REG NO: 230701045 NAME : Atchaya S

DEPT : CSE - A

TIME COMPLEXITY

QUESTION 2.A

AIM:

Convert the following algorithm into a program and find its time complexity using the counter method.

```
void function (int n)
{
    int i= 1;

    int s =1;

    while(s <= n)
    {
         i++;
         s += i;
     }
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:
A positive Integer n

Output:
Print the value of the counter variable

For example:

Input	Result
9	12

AIM:

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize c to 0 to count operations

Step 4: Initialize i to 1

Step 5: Increment c by 1

Step 6: Initialize s to 1

Step 7: Increment c by 1

Step 8: While s is less than or equal to n, do Steps 8.1 to 8.5

Step 8.1: Increment c by 1

Step 8.2: Increment i by 1

Step 8.3: Increment c by 1 Step 8.4: Add i to s (s += i)

Step 8.5: Increment c by 1

Step 9: Increment c by 1

Step 10: Print the value of c

Step 11: Stop

PROGRAM:

```c
#include<stdio.h>
void function(int n)
{
    int c=0;
    int i=1;
    c++;
    int s=1;
    c++;
    while(s<=n)
    {
        c++;
        i++;
        c++;
        s+=i;
        c++;

    }
    c++;
    printf("%d",c);

}
int main()
{
    int n;
    scanf("%d",&n);
    function(n);
    return 0;
}
```

OUTPUT :

	Input	Expected	Got	
✔	9	12	12	✔
✔	4	9	9	✔

Passed all tests! ✔

RESULT:

The above code is executed successfully and gives expected output.

QUESTION 2.b

Convert the following algorithm into a program and find its time complexity using the counter method.

```
void func(int n)
{
    if(n==1)
    {
      printf("*");
    }
    else
    {
     for(int i=1; i<=n; i++)
     {
       for(int j=1; j<=n; j++)
       {
          printf("*");
          printf("*");
          break;
       }
     }
   }
 }
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:

A positive Integer n

Output:

Print the value of the counter variable

ALGORITHM:

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize c to 0 to count operations

Step 4: If n is equal to 1, go to Step 5, else go to Step 7

Step 5: Increment c by 1

Step 6: Print "*" and go to Step 12

Step 7: Increment c by 1

Step 8: For each integer i from 1 to n, do Steps 9 to 11

Step 9: Increment c by 1

Step 10: For each integer j from 1 to n, do Steps 10.1 to 10.4

Step 10.1: Increment c by 1

Step 10.2: Increment c by 1

Step 10.3: Increment c by 1

Step 10.4: Break out of the inner loop

Step 11: Increment c by 1

Step 12: Increment c by 1

Step 13: Print the value of c

Step 14: Stop

PROGRAM:

```
#include<stdio.h>
void func(int n)
{   int c=0;
    if(n==1)
    { c++;
      printf("*");
    }
    else
    {
        c++;
        for(int i=1; i<=n; i++)
        {
            c++;
            for(int j=1; j<=n; j++)
            {
                c++;
                //printf("*");
                c++;
               //printf("*");
                c++;
                break;
            }
            c++;
        }
        c++;
    }
    printf("%d",c);
}


 int main()
{
     int n;
     scanf("%d",&n);
     func(n);
}
```

OUTPUT:

	Input	Expected	Got	
✔	2	12	12	✔
✔	1000	5002	5002	✔
✔	143	717	717	✔

RESULT:

The above code is executed successfully and gives expected output.

QUESTION 2.C

Convert the following algorithm into a program and find its time complexity using counter method.

```
Factor(num) {
{
   for (i = 1; i <= num;++i)
   {
    if (num % i== 0)
       {
         printf("%d ", i);
       }
    }
 }
```

Note: No need of counter increment for declarations and scanf() and counter variable printf() statement.

Input:
A positive Integer n
Output:
Print the value of the counter variable

ALGORITHM:

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize c to 0 to count operations

Step 4: For each integer i from 1 to n, do Steps 5 to 7

Step 5: Increment c by 1

Step 6: If n is divisible by i (n % i == 0), increment c by 1

Step 7: Increment c by 1

Step 8: Increment c by 1

Step 9: Print the value of c

Step 10: Stop

PROGRAM:

```c
#include <stdio.h>

void Factor(int num) {
    int c = 0;
    for (int i = 1; i <= num; ++i)
    {
        c++;
        if (num % i == 0)
        {
            c++;
        }
        c++;
    }
    c++;
    printf("%d", c);
}

int main() {
    int n;
    scanf("%d", &n);
    Factor(n);
    return 0;
}
```

OUTPUT:

	Input	Expected	Got	
✔	12	31	31	✔
✔	25	54	54	✔
✔	4	12	12	✔

Passed all tests! ✔

RESULT:

The above code is executed successfully and gives expected output.

