DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TASK PLAN

Course Code / Course Name: 10211CS306/ COMPETITIVE CODING-1

Year / Semester : 2024-25 / SUMMER

Faculty Name : Dr.D.Prabhu

Class Slot : L13

Task1: Point out the Syntax and Semantic errors in the code snippet, and debug

1. Point out the Semantic errors in the code snippet

```
#include <stdio.h>
int main()
{
   a = 10;
printf("The value of a is : %d", a);
return 0;
}
```

2. Point out the Syntax errors in the code snippet:

```
#include <stdio.h>
int main()
{
int a=2;
if(.)

printf("a is greater than 1");
return 0;
}
```

3. Point out the Syntax errors in the code snippet

```
#include <stdio.h>
int main()
{
int a=2;
int b=2/0;
printf("The value of b is : %d", b);
return 0;
}
```

4. Find the output of the code

```
int main()
{
int x=1,y=5;
printf(“%d”,++(x+y);
return 0;
}
```

5. Find the output of the code

```
int main()
{
int a=2,b=3,c=3;
a=b==c;
printf(“a=%d”,a);
return 0;
}
```

6.Find the output of the code

```
int main()
{
int i=9;
for(i--;i--;i--);
printf("%d",i);
return 0;
}
```

6. Find the output of the code

```
int main()
{
int i;
for(i=5;++i;i-+3)
printf("%d",i);
return 0;
}
```

Task2: Conceptual Questions in Programming

1. What is the value assigned to the variable X if b is 7.

X = 7 >8 ? b << 3:7 >4 ? 7 >> 1: b;

2. What will be printed by the code below?

```
int main()
{
printf(“Hi Friends”+3);
return 0’
}
```

3. What will be printed as the result of the operation below?

```
int main()/
{
char a=5,b=9;
printf(“%d\n”,a&b);
return 0;
}
```

4. What is the value assigned to the variable z

```
int main(void)
{
int a=0,b=;
if(!a)
{
b =!a;
if(b)
a =!b;
}
printf(“%d %d\n”,a,b);
return 0;
}
```

5. What will be printed as the result of the operation below?

```
int main()
{
int x=5,y=-11,a=2,b=8;
int z=x++  - --y * b / a;
printf(“%d”,z);
return 0;
}
```

6. What will be printed as the result of the operation below?

```
int main()
{
char a=5,b=9;
printf(“%d”,b>>1;
return 0;
}
```

7. Find the output:

```
#include <stdio.h>
main()
{
int i;
int *pi = &i;
scanf("%d", pi);
printf("%d\n", i+5);
}
```

8.Find the output:

```
double f(double x)
{
if (abs(x*x - 3) < 0.01) return x;
else return f(x/2 + 1.5/x);
}
```

```
9.charinchar = 'A';
switch (inchar)
{
case 'A' :
printf ("choice A \n") ;
case 'B' :
printf ("choice B ") ;
case 'C' :
case 'D' :
case 'E' :
default:
printf ("No Choice") ;
}
```

10.
What output will be generated by the given code d\segment if:
Line 1 is replaced by “auto int a = 1;”
Line 2 is replaced by “register int a = 2;”

```
int a, b, c = 0;
voidprtFun (void);
int main ()
{
static int a = 1; /* line 1 */
prtFun();
    a += 1;
prtFun();
printf ( "\n %d %d " , a, b) ;
}
voidprtFun (void)
{
staticint a = 2; /* line 2 */
int b = 1;
    a += ++b;
         printf (" \n %d %d " , a, b);
}
```

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

TASK PLAN

Course Code / Course Name: 10211CS306/ COMPETITIVE CODING-1

Year / Semester : 2024-25 / SUMMER

Faculty Name : Dr.D.Prabhu

Class Slot : L13

Task2: Conceptual Questions in Programming

Aim: To Execute the code based on Conceptual Questions.

1. **What will be the output of the following program?**

```
#include<stdio.h>
int main()
    {
func(); func
return 0;
    }
void func()
    {
auto int i =0;
register int j =0;
static int k =0;
i++ ;j++ ; k++ ;
printf( "%d %d %d\n" , i, j k);
    }
```

Error: function not define properly

2. What will be the output of the following program?

```
#include<stdio.h>
int x = 10;
int main()
    {
int x = 20;
    {
int x =30;
printf( "%d\n", x);
    }
    printf( "%d\n", x); return 0;
    }
```

OUTPUT:

```
30
20
```

3. What will be the output of the following program?

```
#include<stdio.h>
int i =0;
void main(); int
main()
    {
printf("main's i = %d\n", i);
i++;
val();
printf("main's i = %d\n",i);
val();
return 0;
    }
void val()
{ i=100;
printf ( "val's i = %d\n");
i++;
    }
```

Error:

Use keyword int main either void main

4. What will be the output of the following program?

```
#include<stdio.h>
float circle(int); int
main()
    {
float area ;
int radius = 1 ;
area = circle ( radius ) ;
printf ( "\n%f", area ) ;
return 0;
    }
float circle(int r)
    {
float a;
a = 3.14*r*r;
return a;
    }
```

OUTPUT:

3.140000

5. What will be the output of the following program?

```
#include <stdio.h>
void display();
int main
    {
printf("Learn C\n");
display();
return 0;
    }
void display()
    {
printf("Followed by C++, C# and Java\n");
main();
    }
```

Error: implicit declaration of function 'main'

6. What will be the output of the following program?

```
#include<stdio.h>
int main()
    {
float a = 5, b = 2;
int c;
c = a % b;
printf("%d", c);
return 0;
    }
```

Error: invalid operands to binary %

7. What will be the output of the following program?

```
#include<stdio.h>
int main()
    {
int i = 2, j = 3, k, l ;
float a, b ;
k = i / j * j ;
l = j / i * i ;
a = i / j * j ;
b = j / i * i ;
printf( "%d %d %f %f", k, l, a, b ) ;
return 0;
    }
```

OUTPUT: 0 2 0.000000 2.000000

8. What will be the output of the following program?

```
#include<conio.h>
char p[] = "The sixth sick sheikh's sixth ship is
sick"; int main()
    {
int i = 0;
while (p[i] != '\0')
    {
putch(p[i]);
i++;
} return 0;
    }
```

Error: putch function is not define

9. What will be the output of the following program?

```
#include<stdio.h>
int main()
    {

printf("%d %d %d\n", sizeof('3'), sizeof("3"), sizeof(3));
return 0;
    }
```

Output:4 2 4

10. What will be the output of the following program?

```
#include<stdio.h>
int main()
    {
char str1[] = { 'H', 'e', 'l', 'l', 'o' };
char str2[] = "Hello";
printf("\n%s", str1);
printf("\n%s", str2);
return 0;
    }
```

OUTPUT:

HelloHello
Hello

Result: The Program is Executed and Verified Successfully.

