Amrita School of Computing

23CSE211

Design and Analysis of Algorithms

WEEK – 1 (27 November 2025)

Susendran M

CH.SC.U4CSE24154

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

AMRITA SCHOOL OF COMPUTING

CHENNAI

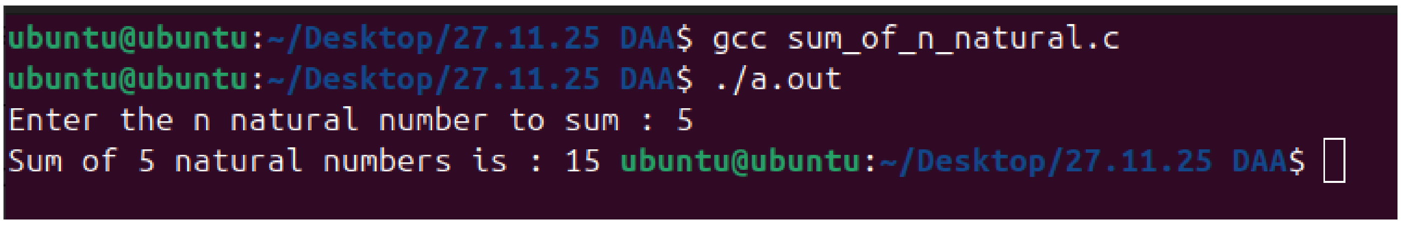
1. Write a program to find the sum of first N natural numbers.

Code :

```c
#include <stdio.h>
int sum_of_n_natural (int n)
{
int sum=0;
for(int i=1;i<=n;i++)
{
sum=sum+i;
}
return sum;
}
int main()
{
int s,k;
printf("Enter the n natural number to sum : ");
scanf("%d",&s);
k=sum_of_n_natural(s);
printf("Sum of %d natural numbers is : %d ",s,k);
return 0;
}

// Fixed part = 20 bytes
// Variable part = 0 bytes
// Total Space complexity = 20 bytes
// Space complexity = O(1)
/* Since the loop does not allocate new memory, and constant number of
variables is used and not changing based on input */
```

Output :



```
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ gcc sum_of_n_natural.c
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ ./a.out
Enter the n natural number to sum : 5
Sum of 5 natural numbers is : 15 ubuntu@ubuntu:~/Desktop/27.11.25 DAA$
```

Justification :

Since the loop does not allocate new memory, and constant number of variables is used and not changing based on input and Space complexity is O(1).

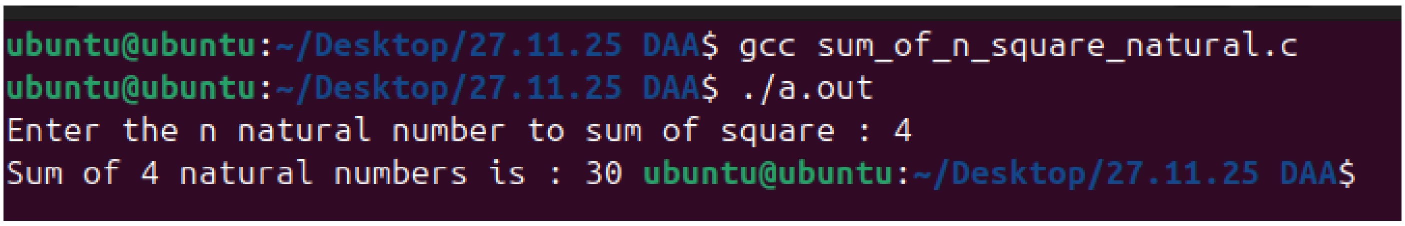
2. Write a program to find the sum of square of first N natural numbers.

Code :

```c
#include <stdio.h>
int main()
{
int n;
printf("Enter the n natural number to sum of square : ");
scanf("%d",&n);
int sum=0;
for(int i=1;i<=n;i++)
{
sum=sum+(i*i);
}
printf("Sum of %d natural numbers is : %d ",n,sum);
return 0;
}

// Fixed part = 12 bytes
// Variable part = 0 bytes
// Total Space complexity = 12 bytes
// Space complexity = O(1)
/* Since the loop does not allocate new memory, and constant number of
variables is used and not changing based on input */
```

Output :



```
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ gcc sum_of_n_square_natural.c
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ ./a.out
Enter the n natural number to sum of square : 4
Sum of 4 natural numbers is : 30 ubuntu@ubuntu:~/Desktop/27.11.25 DAA$
```

Justification :

Since the loop does not allocate new memory, and constant number of variables is used and not changing based on input and Space complexity is O(1).

