NAME: CH. SAI PURNISHA MAHI

ROLL NO: CH.SC.U4CSE24156

(Design and Analysis of Algorithms)

1. **write program to find sum on n natural numbers using user define function**

CODE:

```c
#include <stdio.h>

int sumOfNaturals(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i;
    }
    return sum;
}

int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("Sum of first %d natural numbers is: %d\n", n, sumOfNaturals(n));
    return 0;
}
```

OUTPUT:

```
amma@amma:~$ gcc -o sumOfNaturals sumOfNaturals.c
amma@amma:~$ ./sumOfNaturals
Enter a number: 5
Sum of first 5 natural numbers is: 15
```

Space Complexity: O(1)

Justification:

Variables used:

sum → int → 4 bytes

i → int → 4 bytes

n → int → 4 bytes

Total memory used by variables: 4 + 4 + 4 = 12 bytes

Only 3 integers are used.

No arrays or recursion, so memory does not depend on n.

2. write a program to find sum of squares of first n natural numbers

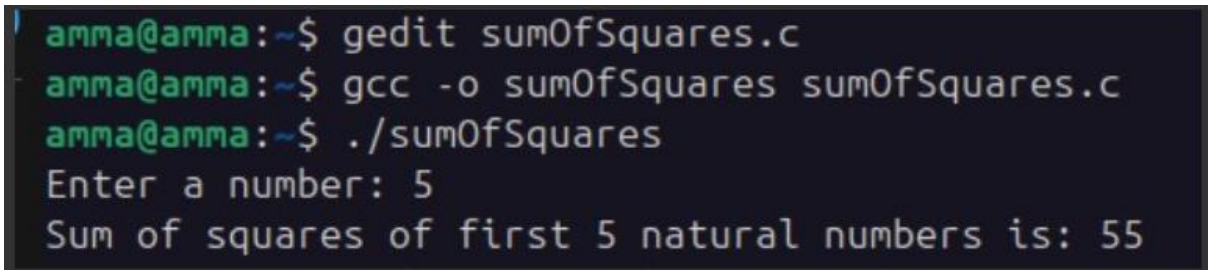
CODE:

```c
#include <stdio.h>

int sumOfSquares(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i * i;
    }
    return sum;
}

int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("Sum of squares of first %d natural numbers is: %d\n", n, sumOfSquares(n));
    return 0;
}
```

OUTPUT:



Space complexity: O(1) (constant space).

Justification:

Variables used:

sum → int → 4 bytes

i → int → 4 bytes

n → int → 4 bytes

Total memory used by variables: 12 bytes

No arrays or recursion, so memory does not depend on n.

3. write a program sum of cubes of first n natural numbers

CODE:

```c
#include <stdio.h>

int sumOfCubes(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        sum += i * i * i;
    }
    return sum;
}

int main() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    printf("Sum of cubes of first %d natural numbers is: %d\n", n, sumOfCubes(n));
    return 0;
}
```

OUTPUT:

```
amma@amma:~$ gedit sumOfCubes.c
amma@amma:~$ gcc -o sumOfCubes sumOfCubes.c
amma@amma:~$ ./sumOfCubes
Enter a number: 5
Sum of cubes of first 5 natural numbers is: 225
```

Space complexity: O(1) (constant space).

Justification:

Variables used:

sum → int → 4 bytes

i → int → 4 bytes

n → int → 4 bytes

Total memory used by variables: 12 bytes

No arrays or recursion, so memory does not depend on n.

4. write a program to find factorial of a given integer using recursion

CODE:

```c
#include <stdio.h>

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

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    if (num < 0)
        printf("Factorial is not defined for negative numbers.\n");
    else
        printf("Factorial of %d is %d\n", num, factorial(num));

    return 0;
}
```

OUTPUT:

```
amma@amma:~$ gedit factorial.c
amma@amma:~$ gcc -o factorial factorial.c
amma@amma:~$ ./factorial
Enter a number: 5
Factorial of 5 is 120
```

Space complexity: O(n) (constant space).