Task 3 : Solve programming problems by implementing necessary control statements

a).Raj and Raju are thick friends. They decided to play a game . Raju is an excellent guy with good communication skills. Raj decided to test his friend raju with few tounge twisters. Raj will utter some sentence to his friend. Upon listening to Raj now raju needs to write those sentences told by his friend.Then raj will utter a word. The task is replace all the words that has been uttered by raj in the sentence with a special character #. Kindly help raju and raj to complete their game. Refer sample input and output for more details

Aim: To Solve programming problems by implementing necessary control statements Algorithm:

Step 1:-Start.
Step 2:-Take integer variable X.
Step 3:- Divide the variable X with (X-1 to 2).
Step 4:- If A is divisible by any value (X-1 to 2)
it is not prime.Step 5:-Print result .

PROGRAM:

```
#include <stdio.h>
#include <string.h>

int main() {
    char
    sentence[100];
    char replace_char;
    printf("Enter a sentence: ");
    fgets(sentence, 100, stdin);
    printf("Enter a character to
    replace: "); scanf("%c",
    &replace_char);

    for (int i = 0; i < strlen(sentence);
      i++) { if (sentence[i] ==
      replace_char) {
        sentence[i] = '#';
      }

    }

    printf("Modified sentence: %s",
    sentence); return 0;
  }
```

OUTPUT:

Enter a sentence: veltech university is in chennai
Enter a character to replace: u
Modified sentence: veltech #niversity is in Chennai

b) Write a program that generates a random number and asks the user to guess what the number is. If the user's guess is higher than the random number, the program should display "Too high, try again." If the user's guess is lower than the random number, the program should display "Too low, try again." The program should use a loop that repeats until the user correctly guesses the random number.

Algorithm:

1. Generate a random number between a specified range.
2. Initialize a loop that will run until the user correctly guesses the random number.
3. Ask the user to enter a guess for the random number.
4. Check if the guess is equal to the random number. If it is, exit the loop and print a message congratulating the user.
5. If the guess is higher than the random number, print a message saying "Too high, try again".
6. If the guess is lower than the random number, print a message saying "Too low, try again".
7. Repeat steps 3-6 until the user correctly guesses the random number.
8. End the program.

PROGRAM:

```
#include
<stdio.h>
#include
<stdlib.h>
#include
<time.h>

int main() {
  srand(time
  (0));
  int randomNumber =
  rand() % 100 + 1; int
  guess;
  do {
    printf("Guess the number
    between 1 and 100: ");
    scanf("%d", &guess);
    if (guess >
      randomNumber) {
```

```
            printf("Too high, try
            again.\n");
        } else if (guess <
            randomNumber) {
            printf("Too low, try
            again.\n");
        }
    } while (guess != randomNumber);
    printf("Congratulations! You guessed the number %d.\n",
    randomNumber); return 0;
}
```

OUTPUT:

```
 Guess the number between 1 and
100: 80 Too low, try again.
Guess the number between 1
and 100: 90 Too high, try again.
Guess the number between 1
and 100: 81 Too low, try again.
Guess the number between 1
and 100: 82 Too low, try again.
Guess the number between 1 and 100: 83
Congratulations! You guessed the number
83.
```

Result: Thus, the program is written and executed successfully.

Task 4:Number Theory – Level 1 (Includes Factorial, Fibonacci Series, Odd or Even, Sum of Digits, …)

Given an integer **N**. You have to find the number of digits that appear in its factorial, where factorial is defined as, factorial(N) = 1*2*3*4… .... *N and factorial(0) = 1.

Aim: To write a program to find factorial of the numbers.

Algorithm:

1. Read the integer N.
2. Initialize a variable called factorial to 1.
3. Initialize an array called digits of size 10 to 0.
4. Loop from 1 to N and for each iteration, multiply the current factorial with the loop index.
5. Convert the factorial to a string.
6. Loop through each character of the string representation of the factorial and for each digit: a. Convert the digit from string to integer. b. Increment the corresponding element of the digits array.
7. Count the number of non-zero elements in the digits array.
8. Print the count of non-zero elements as the result.
9. End the program

Program:

```
#include <stdio.h>

int countDigitsInFactorial(int n);

int main() {

   int n;
   printf("Enter a positive integer: ");
   scanf("%d", &n);
   int count = countDigitsInFactorial(n);
   printf("Number of digits in %d! = %d\n", n, count);
   return 0;
}

int countDigitsInFactorial(int n) {
   if (n < 0) {
      return 0;
   }
   if (n <= 1) {
      return 1;
   }
   double digits = 0;

   for (int i = 2; i <= n; i++) {
      digits += log10(i);
   }
   return (int) floor(digits) + 1;
}
```

OUTPUT:

```
Enter a positive integer: 5
Number of digits in 5! = 3
```

b) Given a number positive number N, find value of $f_0 + f_1 + f_2 + \ldots. + f_N$ where f_i indicates i'th Fibonacci number.
Remember that $f_0 = 0$, $f_1 = 1$, $f_2 = 1$, $f_3 = 2$, $f_4 = 3$, $f_5 = 5$, …
Since the answer can be very large, **answer modulo 1000000007** should be returned.

Algorithm:

1. Read the positive integer N.
2. Initialize variables f0 and f1 to 0 and 1 respectively.
3. Initialize a variable sum to f0.
4. Loop from 1 to N and for each iteration: a. Calculate the current Fibonacci number fn as f0 + f1. b. Update the values of f0 and f1 as f1 and fn respectively. c. Add the current Fibonacci number fn to sum.
5. Return sum modulo 1000000007.
6. End the program.

Program:

```
#include <stdio.h>

#define MOD 1000000007

int fibonacciSum(int n) {
    int prev = 0, curr = 1, next, sum = 0, i;
    for (i = 0; i <= n; i++) {
        sum = (sum + curr) % MOD;
        next = (prev + curr) % MOD;
        prev = curr;
        curr = next;
    }
    return sum;
}

int main() {
    int N;
    scanf("%d", &N);
    printf("%d\n", fibonacciSum(N));
    return 0;
}
```

OUTPUT

Input:

N = 3

Output:

4

Result: Thus the program is exwcuted and verified successfully

Task 5: Basic Number Theory 1

TIC TAC TOE

King Tle4Ever of Time Limit Exceeded is really fascinated about Tic Tac Toe. He organizes a national level contest for Tic Tac Toe every year in Time Limit Exceeded. (Though I agree you need to be really stupid to loose a game of Tic Tac Toe but for the sake of assume playing Tic Tac Toe for them is same as playing Chess for us :P).

