Ex:2 **Date**:08.08.24

Name: Veronica Regina Paul **Register number**:230701377

Finding Time Complexity of Algorithms

Problem 1: Finding Complexity using Counter Method

AIM

To find the time complexity of a function using the counter method

ALGORITHM

1)Initialize Counter Variable
Set counter = 0 to keep track of the number of operations performed.

2)Initialize Variables
Set i = 1 and increment counter by 1.
Set s = 1 and increment counter by 1.

3)Input n
Read an integer n from the user.

4)Loop Execution

- Enter a while loop with the condition s <= n.
- For each iteration:
    - Increment counter by 1 for checking the loop condition.
    - Increment i by 1 and increment counter by 1.
    - Update s by adding i to s and increment counter by 1.

5)Exit Loop
When s > n, exit the loop and increment counter by 1 for the final loop check.

6)Output Counter Value
Print the final value of counter to display the total number of operations executed.

PROBLEM

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

PROGRAM

```
#include <stdio.h>

int main()

{

   int counter=0;

   int i=1;

   counter++;

   int s=1;

   counter++;

   int n;

   scanf("%d",&n);

   while (s<=n){

      counter++;

      i++;

      counter++;

      s=s+i;
```

```
        counter++;
    }
    counter++;
    printf("%d",counter);
}
```

OUTPUT

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

Problem 2: Finding Complexity using Counter method

AIM

To find the time complexity of a function using the counter method

Finding Complexity using Counter method

ALGORITHM

1)Check Base Case

- If n == 1, print "*" once and end the function.

2)Outer Loop

- For each i from 1 to n:
    - Run the inner loop.

3)Inner Loop

- For each j from 1 to n (only runs once due to break):
    - Print "*" twice.
    - Use break to exit the inner loop.

4)Output Counter

- Print the counter, which tracks total printf calls to analyze time complexity.

PROBLEM

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

PROGRAM

```
#include<stdio.h>
int main(){
    int n;
    int count=0;
    scanf("%d",&n);
    if(n==1)
    {
      count++;
    }
    else
    {
     count++;
     for(int i=1; i<=n; i++)
     {
       count++;
       for(int j=1; j<=n; j++)
       {
          count++;
          count++;
          count++;
          break;
       }
       count++;
     }
     count++;
   }

   printf("%d",count);

}
```

OUTPUT

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

Passed all tests! ✔

Problem 3:Finding Complexity using Counter Method

AIM

To find the time complexity of a function using the counter method

ALGORITHM

1) Initialize Variables

- Define num to store the input number.
- Set count = 0 to track the number of operations performed.

2) Take User Input

- Read an integer num from the user.

3) Loop Through Possible Divisors

- Use a loop with i running from 1 to num (inclusive) to check for divisors.
- For each iteration:
    - Increment count by 1 for the start of the iteration.
    - Increment count by 1 for checking if condition.
    - If num is divisible by i, increment count by 1 to account for the if statement's body.

4) Final Increment

- After the loop completes, increment count by 1 to account for the final loop check.

5) Output the Counter Value

- Print the value of count, which represents the total number of operations performed.

PROBLEM

```
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

PROGRAM

```
#include <stdio.h>

int main()

{

   int num,count=0;

   scanf("%d",&num);

   for (int i = 1; i <= num;++i)

   {

      count++;

      count++;

      if (num % i== 0)

      {

         count++;

      }

   }

   count++;

   printf("%d",count);

}
```

OUTPUT

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

Problem 4: Finding Complexity using Counter Method

AIM

To find the time complexity of a function using the counter method

ALGORITHM

1. Initialize Variables
    - count = 0 to track operations.
2. Take Input
    - Read integer num.
3. Check Divisors
    - Loop i from 1 to num:
        - Increment count twice for each iteration.
        - If num % i == 0, increment count once.
4. End
    - Increment count once after the loop.
    - Print count.

PROBLEM

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

PROGRAM

```
#include <stdio.h>


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

OUTPUT

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

Problem 5: Finding Complexity using Counter Method

AIM

To find the time complexity of a function using the counter method

ALGORITHM

1. Initialize Variables
    - n: Input integer.
    - count = 0: Tracks the number of operations.
    - c = 0: Counts the innermost loop executions (optional).
2. Take Input
    - Read the integer n from the user.
    - Increment count for variable initialization.
3. Loop Structure
    - Outer Loop: i runs from n/2 to n-1 (approximately n/2 times).
        - Increment count once per iteration.
    - Middle Loop: j starts at 1 and doubles each time, running log(n) times.
        - Increment count once per iteration.
    - Inner Loop: k starts at 1 and doubles each time, running log(n) times.
        - Inside the loop:
            - Increment count twice per iteration.
            - Increment c (optional) to track loop executions.
4. Final Increment
    - Increment count once after all loops are completed.
5. Output
    - Print count to display the total operations.

PROBLEM

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

PROGRAM

```
#include <stdio.h>


int main()

{

   int n,rev = 0,count=0,remainder;

   count++;

   scanf("%d",&n);


   while (n != 0)

   {

      count++;

      remainder = n % 10;

      count++;

      rev = rev * 10 + remainder;

      count++;

      n/= 10;

      count++;

   }

   count++;

   count++;
```

```
    printf("%d",count);
}
```

OUTPUT

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