i][j]);
}
printf("\n");
}
printf("The space complexity is O(n^2) because we are storing two 3x3 matrices (each requiring n^2 space) - one for the original matrix and another for its transpose.");
return 0;
}
```



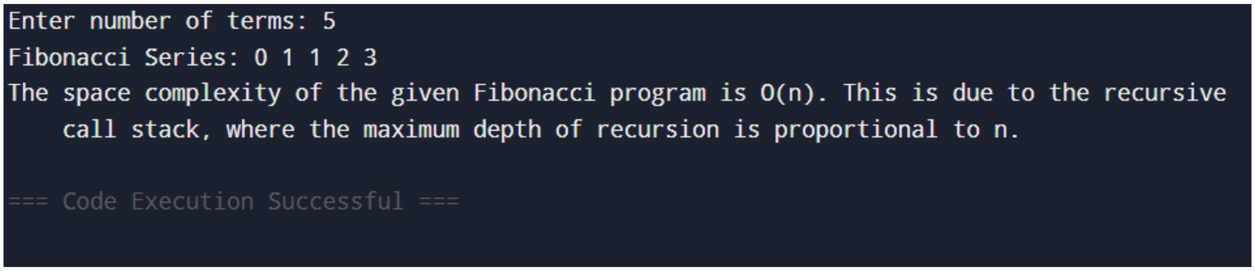
```
Enter elements of 3x3 matrix:
1 2 3 4 5 6 7 8 9
Transpose of the matrix:
1 4 7
2 5 8
3 6 9
The space complexity is O(n^2) because we are storing two 3x3 matrices (each requiring n^2
    space) - one for the original matrix and another for its transpose.

=== Code Execution Successful ===
```

The space complexity is O(n^2) because we are storing two 3x3 matrices (each requiring n^2 space) - one for the original matrix and another for its transpose.



```
//6
#include <stdio.h>
int fibonacci(int n) {
if(n == 0){
return 0;
}
else if(n == 1){
return 1;
}
else{
return fibonacci(n-1) + fibonacci(n-2);
}
}
int main() {
int n, i;
printf("Enter number of terms: ");
scanf("%d", &n);
printf("Fibonacci Series: ");
for(i = 0; i < n; i++) {
printf("%d ", fibonacci(i));
}
printf("\n");
printf("The space complexity of the given Fibonacci program is O(n). This is due to the recursive call stack, where the maximum depth of recursion is
proportional to n.");
return 0;
}
```



```
Enter number of terms: 5
Fibonacci Series: 0 1 1 2 3
The space complexity of the given Fibonacci program is O(n). This is due to the recursive
    call stack, where the maximum depth of recursion is proportional to n.

=== Code Execution Successful ===
```

The space complexity of the given Fibonacci program is O(n). This is due to the recursive call stack, where the maximum depth of recursion is proportional to n.