Every year the contest has lots of participants. This year there are n participants from all over the country. Seeing this huge participation he asks Moron a simple .

Suppose participant pi wins wi matches. The king wants to know the sum of wi2 from 1 to n. Now as you already know Moron is not good with maths, he asks you to help him.

Given the value of n find the minimum and maximum value of sum of wi2 from 1 to n. As values can be too large output the values mod 109+7.

Aim: To write and execute the program for given scenario based on Basic Number Theory-1

Algorithm:

1. Read the integer n.
2. Calculate the minimum value of the sum as (n*(n-1)*(2*n-1))/6 modulo 109+7.
3. Calculate the maximum value of the sum as ((n*(n-1))/2 * n*n* - *((n*(n-1))/2) * (2*n-1))/3 modulo 109+7.
4. Print the minimum and maximum values of the sum as output.
5. End the program.

Program:

```
#include <stdio.h>
#include <math.h>

#define MOD 1000000007
int main() {
   int t;
   scanf("%d", &t);
   while (t--) {
      long long n;
      scanf("%lld", &n);
      long long min_sum = ((n - 1) * (n - 1)*n /4) % MOD; // Minimum sum
      long long max_sum = ((n * (n + 1)) * (2 *( n + 1))/(12)) % MOD; // Maximum sum
      printf("%lld %lld\n", min_sum, max_sum);
   }
   return 0;
}
```

OUTPUT:

No of inputs: 2

5

20 30

Problem:

Given two integers, and , a recursive technique to find their GCD is the Euclidean Algorithm.

The algorithm states that, for computing the GCD of two positive integers and , if and are equal, . Otherwise if . There are a few optimizations that can be made to the above logic to arrive at a more efficient implementation.

Algorithm:

1. Read the two positive integers a and b.
2. If b is zero, return a as the GCD.
3. Otherwise, recursively call the function with arguments b and the remainder of a divided by b.
4. Return the result of the recursive call as the GCD.

Program:

```
#include <stdio.h>
int gcd(int a, int b) {
   if (b == 0) {
      return a;
   }
   return gcd(b, a % b);
}
int main() {
   int a, b;
   scanf("%d %d", &a, &b);
   int result = gcd(a, b);
   printf("%d\n", result);
   return 0;
}
```

OUTPUT:

Sample Input

1 5

Sample Output

1

Task 6-Basic Number Theory - 2

Euclidean

Given four integers x1, y1, x2 and y2, which represents two coordinates (x1, y1) and (x2, y2) of a two-dimensional graph. The task is to find the Euclidean distance between these two points.

Euclidean distance between two points is the length of a straight line drawn between those two given points.

Examples:

Input: x1, y1 = (3, 4)
x2, y2 = (7, 7)
Output: 5
Input: x1, y1 = (3, 4)
x2, y2 = (4, 3)
Output: 1.41421

Approach: Since the Euclidean distance is nothing but the straight line distance between two given points, therefore the distance formula derived from the Pythagorean theorem can be used.

Aim: To write and exexute the program based on basic Number Theory-2

Algorithm:

1. Read the values of x1, y1, x2 and y2.
2. Calculate the difference between x2 and x1 and store it in a variable dx.
3. Calculate the difference between y2 and y1 and store it in a variable dy.
4. Calculate the square of dx and store it in a variable dx2.

5. Calculate the square of dy and store it in a variable dy2.
6. Calculate the sum of dx2 and dy2 and store it in a variable d2.
7. Calculate the square root of d2 and store it in a variable distance.
8. Print the value of distance as output.

9. End the program.

Program:

```
#include <stdio.h>
#include <math.h>

int main() {
   int x1, y1, x2, y2;
   double distance;

   printf("Enter x1 and y1: ");
   scanf("%d %d", &x1, &y1);
   printf("Enter x2 and y2: ");
   scanf("%d %d", &x2, &y2);

   distance = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));

   printf("The Euclidean distance between (%d, %d) and (%d, %d) is %lf\n", x1, y1, x2, y2, distance);
   return 0;
}
```

OUTPUT:

Input**:** x1, y1 = (3, 4)
x2, y2 = (7, 7)
Output**:** 5

Input**:** x1, y1 = (3, 4)
x2, y2 = (4, 3)
Output**:** 1.41421

GCD

Given an array of size N of integers with each element denoted as array[i] . In this problem we are given a parameter K and we are supposed to find the size of the largest contiguous subarray whose GCD is atleast K If there is no subarray whose GCD is atleast K , print "0".

INPUT

The first line contains 2 integers N and K the size of the array and the parameter as described in the problem statement The second line containts N space separated integers denoting the array.

OUTPUT

Print a single integer in a single line denoting the maximum contiguous subarray size whose GCD is atleast K.

Contraints

1 <= N <= 500000
1 <= array[i] , K <= 1000000

Sample Input

5 9

10 20 5 15 45

Algorithm:

1. Read the value of N and the array of integers array[] of size N.
2. Read the value of K.
3. Initialize a variable max_len to 0, which will store the length of the largest contiguous subarray whose GCD is at least K.
4. Initialize two variables, left and right, to 0, which represent the left and right indices of the current subarray.
5. Initialize a variable current_gcd to array[0], which represents the GCD of the current subarray.
6. Start a loop with variable i from 1 to N-1: a. Update the current_gcd to be the GCD of the current_gcd and array[i]. b. While the current_gcd is greater than or equal to K: i. Update max_len to be the maximum of max_len and the difference between right and left plus 1. ii. Update current_gcd to be the GCD of current_gcd and array[left]. iii. Increment left by 1. c. Increment right by 1.
7. If max_len is still 0, print "0" as there is no subarray whose GCD is at least K.
8. Otherwise, print the value of max_len as the size of the largest contiguous subarray whose GCD is at least K.
9. End the program.