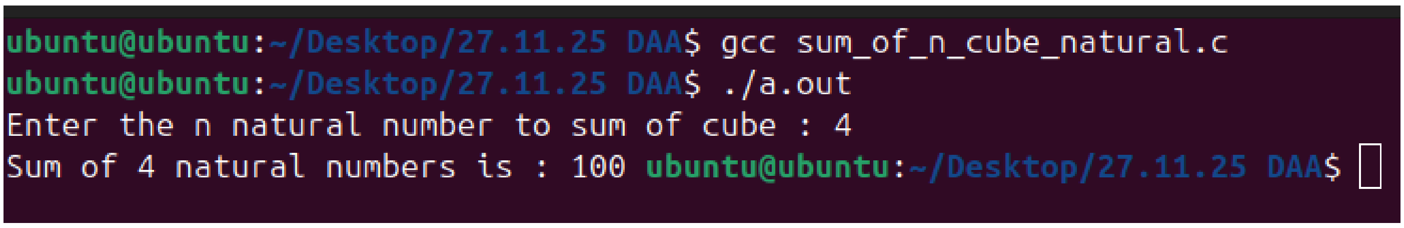
3. Write a program to find the sum of cube of first N natural numbers.

Code :

```
#include <stdio.h>
int main()
{
int n;
printf("Enter the n natural number to sum of cube : ");
scanf("%d",&n);
int sum=0;
for(int i=1;i<=n;i++)
{
sum=sum+(i*i*i);
}
printf("Sum of %d natural numbers is : %d ",n,sum);
return 0;
}

// Fixed part = 12 bytes
// Variable part = 0 bytes
// Total Space complexity = 12 bytes
// Space complexity = O(1)
/* Since the loop does not allocate new memory, and constant number of
variables is used and not changing based on input */
```

Output :



```
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ gcc sum_of_n_cube_natural.c
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ ./a.out
Enter the n natural number to sum of cube : 4
Sum of 4 natural numbers is : 100 ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ 
```

Justification :

Since the loop does not allocate new memory, and constant number of variables is used and not changing based on input and Space complexity is O(1).

4. **Write a program to find factorial of a given number using recursion.**

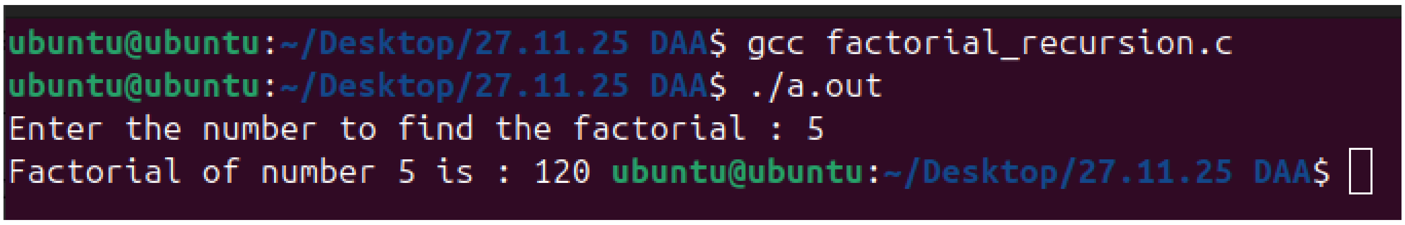
Code :

```
#include <stdio.h>
int factorial_of_number (int n)
{
if(n==0)
{
return 1;
}
else
{
return n * factorial_of_number (n-1);
}
}

int main()
{
int s,k;
printf("Enter the number to find the factorial : ");
scanf("%d",&s);
k=factorial_of_number(s);
printf("Factorial of number %d is : %d ",s,k);
return 0;
}

// Fixed part = 8 bytes
// Variable part = 4*n bytes
// Total Space complexity = 8 + 4*n bytes
// Space complexity = O(n)
/* Since the n increases, the variable part 4*n gives the actual space
complexity and the function factorial_of_number is recursive. So, memory
increases with changes in input size */
```

Output :



```
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ gcc factorial_recursion.c
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ ./a.out
Enter the number to find the factorial : 5
Factorial of number 5 is : 120 ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ 
```

Justification :

Since the n increases, the variable part 4*n gives the actual space complexity and the function factorial_of_number is recursive. So, memory increases with changes in input size and Space complexity is O(n).