Justification:

Variables used in main:

num → int → 4 bytes

n → int → 4 bytes

Since recursion depth is num (for factorial(num)), total stack memory used by recursion:

(4 + 4) * num = 8 × num bytes

Total =8 × num + 4 bytes

5. write a program for transposing a 3*3 matrices

CODE:

```c
#include <stdio.h>

int main() {
    int matrix[3][3], transpose[3][3];

    printf("Enter elements of 3x3 matrix:\n");
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }
printf("\nTranspose of the matrix:\n");
    for(int i = 0; i < 3; i++) {
        for(int j = 0; j < 3; j++) {
            printf("%d ", transpose[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

OUTPUT:

```
amma@amma:~$ gedit transpose.c
amma@amma:~$ gcc -o transpose transpose.c
amma@amma:~$ ./transpose
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
```

Space complexity: O(n) (constant space).

Justification:

Variables used:

matrix[3][3] → 3×3 integers → 9 integers × 4 bytes = 36 bytes

transpose[3][3] → 3×3 integers → 9 integers × 4 bytes = 36 bytes

Loop counters i and j → 2 integers × 4 bytes = 8 bytes

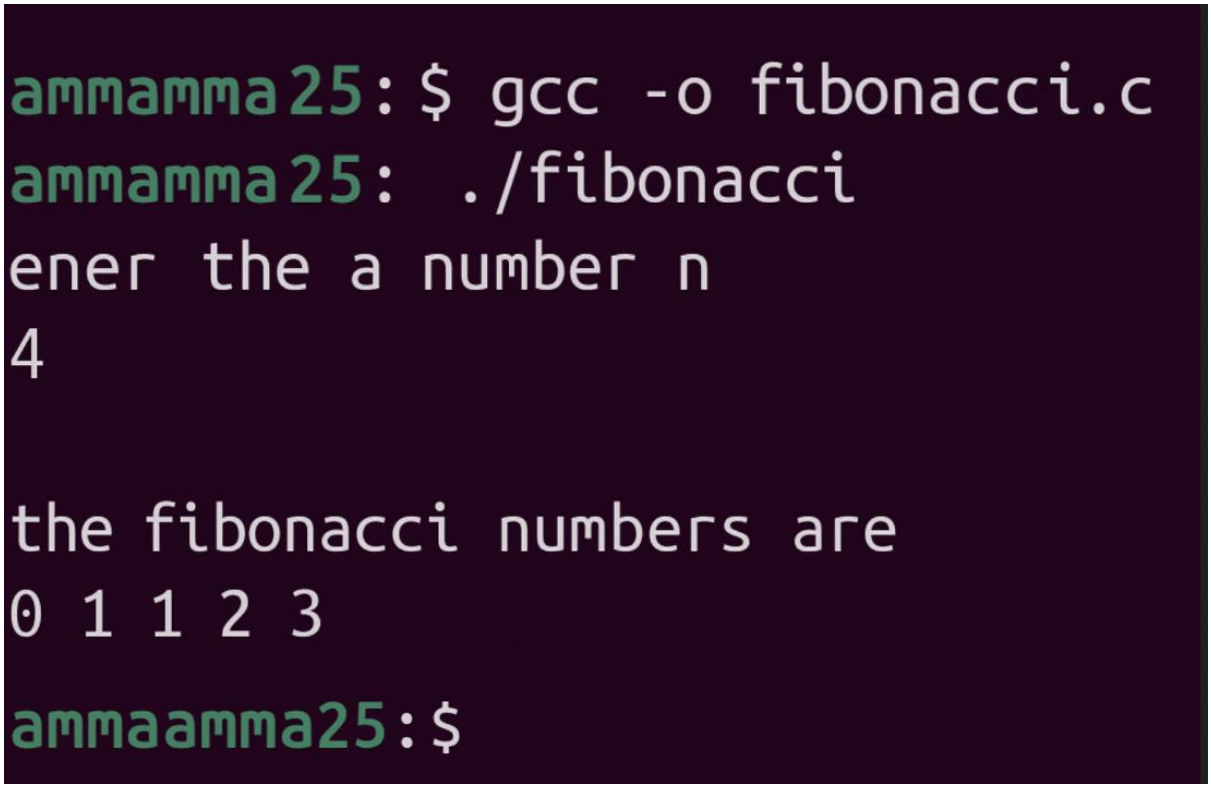
Total memory used by variables: 36 + 36 + 8 = 80 bytes

6. write a program to find fibonacci series

CODE:

```c
#include <stdio.h>
int main(){
int a,b,c,n;
printf("enter the a number n\n");
scanf("%d",&n);
a=0;
b=1;
printf("the fibonacci numbers are \n");
while(a<n){
printf("%d ",a);
 c=a+b;
 a=b;
 b=c;
 }
 }
```

OUTPUT:



Space complexity: O(n) (constant space).

Justification:

Variables used:

a → int → 4 bytes

b → int → 4 bytes

c → int → 4 bytes

n → int → 4 bytes

Total memory used by variables: 4 + 4 + 4 + 4 = 16 bytes

Only 4 integer variables are used.

No recursion, no arrays, no dynamic memory allocation.