Program:

```
#include <stdio.h>
int gcd(int a, int b) {
   if (b == 0) {
      return a;
   } else {

  return gcd(b, a % b);
   }
}

int main() {
   int n, k;
   scanf("%d %d", &n, &k);

   int arr[n];
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   int maxLength = 0;
   int length = 0;
   for (int i = 0; i < n; i++) {
      int currentGcd = arr[i];
      if (currentGcd >= k) {
         length = 1;
      } else {
         continue;
      }

      for (int j = i + 1; j < n; j++) {
         currentGcd = gcd(currentGcd, arr[j]);
```

```
            if (currentGcd >= k) {
                length++;
            } else {
                break;
            }
        }

        if (length > maxLength) {
            maxLength = length;
        }
    }

    printf("%d\n", maxLength);
    return 0;
}
```

OUTPUT:

Sample Input

5 9
10 20 5 15 45

Sample Output

2

Result: Thus the program is executed and verified successfully.

xecuted and verified successfully.

Task 7: Primality Tests

To solve programming problems which includes the concept of Prime numbers and its related properties.

a. **Sieve of Eratosthenes** with example
b. **Fermat's Primality Testing** with example
c. **Miller-Rabin Primality** Testing with example

a. Sieve of Eratosthenes with example

The **Sieve of Eratosthenes** is an efficient algorithm to find all prime numbers up to a given number **n**.
Steps:

1. Create a list of numbers from **2 to n**.
2. Start with the first prime (2).
3. Eliminate all multiples of 2 (except 2 itself).
4. Move to the next unmarked number (3) → mark its multiples.
5. Repeat until you reach √n.
6. The remaining unmarked numbers are all primes.

🔹 **Example in C**

```
#include <stdio.h>
#include <stdbool.h>

void sieveOfEratosthenes(int n) {
   bool prime[n+1];   // Boolean array to track primes
   for (int i = 0; i <= n; i++)
      prime[i] = true;  // Assume all numbers are prime initially

   prime[0] = prime[1] = false; // 0 and 1 are not prime

   for (int p = 2; p * p <= n; p++) {
      if (prime[p] == true) {
         // Mark all multiples of p as not prime
         for (int i = p * p; i <= n; i += p)
            prime[i] = false;
      }
   }

   // Print prime numbers
   printf("Prime numbers up to %d are:\n", n);
   for (int i = 2; i <= n; i++) {
      if (prime[i])
         printf("%d ", i);
   }
   printf("\n");
}

int main() {
   int n;
```

```
    printf("Enter the limit: ");
    scanf("%d", &n);

    sieveOfEratosthenes(n);

    return 0;
}
```

🔹 **Example Run**

Input:

Enter the limit: 30

Output:

Prime numbers up to 30 are:

2 3 5 7 11 13 17 19 23 29

B. Fermat's Primality Testing with example

Fermat's Little Theorem:

If **p** is prime and **a** is any integer such that 1 < a < p, then

ap−1≡1(modp)a^{p-1} \equiv 1 \pmod{p}ap−1≡1(modp)

Idea:

- Pick a random number a in range [2, p-2].
- Compute ap−1mod pa^{p-1} \mod pap−1modp.
- If result ≠ 1 → **p is composite**.
- If result = 1 for several random values of a → **p is probably prime**.

⚠️ Note: It's a **probabilistic test**. Carmichael numbers may pass even though they are not prime.

🔹 **Example in C**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function for modular exponentiation (a^b % mod)
long long power(long long a, long long b, long long mod) {
    long long result = 1;
    a = a % mod;
    while (b > 0) {
        if (b & 1)
            result = (result * a) % mod;
        b = b >> 1;   // Divide b by 2
        a = (a * a) % mod;
    }
    return result;
}

// Fermat Primality Test
int isPrimeFermat(int n, int k) {
    if (n <= 1 || n == 4) return 0;
    if (n <= 3) return 1;

    // Try k times
```

```
    for (int i = 0; i < k; i++) {
        int a = 2 + rand() % (n - 4); // random number in [2, n-2]
        if (power(a, n - 1, n) != 1)
            return 0;  // composite
    }
    return 1; // probably prime
}

int main() {
    srand(time(0)); // Seed for random numbers

    int n, k;
    printf("Enter number to test: ");
    scanf("%d", &n);
    printf("Enter number of iterations: ");
    scanf("%d", &k);

    if (isPrimeFermat(n, k))
        printf("%d is probably prime.\n", n);
    else
        printf("%d is composite.\n", n);

    return 0;
}
```

🔹 **Example Run**

Input:
Enter number to test: 97
Enter number of iterations: 5
Output:
97 is probably prime.
Input:
Enter number to test: 91
Enter number of iterations: 5
Output:
91 is composite.

C. Miller-Rabin Primality Testing with example C program

Concept: Miller–Rabin Primality Test

It's a **probabilistic test** but much stronger than Fermat's test.

Idea:

1. Write n−1=2s·dn-1 = 2^s \cdot dn−1=2s·d, where `d` is odd.
2. Pick a random number `a` in `[2, n-2]`.
3. Compute x=admod nx = a^d \mod nx=admodn.
4. If x=1x = 1x=1 or x=n−1x = n-1x=n−1, then continue with next iteration.

5. Otherwise, square `x` repeatedly:
    - If you ever get n−1n-1n−1, then continue.
    - If you never get n−1n-1n−1, then `n` is composite.
6. Repeat the test multiple times with different random `a`.
7. If it passes all, `n` is **probably prime**.

Example C Program

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function for modular exponentiation (a^b % mod)
long long power(long long a, long long b, long long mod) {
    long long result = 1;
    a = a % mod;
    while (b > 0) {
        if (b & 1)
            result = (result * a) % mod;
        b = b >> 1;
        a = (a * a) % mod;
    }
    return result;
}

// Miller test for a single base 'a'
int millerTest(long long d, long long n) {
    long long a = 2 + rand() % (n - 4);
    long long x = power(a, d, n);

    if (x == 1 || x == n - 1)
        return 1;

    // Keep squaring x until d becomes n-1
    while (d != n - 1) {
        x = (x * x) % n;
        d *= 2;

        if (x == 1)  return 0;  // composite
        if (x == n - 1) return 1;
    }
    return 0; // composite
}

// Miller-Rabin primality test
int isPrimeMillerRabin(long long n, int k) {
    if (n <= 1 || n == 4) return 0;
    if (n <= 3) return 1;

    // Find d such that n-1 = 2^s * d
    long long d = n - 1;
    while (d % 2 == 0)
        d /= 2;

    // Perform k iterations
    for (int i = 0; i < k; i++) {
```

```
        if (!millerTest(d, n))
            return 0;
    }
    return 1;
}

int main() {
    srand(time(0));
    long long n;
    int k;

    printf("Enter number to test: ");
    scanf("%lld", &n);
    printf("Enter number of iterations: ");
    scanf("%d", &k);

    if (isPrimeMillerRabin(n, k))
        printf("%lld is probably prime.\n", n);
    else
        printf("%lld is composite.\n", n);

    return 0;
}
```