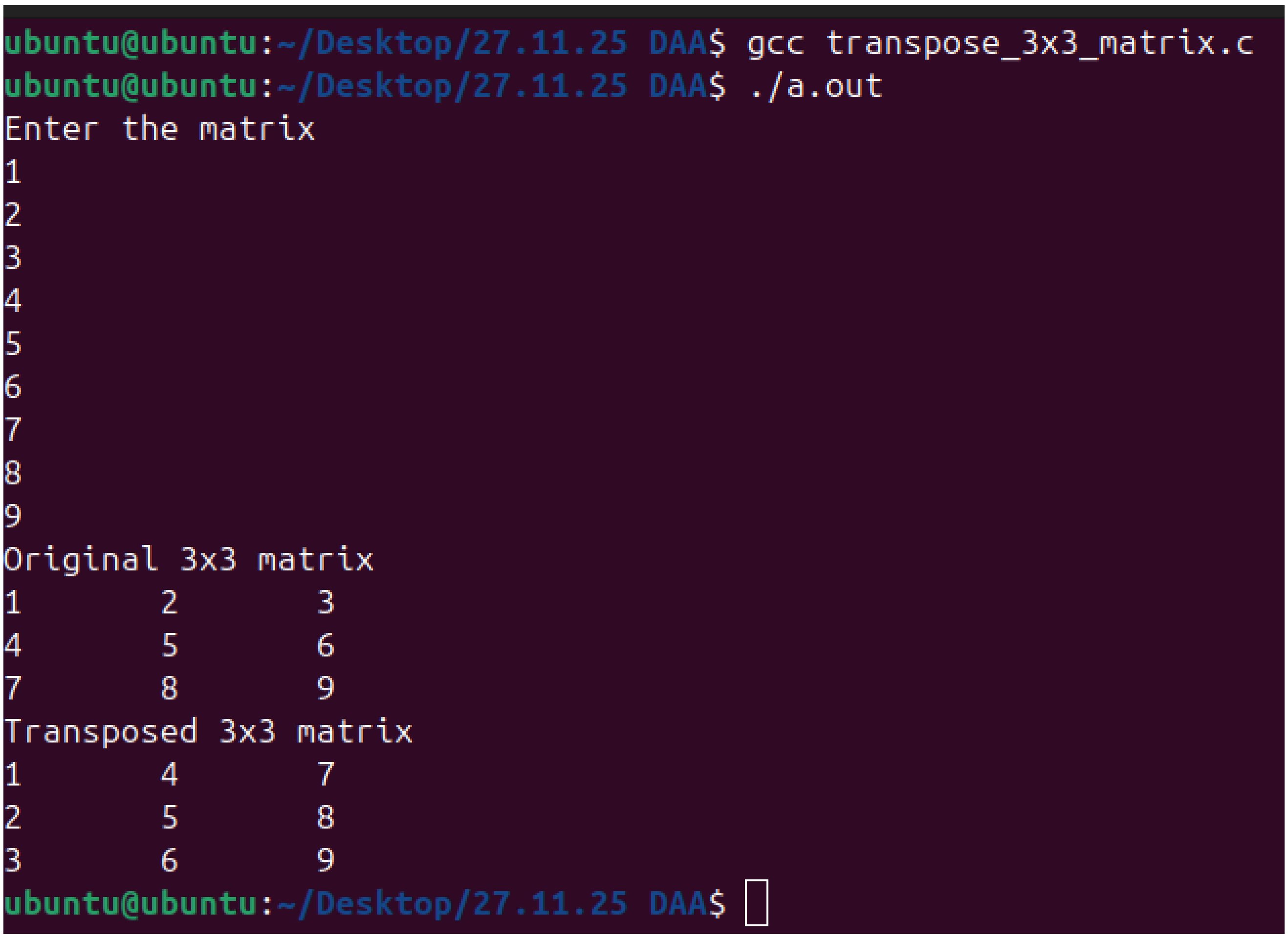
5. **Write a program to find transpose of a 3x3 matrix.**

