DAA SPACE COMPLEXITY PRACTICE PROBLEMS

1. Write a program to find sum of first n natural numbers using user defined function.

Code:

```
#include<stdio.h>
int main ()
{
int s=0;
int n;
scanf("%d",&n);
for(int i=1; i<=n;i++)
{
  s+=i;
}
printf("sum:%d",s);
}
```

Output:

```
square :5\amma@amma14:~/Documents$ gcc -o sum sum.c
amma@amma14:~/Documents$ ./sum
2
sum :3amma@amma14:~/Documents$ gcc -o 4 4.c
```

Space complexity: The space complexity of this program is O(1)

REASON: constant space. Because the program uses a **fixed number of variables**, independent of the input size, its space usage does not grow.

2. Write a program to find the Sum of Square of first N natural numbers.

Code:

```
#include<stdio.h>
int main()
{
int s=0;
int n;
scanf("%d",&n);
for (int i=1; i<=n;i++)
{
 s+=i*i;
}
printf("square: %d",s);
}
```

Output:

```
cube :441amma@amma14:~/Documents$ gcc -o square square.c
amma@amma14:~/Documents$ ./square
2
square :5\amma@amma14:~/Documents$ gcc -o sum sum.c
```

Space complexity: The space complexity of this program is O(1)

REASON: constant space . Because there is no recursion, no arrays , no dynamic memory allocation , the amount of memory used **does not depend on n**.

3. Write a program to find Sum of Cubes of first N natural numbers.

Code:

```
#include<stdio.h>
int main()
{
int s=0;
int n;
scanf("%d",&n);
for(int i=1;i<=n;i++)
{
 s+=i*i*i;
}
printf("cube :%d",s);
}
```

Output:

```
amma@amma14:~/Documents$ gcc -o cube cube.c
amma@amma14:~/Documents$ ./cube
6
cube :441amma@amma14:~/Documents$ gcc -o square square.c
```

Space complexity: The space complexity of this program is O(1)

REASON: constant space . Because there is no recursion, no arrays , no dynamic memory allocation , the amount of memory used **does not depend on n**.

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

Code:

```
#include <stdio.h>

int fact(int n) {
 if(n==1) {
  return 1;
 }
 return n*fact(n-1);
}

void main (){
   int n=0;
   scanf("%d",&n);
   printf("Fact: %d",fact(n));
}
```

Output:

```
sum :3amma@amma14:~/Documents$ gcc -o 4 4.c
amma@amma14:~/Documents$ ./4
4
Fact: 24amma@amma14:~/Documents$ gcc -o 5 5.c
```

Space complexity: The space complexity of this problem is O(n).

REASON: There are no arrays or dynamic memory allocation, but the recursive call stack makes the memory usage **depend on n**, resulting in **O(n)** space complexity.

5. Writea program to transpose a 3x3 matrix.

Code:

```
#include <stdio.h>

void main () {
 int arr[3][3];
 for(int i=0; i<3; i++) {
  for (int j=0; j<3; j++){
   scanf("%d",&arr[i][j]);
  }
 }
 for (int i=0; i<3; i++) {
  for (int j=0; j<3; j++) {
   if(i<j) {
    int tem=arr[i][j];
    arr[i][j] =arr[j][i];
    arr[j][i]=tem;
   }
  }
 }
 printf("Transpose: \n");
 for (int i=0; i<3; i++) {
  for (int j=0; j<3; j++){
   printf(" %d ",arr[i][j]);
  }
  printf("\n");
 }
}
```

Output:

```
amma@amma14:~/Documents$ gcc -o 6 6.c
amma@amma14:~/Documents$ ./6
1 2 3
4 5 6
7 8 9
Transpose:
 1  4  7
 2  5  8
 3  6  9
```

Space complexity: The space complexity of this program is O (1).

REASON: constant space. Because the memory used does **not grow with input size**, the total space remains constant.

6. Write a program to find Fibonacci numbers of a given number.

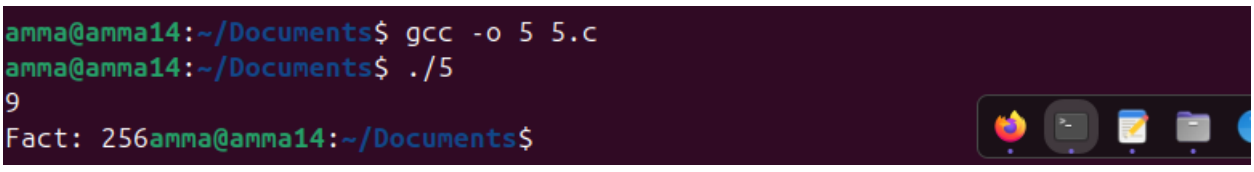
Code:

```
#include <stdio.h>

int fibbo(int n) {
 int s=0; int s1=1; int num=0;
 for (int i=0; i<n;i++) {
   num=s+s1;
   s1=num;
   s=s1;
 }
 return num;
}

void main () {
  int n=0;
  scanf("%d",&n);
  printf("Fact: %d",fibbo(n));
}
```

Output:



Space complexity: The space complexity of this program is O (1).

REASON: constant space. There are **no arrays**, **no recursion**, and **no dynamic memory allocation**, so the memory usage does **not depend on the number of Fibonacci terms printed**