Example Run

Input:

```
Enter number to test: 97
Enter number of iterations: 5
```

Output:

```
97 is probably prime.
```

Input:

```
Enter number to test: 221
Enter number of iterations: 5
```

Output:

```
221 is composite.
```

Task 8 : Arrays -Introduction, memory allocation (with row/column major), operations (insert, delete, search) sorted and unsorted array, suffix array, subsequence and subarray. -CO3-K3

Raman loves Mathematics a lot. One day his maths teacher gave him an interesting problem. He was given an array 'A' consisting of 'n' integers, he was needed to find the maximum value
Input:

- First line of input contains an integer *T* denoting number of test cases.
- Each test case contains two lines, first line contains integer *n* where *n* is the number of elements in array
- Second line contains *n* space separated integers

A_i. Output:
Print the maximum value of the above give expression, for each test case separated in a new line

Sample Input:

2

3

1 2 5

4

1 2 3 4

Sample output:

5

4

Algorithm:

1. Read the number of elements n in the array A
2. Read the array A
3. Initialize max_val to -1
4. Repeat the following steps for all possible pairs (i, j) where 1 <= i < j <= n:

i. Calculate the value of the expression (A[i] * A[j]) + (i - j)

ii. If the calculated value is greater than max_val, set max_val to the calculated value

5. Print the value of max_val for the current test case
6. End

Program:

```
#include
<stdio.h>
```

```
void rotate(int arr[], int n, int k) {
    k = k % n;
    int temp[k];

  for (int i = 0; i < k; i++) {
        temp[i] = arr[n - k + i];
    }
    for (int i = n - 1; i >= k; i--) {
        arr[i] = arr[i - k];
    }
    for (int i = 0; i < k; i++) {
        arr[i] = temp[i];
    }
}

int main()
    { int n,
    k;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int arr[n];
    printf("Enter the array elements:
    "); for (int i = 0; i < n; i++) {

 scanf("%d", &arr[i]);
    }
    printf("Enter the number of positions to rotate: ");
    scanf("%d", &k);
    rotate(arr, n, k);
    printf("The rotated array is: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

OUTPUT:

Enter the size of the array: 10

1Enter the array elements: 1 2 3 4 5 6 7 8 9 0

Enter the number of positions to rotate: 5

The rotated array is: 6 7 8 9 0 1 2 3 4 5

Problem:

Students have become secret admirers of grade. They find the course exciting and the professors amusing. After a superb Mid Semester examination its now time for the results. The TAs have released the marks of students in the form of an array, where arr[i] represents the marks of the ith student. Since you are a curious kid, you want to find all the marks that are not smaller than those on its right side in the array.

Input Format

The first line of input will contain a single integer n denoting the number of students.

The next line will contain n space separated integers representing the marks of students.

Output Format
Output all the integers separated in the array from left to right that are not smaller than those on its right side.

Constraints

1 <= n <= 1000000

0 <= arr[i] <= 10000

Sample input:

4

5 7 3 6

Sample output:

7

Algorithm:

1. Read the input values of n and the array arr[].
2. Initialize a variable max as the first element of the array arr[0].
3. Traverse the array arr[] from right to left and for each element arr[i] do the following: a. If arr[i] >= max, then set max = arr[i] and print arr[i].
4. End.

Program:

```
#include <stdio.h>

int maxSubArray(int arr[], int n) {
   int max_so_far = arr[0];
   int max_ending_here = arr[0];

   for (int i = 1; i < n; i++) {
      max_ending_here = (arr[i] > max_ending_here + arr[i]) ? arr[i] : max_ending_here + arr[i];
      max_so_far = (max_so_far > max_ending_here) ? max_so_far : max_ending_here;
   }

   return max_so_far;
}

int main()
   { int n;
   printf("Enter the size of array: ");
   scanf("%d", &n);

   int arr[n];
   printf("Enter the elements of array: ");
   for (int i = 0; i < n; i++) {
      scanf("%d", &arr[i]);
   }

   int max_sum = maxSubArray(arr, n);
   printf("The maximum subarray sum is: %d\n", max_sum);

   return 0;
}
```

OUTPUT:

Enter the size of array: 5
Enter the elements of array: 1 2 3 4 5

The maximum subarray sum is: 15

Result: Thus the program is executed and verified successfully

Task 9: String Processing- Basics, String functions (based on different language), String operations, Two algorithms for string pattern matching - Naïve approach and Robin karp approach.-CO3-K3

Merge two strings

Given two strings S1 and S2 as input, the task is to merge them alternatively i.e. the first character of S1 then the first character of S2 and so on till the strings end.
NOTE: Add the whole string if other string is empty.
Input:
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains two strings S1 and S2.
Output:
For each test case, in a new line, print the merged string.
Constraints:
1 <= T <= 100
1 <= |S1|, |S2| <= 10^4
Example:
Input:
2

Hello
Bye abc
def
Output:
HBeylel
o adbecf

Algorithm:

a. Read the first string S1
b. Read the second string S2
c. Initialize two variables i and j to 0
d. Initialize an empty string mergedStr
e. Repeat the following steps while i is less than the length of S1 and j is less than the length of S2:
i. Append the i-th character of S1 to mergedStr
ii. Append the j-th character of S2 to mergedStr
iii. Increment i by 1
iv. Increment j by 1
f. If i is less than the length of S1, append the remaining characters of S1 to mergedStr
g. If j is less than the length of S2, append the remaining characters of S2 to mergedStr
h. Print mergedSt

Program:

```
#include <stdio.h>
#include <string.h>
```

```
int main()
    { int t;
    scanf("%d", &t); // read the number of test cases
```

```
    while (t--) {
        char s1[10005], s2[10005], ans[20005];
        scanf("%s %s", s1, s2); // read the two strings

        int len1 = strlen(s1), len2 =
        strlen(s2); int i, j, k;
        i = j = k = 0;

        // merge the strings alternatively
        while (i < len1 && j < len2) {
            ans[k++] =
            s1[i++]; ans[k++]
            = s2[j++];
        }