QUESTION 2.D

Convert the following algorithm into a program and find its time

complexity using counter method.

```
void function(int n)
{
    int c= 0;
    for(int i=n/2; i<n; i++)
        for(int j=1; j<n; j = 2 * j)
            for(int k=1; k<n; k = k * 2)
                c++;
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:
A positive Integer n
Output:
Print the value of the counter variable

ALGORITM:

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize count to 0 to count operations

Step 4: Initialize c to 0

Step 5: Increment count by 1

Step 6: For each integer i from n/2 to n – 1, do Steps 7 to 9

Step 7: Increment count by 1

Step 8: Initialize j to 1 and while j is less than n, do Steps 8.1 to 8.5

Step 8.1: Increment count by 1

Step 8.2: Initialize k to 1 and while k is less than n, do Steps 8.2.1 to 8.2.4

Step 8.2.1: Increment count by 1

Step 8.2.2: Increment c by 1

Step 8.2.3: Increment count by 1

Step 8.2.4: Multiply k by 2 (k = k * 2)

Step 8.3: Increment count by 1

Step 8.4: Multiply j by 2 (j = j * 2)

Step 9: Increment count by 1

Step 10: Increment count by 1

Step 11: Print the value of count

Step 12: Stop

PROGRAM:

```
#include<stdio.h>
void function(int n)
{
    int count=0;
    int c= 0;
    count++;
    for(int i=n/2; i<n; i++){
        count++;
        for(int j=1; j<n; j = 2 * j){
            count++;
            for(int k=1; k<n; k = k * 2){
                count++;
                c++;
                count++;
            }
            count++;
        }
        count++;
    }
    count++;
    printf("%d",count);
}
int main(){
    int n;
    scanf("%d",&n);
    function(n);
}
```

OUTPUT:

	Input	Expected	Got	
✔	4	30	30	✔
✔	10	212	212	✔

Passed all tests! ✔

RESULT:

The above code is executed successfully and gives expected output.

QUESTION 2.E

Convert the following algorithm into a program and find its time complexity using counter method.

```
void reverse(int n)
{
   int rev = 0, remainder;
   while (n != 0)
    {
        remainder = n % 10;
        rev = rev * 10 + remainder;
        n/= 10;

    }
print(rev);
}
```

Note: No need of counter increment for declarations and scanf() and count variable printf() statements.

Input:
A positive Integer n
Output:
Print the value of the counter variable

ALGORITHM:

Step 1: Start

Step 2: Input the integer n

Step 3: Initialize counter to 0 to count operations

Step 4: Initialize rev to 0 and remainder as unassigned

Step 5: Increment counter by 1

Step 6: While n is not equal to 0, do Steps 6.1 to 6.7

Step 6.1: Increment counter by 1

Step 6.2: Calculate remainder as n % 10

Step 6.3: Increment counter by 1

Step 6.4: Update rev to rev * 10 + remainder

Step 6.5: Increment counter by 1

Step 6.6: Divide n by 10 (n /= 10)

Step 6.7: Increment counter by 1

Step 7: Increment counter by 1

Step 8: Increment counter by 1

Step 9: Print the value of counter

Step 10: Stop

PROGRAM:

```c
#include <stdio.h>
void reverse(int n)
{
    int counter=0;
    int rev = 0, remainder;
    counter++;
    while (n != 0)
     { counter++;
         remainder = n % 10;
         counter++;
         rev = rev * 10 + remainder;
         counter++;
         n/= 10;
         counter++;

     }counter++;
  counter++;
//print(rev);
printf("%d",counter);
}
int main(){
     int n;
     scanf("%d",&n);
     reverse(n);
}
```

OUTPUT:

	Input	Expected	Got	
✔	12	11	11	✔
✔	1234	19	19	✔

Passed all tests! ✔

RESULT:

The above code is executed successfully and gives expected output.