Code :

```c
#include <stdio.h>
int main()
{
int arr[3][3];
printf("Enter the matrix \n");
for(int i=0;i<3;i++)
{ for(int j=0;j<3;j++) {
scanf("%d",&arr[i][j]);
}}
printf("Original 3x3 matrix \n");
for(int i=0;i<3;i++)
{ for(int j=0;j<3;j++) {
printf("%d \t",arr[i][j]);
} printf("\n");}
printf("Transposed 3x3 matrix \n");
for(int i=0;i<3;i++)
{ for(int j=0;j<3;j++) {
printf("%d \t",arr[j][i]);
} printf("\n"); }
return 0;
}

// Fixed part = 44 bytes
// Variable part = 0 bytes
// Total Space complexity = 44 bytes
// Space complexity = O(1)
/* Since the arr is fixed size and loop does not allocate new memory, and constant
number of variables is used and not changing based on input, Suppose the matrix dimension
are not fixed and like arr[m][n] then the space complexity = O(m*n) because the variable
part = 4*m*n as m and n changes with the input size */
```

Output :



```
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ gcc transpose_3x3_matrix.c
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ ./a.out
Enter the matrix
1
2
3
4
5
6
7
8
9
Original 3x3 matrix
1       2       3
4       5       6
7       8       9
Transposed 3x3 matrix
1       4       7
2       5       8
3       6       9
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$
```

Justification :

Since the arr is fixed size and loop does not allocate new memory, and constant number of variables is used and not changing based on inputSuppose the matrix dimension are not fixed and like arr[m][n] then the space complexity = O(m*n) because the variable part = 4*m*n as m and n changes with the input size and Space complexity is O(1).

6. **Write a program to find Fibonacci series upto first N terms.**

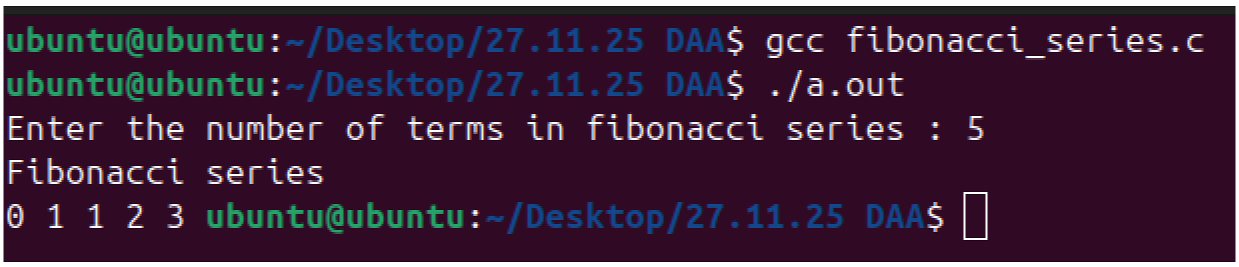
Code :

```c
#include <stdio.h>
int fibonacci_series (int n)
{
if(n==0) {
return 0; }
if(n==1) {
return 1; }
else {
return fibonacci_series (n-1) + fibonacci_series (n-2); }
}

int main()
{
int s,k;
printf("Enter the number of terms in fibonacci series : ");
scanf("%d",&s);
printf("Fibonacci series \n");
for(int i=0;i<s;i++) {
k=fibonacci_series(i);
printf("%d ",k); }
return 0;
}

// Fixed part = 12 bytes
// Variable part = 4*n bytes
// Total Space complexity = 12 + 4*n bytes
// Space complexity = O(n)
/* Since the n increases, the variable part 4*n gives the actual space
complexity and the function fibonacci_series is recursive. So, memory
increases with changes in input size */
```

Output :



```
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ gcc fibonacci_series.c
ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ ./a.out
Enter the number of terms in fibonacci series : 5
Fibonacci series
0 1 1 2 3 ubuntu@ubuntu:~/Desktop/27.11.25 DAA$ 
```

Justification :

Since the n increases, the variable part 4*n gives the actual space complexity and the function fibonacci_series is recursive. So, memory increases with changes in input size and Space complexity is O(n).