        // add the remaining characters from s1 or s2
        while (i < len1) ans[k++] = s1[i++];
        while (j < len2) ans[k++] = s2[j++];

        ans[k] = '\0'; // add null terminator to the merged string

        printf("%s\n", ans); // print the merged string
    }

    return 0;
}
```

Input:
2
Hello
Bye abc
def
Output:
HBeylel
o adbecf

Taking input

You are given two inputs: a(integer), and b(string). You need to take the input and print a and b separated by a space.

Input Format:

First line of input conatins number of testcases T. T testcases follow. For each testcase, there will be one line of input containing a and b.

Output Format:

For each testcase, print a and b separated by a space.

Your Task:

This is a function problem. You need to write the command to take input of a and b inside the function inputData()

Taking input

Constraints:

1 <= T <=

10
$1 <= a <= 10^6$
Input:
2
5

Hell
o 7
Geeks
Output:
5 Hello
7 Geek

Algorithm:

1. Read the number of testcases T from the input.
2. Loop through T testcases
    a. Read the integer a from input.
    b. Read the string b from input using fgets() or scanf("%[^\n]s", b).
    c. Print the values of a and b separated by a space using printf() function.
3. End loop.

Program:

```
#include <iostream>
using namespace std;

void inputData()
   { int t, a;
   string b;
   cin >> t;
   while(t--)
   {
      cin >> a >> b;
      cout << a << " " << b << endl;
   }
}

int main() {
   inputData()
   ; return 0;
}
```

Input:
2
5
Hell
o 7
Geeks
Output:
5 Hello
7 Geek

Result: Thus the program is executed and verified successfully.

Task 10: Bit Map-Introduction, XOR, AND, OR, right shift, left shift.-CO3-K3

Reverse bits of a given 32 bits unsigned integer.
Note:
Note that in some languages, such as Java, there is no unsigned integer type. In this case, both input

and output will be given as a signed integer type. They should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in

Example 2 above, the input represents the signed integer -3 and the output represents the signed integer -1073741825.

Example 1:

Input: n = 00000010100101000001111010011100

Output: 964176192 (00111001011110000010100101000000)

Explanation: The input binary string 00000010100101000001111010011100 represents the unsigned integer 43261596, so return 964176192 which its binary representation is 00111001011110000010100101000000.

Example 2:

Input: n = 11111111111111111111111111111101

Output: 3221225471 (10111111111111111111111111111111)

Explanation: The input binary string 11111111111111111111111111111101 represents the unsigned integer 4294967293, so return 3221225471 which its binary representation is 10111111111111111111111111111111.

Constraints:

The input must be a binary string of length 32.

Algorithm:

1. Take the 32-bit unsigned integer as input in binary format.
2. Convert the binary string into a character array.
3. Initialize two pointers: one pointing to the beginning of the array (i.e., the most significant bit), and the other pointing to the end of the array (i.e., the least significant bit).
4. Swap the bits at the two pointers and increment the first pointer and decrement the second pointer until they meet or cross each other.
5. Convert the modified character array back to a binary string.
6. Convert the binary string to an integer and return the result.

Program:

```
#include <stdio.h>

unsigned int reverseBits(unsigned int num) {
   unsigned int reversed = 0;
   int bits = sizeof(num) * 8;

   for (int i = 0; i < bits; i++) {
      if (num & (1 << i)) {
         reversed |= 1 << (bits - 1 - i);
      }
   }

   return reversed;
}

int main() {
   unsigned int num = 10;
   printf("%u\n", num);
   printf("%u\n",
   reverseBits(num)); return 0;
}
```

OUTPUT:

10
1342177280

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight).

Note:

Note that in some languages, such as Java, there is no unsigned integer type. In this case, the input will be given as a signed integer type. It should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.

In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in Example 3, the input represents the signed integer. -3.

Example 1:

Input: n = 00000000000000000000000000001011

Output: 3

Explanation: The input binary string 00000000000000000000000000001011 has a total of three '1' bits.

Example 2:

Input: n = 00000000000000000000000010000000

Output: 1

Explanation: The input binary string 00000000000000000000000010000000 has a total of one '1' bit.

Example 3:

Input: n = 11111111111111111111111111111101

Output: 31

Explanation: The input binary string 11111111111111111111111111111101 has a total of thirty one '1' bits.

Constraints:

The input must be a binary string of length 32.

3. The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Give two integers x and y, return the Hamming distance between them.

Example 1:

Input: x = 1, y = 4

Output: 2

Explanation:

1 (0 0 0 1)

4 (0 1 0 0)

↑ ↑

The above arrows point to positions where the corresponding bits are different.

Example 2:

Input: x = 3, y = 1

Output: 1

Constraints:

0 <= x, y <= 231 – 1

Algorithm:

Algorithm for counting number of 1 bits in an unsigned integer:

1. Initialize a variable count to 0.
2. Loop through each bit in the 32-bit integer.
3. If the current bit is a 1, increment the count by 1.
4. After looping through all the bits, return the count.

Algorithm for computing the Hamming distance between two integers:

1. Initialize a variable count to 0.
2. Loop through each bit in the 32-bit integers x and y.

3. If the current bit in x is different from the current bit in y, increment the count by 1.
4. After looping through all the bits, return the count.

Program:

```
#include <stdio.h>

unsigned int hammingDistance(unsigned int x, unsigned int y)

 {
   unsigned int dist = 0;
   unsigned int val = x ^ y; // bitwise XOR of x and y

   while (val) {
      // count the number of set bits (i.e., 1s) in val
      dist++;
      val &= val - 1;
   }

   return dist;
}

int main() {
   unsigned int x = 10;
   unsigned int y = 15;
   printf("%u\n", hammingDistance(x, y));
   return 0;
```

OUTPUT:

2

Result: Thus the program is executed and verified successfully.

executed and verified successfully.

Task 11: Recursion and its concepts C03-K3

The Fibonacci numbers, commonly denoted F(n) form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
Given n, calculate F(n).
Example 1:
Input: n = 2
Output: 1
Explanation: F(2) = F(1) + F(0) = 1 + 0 = 1.
Example 2:
Input: n = 3
Output: 2
Explanation: F(3) = F(2) + F(1) = 1 + 1 = 2.
Example 3:
Input: n = 4
Output: 3
Explanation: F(4) = F(3) + F(2) = 2 + 1 = 3.
Constraints:
0 <= n <= 30

Algorithm:

1. If n is 0, return 0
2. If n is 1, return 1
3. Initialize variables a and b to 0 and 1, respectively
4. For i from 2 to n, calculate the next Fibonacci number by setting a = b and b = a + b
5. Return b as the nth Fibonacci number

Program:

```
#include <stdio.h>
int fibonacci(int n) {
   if (n <= 1) {
      return n;
   }
 return fibonacci(n-1) + fibonacci(n-2);
}
int main() {
   int n = 10; // example number

 printf("The %dth Fibonacci number is %d\n", n, fibonacci(n));
   return 0;
}
```

OUTPUT:

The 10th Fibonacci number is 55

Problem: The Tribonacci sequence Tn is defined as follows:
T0 = 0, T1 = 1, T2 = 1, and Tn+3 = Tn + Tn+1 + Tn+2 for n >= 0.
Given n, return the value of Tn.
Example 1:
Input: n = 4
Output: 4

Explanation
:
T_3 = 0 + 1 + 1 = 2
T_4 = 1 + 1 + 2 = 4
Example 2:
Input: n = 25
Output: 1389537
Constraints:
0 <= n <= 37
The answer is guaranteed to fit within a 32-bit integer, ie. answer <= 2^31 - 1.

Álgorithm:

1. If n is 0, return 0.
2. If n is 1 or 2, return 1.
3. Initialize variables t0 = 0, t1 = 1, t2 = 1, and t3 = 2.
4. Loop from 3 to n, and at each iteration, calculate the value of the next term in the sequence by setting t3 = t0 + t1 + t2, then update the values of t0, t1, and t2 as follows: t0 = t1, t1 = t2, t2 = t3.
5. Return the value of t3.

Program:

```
#include <stdio.h>
int fibonacci(int n) {
   if (n <= 1) {
      return n;
   }
   return fibonacci(n-1) + fibonacci(n-2);
}
int main() {
   int n = 10; // example number
   printf("The %dth Fibonacci number is %d\n", n, fibonacci(n));
   return 0;
}
```

OUTPUT:

Enter the number 0f elements:15
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Result: Thus the Program is executed and verified successfully.

Task 12: Elementary data structure algorithms-based Questions and finding complexity.-CO4-K3

Little Shino and Paris

Given a permutation of number from 1 to N. Among all the subarrays, find the number of unique pairs such that and a is maximum and b is second maximum in that subarray.

Input:

First line contains an integer, N . Second line contains N space separated distinct integers, , denoting the permutation.

Output:

Print the required answer.

SAMPLE INPUT

5

1 2 3 4 5

SAMPLE

OUTPUT 4

Q UEUE

Algorithm:

1. Read the input values of N and the permutation array A.
2. Initialize a variable count to 0, to keep track of the number of valid pairs.
3. Loop over all possible subarrays of A, with nested loops i and j:
   a. Find the maximum element maxval and its index maxidx in the subarray [i,j].
   b. Find the second maximum element secondmax in the subarray [i,j] using a separate loop or a priority queue.
   c. If maxval is unique in the subarray and secondmax is also present, increment count.
4. Print the final value of count.

Program:

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

```
            second_max = arr[j];
        }
        if (second_max != -1 && arr[i] == second_max) {
            count++;
            break;
        }
    }

  }
  printf("%d", count);
  return 0;
}
```

OUTPUT:

SAMPLE
INPUT 5
1 2 3 4 5
SAMPLE
OUTPUT 4

FUN SORT

Divide the given array of size equal to the sum of first n natural numbers into subarrays of size 1,2,3,,n. For the xth subarray of size x – if x is odd then sort it in descending order, otherwisesort it in ascending order. Output the sum of 1st element of each subarray.

Algorithm:

1. Calculate the sum of first n natural numbers:
    - Initialize a variable sum to 0
    - Loop through the range from 1 to n+1
    - Add each number to the sum variable
2. Create an array of size equal to the sum calculated in step 1
3. Loop through the range from 1 to n+1:
    - If x is odd:
        - Get the subarray of size x starting from the index sum-x
        - Sort the subarray in descending order
    - Else:
        - Get the subarray of size x starting from the index sum-x
        - Sort the subarray in ascending order
    - Add the first element of the subarray to a variable called result
    - Increase the sum of the indices of the subarray to x
4. Return the variable result as the output

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  int T, n, i, j, k, sum, size;
  scanf("%d", &T);
  while (T--) {
```

```
    scanf("%d", &n);
    size = (n * (n + 1)) / 2;
    int *arr = (int*) malloc(sizeof(int) * size);
    for (i = 0; i < size; i++) {
       scanf("%d", &arr[i]);
    }
    sum = 0;
    k = 0;
    for (i = 1; i <= n; i++) {
       if (i % 2 != 0) {

          for (j = 0; j < i; j++) {
             int max = j;
             for (int l = j + 1; l < i; l++) {
                if (arr[l + k] > arr[max + k]) {
                   max = l;
                }
             }
             int temp = arr[j + k];
             arr[j + k] = arr[max +
             k]; arr[max + k] = temp;
          }
          sum +=
          arr[k]; k++;
       }
       else {
          for (j = 0; j < i; j++) {
             int min = j;
             for (int l = j + 1; l < i; l++) {
                if (arr[l + k] < arr[min + k]) {
                   min = l;
                }
             }
             int temp = arr[j + k];
             arr[j + k] = arr[min + k];
             arr[min + k] = temp;
          }
          sum +=
          arr[k]; k += i;
       }
    }
    printf("%d\n",
    sum); free(arr);
  }
  return 0;
}
```

OUTPUT:

Input:

1

3
1 2 3 4 5 6
Output:
9

Result: Thus the program is executed and verified successfully.

Task 13 : Partial Code Completion.-CO5-K3

1. How many times CppBuzz.com is printed?

```
int main()
{
int a = 0;
while(a++)
;
{
   printf("CppBuzz.com");
}
return 0;
}
```

OUTPUT: infinite “CppBuzz.com”

2. What is output of below program?

```
int main()
{
int i,j,count;
count=0;
for(i=0; i<5; i++);
{
        for(j=0;j<5;j++);
        {
           count++;
        }
}
printf("%d",count);
return 0;
}
```

OUTPUT: 25

3. What is output of below program?

```
int main()
{
```

```
  int i;
  for(i=0; i<5; i++);
  {
    printf("cppbuzz");
  }
  return 0;
}
```

OUTPUT:
Cppbuzz

Cppbuz
z
Cppbuz
z
cppbuzz
cppbuzz

4. What is output of below program?

```
int main()
{
int i,j,k,count;
count=0;
for(i=0;i<5;i++
)
{
  for(j=0;j<5;j++)
  {
    count++;
  }
}
printf("%d",count);
return 0;
}
```

OUTPUT: 25

5. What is output of below program?

```
int main()
{
 int i,j;
 for(i = 0,j=0;i<5;i++)
 {
  printf("%d%d--",i,j);
 }
 return 0;
}
```

OUTPUT:

00--10--20--30--40--

6. What is output of below program?

```
int main()
{
 int i;
 for(i=0; i<5; ++i++)
 {
  printf("Hello");
 }
 return 0;
}
```

ERROR: invalid increment operand

7. What is the output of below program?

```
int main()
```

```
{
 int i;
 for(i = 0,i<5,i++)
 {
   printf("Hello");
 }
 return 0;
```

```
}
```

OUTPUT:

```
Hell
o
Hell
o
Hell
o
Hell
o
Hell
o
```

8. What is output of below program?

```
int main()
{
 for(; ;);
 for(; ;);
    printf("Hello")
; return 0;
}
```

NO OUTPUT

9. What is the output of below program?

```
int main()

{
for(; ;)
for(; ;)
   printf("Hello..");

return 0;
}
```

OUTPUT:
INFINITE “Hello..”

10. What is output of below code?

```
int main()
{
char name[]="Cppbuz";
function “size_of()”
int len;
int
size;
len = strlen(name);
size =
size_of(name);
printf("%d,%d",len,size);
return 0;
}
```

ERROR: invalid/undeclared

11. What is output of below program?

```
int main()
{
const int a = 10;
variable
printf("%d",++a);
return 0;
}
```

ERROR: cannot increment a constant

12. What is output of below program?

```
int main()
{
const int a = 10;
printf("%d",++a + ++a);
return 0;
}
```

ERROR: cannot increment a constant variable

13. What is output of below program?

```
int main()
{
const int a = 10;
printf("%d",--a);
return 0;
}
```

ERROR: decrement of read-only variable 'a'

14. What is storage class for variable A in below code?

```
int main()
{
int A;
A = 10;
printf("%d", A);
return 0;
}
```

RESULT: Class of A integer

15.

```
#include "stdio.h"
char * f();
char a = 'a';
int main(int argc, char *argv[])
{
char *temp = f();
printf("%&", temp);
return 0;
}
char *f()
{ return &a;
}
```

RESULT: Class of a character

Task 14:Practice Problems for Coding Preparation-CO5-K3

N QUEENS

Given a chess board having N×N cells, you need to place N queens on the board in such a way that no queen attacks any other queen.

Input:

The only line of input consists of a single integer denoting N.

Output:

If it is possible to place all the N queens in such a way that no queen attacks another queen, then print N lines having N integers. The integer in ith line and jth column will denote the cell (i,j) of the board and should be 1 if a queen is placed at (i,j) otherwise 0. If there are more than way of placing queens print any of them. If it is not possible to place all N queens in the desired way, then print "Not possible" (without quotes).

Constraints

: $1 \leq N \leq 10$.

SAMPLE INPUT

4

SAMPLE OUTPUT

0 1 0 0

0 0 0 1

1 0 0 0

0 0 1 0

Algorithm:

1. Define a function to check if a queen can be placed in a given cell of the board.
2. Define a function to recursively place queens on the board.
3. In the recursive function, check if all the queens have been placed. If yes, return the board configuration.
4. If not, loop through all the cells in the current row and check if a queen can be placed in that cell using the isSafe() function.
5. If a queen can be placed in a cell, mark the cell as occupied and recursively call the function for the next row.
6. If the recursive call returns a board configuration, return it.
7. If the recursive call does not return a board configuration, unmark the cell and continue the loop for the next cell.
8. If no queen can be placed in any cell of the current row, return None.
9. In the main function, call the recursive function to place the queens and print the board configuration.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_N 10
```

```
int n;
int board[MAX_N][MAX_N];

int is_valid(int row, int col) {
    int i, j;
    for (i = 0; i < n; i++) {

        if (board[row][i] || board[i][col])
            return 0;
    }
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j])
            return 0;
    }
    for (i = row, j = col; i < n && j >= 0; i++, j--) {
        if (board[i][j])
            return 0;
    }
    return 1;
}

int solve(int col) {
    int row, i;
    if (col >=
        n) return
        1;
    for (row = 0; row < n; row++) {
        if (is_valid(row, col)) {
            board[row][col] =
            1; if (solve(col + 1))
                return 1;
            board[row][col] = 0;
        }
    }
    return 0;
}
int main()
    { int i, j;
    scanf("%d", &n);
    if (solve(0)) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++)
            {
                printf("%d ", board[i][j]);
            }
            printf("\n");
        }
    } else {
```

```
      printf("Not possible\n");
   }
   return 0;
}
```

OUTPUT:

SAMPLE INPUT
4
SAMPLE OUTPUT
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0

THE CASTLE GATE

Gudi, a fun loving girl from the city of Dun, travels to Azkahar - a strange land beyond the mountains. She arrives at the gates of Castle Grey, owned by Puchi,the lord of Azkahar to claim the treasure that it guards. However, destiny has other plans for her as she has to move through floors, crossing obstacles on her way to reach the treasure.

The gates of the castle are closed. An integer N is engraved on the gates. A writing on the wall says Tap the gates as many times as there are unordered pairs of distinct integers from 1 to N whose bit- wise XOR does not exceed N.

Help her find the number of the times she has to tap.

Input:

First line contains an integer T, T testcases follow.

Each testcase consists of an integer N.

Output:

Print the answer to each testcase in a
newline. Constraints:

SAMPLE
INPUT 3
4
6
8
SAMPLE
OUTPUT 3
12

Algorithm:

1. Read the input value of T.
2. Repeat the following steps for T test cases:
    a. Read the value of N.
    b. Initialize the variable count to 0.
    c. Iterate i from 1 to N-1.
        1. Iterate j from i+1 to N.
    d. If (i XOR j) is less than or equal to N, increment count.
    e. Print count.
3. End

Program:

```
#include
```

```
<stdio.h>
int countSetBits(int n) {
    int count = 0;
    while (n > 0) {
        n &= (n - 1);
        count++;
    }
    return count;
}
int main()
    { int t, n;
    scanf("%d", &t);
    while (t--) {
        scanf("%d", &n);

        int count = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = i + 1; j <= n; j++) {
                if ((i ^ j) <= n) {
                    count++;
                }
            }
        }
        printf("%d\n", count);
    }
    return 0;
}
```

OUTPUT:
Sample
input: 3 4 6 8

Sample output:
3
12
21

Result: Thus the Program is executed and verified successfully.