Class Notes 8

Python

💡 Question 1

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)`.

Explanation :

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

Fn = Fn-1 + Fn-2
with seed values

F0 = 0 and F1 = 1.
Time Complexity: Exponential,

as every function calls two other functions.

Auxiliary space complexity:

O(n), as the maximum depth of the recursion tree is n.

Solution :

```
import functools

class Solution:
    @functools.cache
```

```
  def fib(self, n: int) -> int:
      if n == 0:
          return 0
      elif n == 1:
          return 1
      else:
          return self.fib(n-1) + self.fib(n-2)
```

💡 Question 2

Given an integer `n`, return `*true` if it is a power of four. Otherwise, return `false`*.

An integer `n` is a power of four, if there exists an integer `x` such that `n == 4x`.

Example 1:

Input: n = 16
Output: true

Example 2:

Input: n = 5
Output: false

Example 3:

Input: n = 1
Output: true

Solution :

```
class Solution:
  def isPowerOfFour(self, n: int) -> bool:
      if (n == 1): return True
      if (n < 1 or n % 4 != 0): return False
      return self.isPowerOfFour(n // 4)
```

💡 Question 3

Given a number, we need to find sum of its digits using recursion.

Examples:

Input : 12345
Output : 15

Input : 45632
Output :20

Explanation :

The step-by-step process for a better understanding of how the algorithm works.

Let the number be 12345.

Step 1-> 12345 % 10 which is equal-too 5 + (send 12345/10 to next step)

Step 2-> 1234 % 10 which is equal-too 4 + (send 1234/10 to next step)

Step 3-> 123 % 10 which is equal-too 3 + (send 123/10 to next step)

Step 4-> 12 % 10 which is equal-too 2 + (send 12/10 to next step)

Step 5-> 1 % 10 which is equal-too 1 + (send 1/10 to next step)

Step 6-> 0 algorithm stops

Time complexity : O(logn) where n is the given number.

Auxiliary space : O(logn) due to recursive stack space.

Solution :

```
def sum_of_digit( n ):
        if n == 0:
                return 0
        return (n % 10 + sum_of_digit(int(n / 10)))

# Driven code to check above
num = 12345
result = sum_of_digit(num)
print("Sum of digits in",num,"is", result)
```

💡 **Question 4**

Given two numbers **a** and **b**, the task is to find the GCD of the two numbers.

Note: GCD (Greatest Common Divisor) or HCF (Highest Common Factor) of two numbers is the largest number that divides both of them.

Input: *a = 20, b = 28*

Output: 4

Explanation: The factors of 20 are 1, 2, 4, 5, 10 and 20. The factors of 28 are 1, 2, 4, 7, 14 and 28. Among these factors, 1, 2 and 4 are the common factors of both 20 and 28. The greatest among the common factors is 4.

Input: *a = 60, b = 36*

Output: 12

Time Complexity: O(log(min(a,b))

Auxiliary Space: O(log(min(a,b))

Solution :

```
def gcd(a,b):

        # Everything divides 0
        if (b == 0):
                return a
        return gcd(b, a%b)

# Driver program to test above function
a = 98
b = 56
if(gcd(a, b)):
        print('GCD of', a, 'and', b, 'is', gcd(a, b))
else:
        print('not found')
```

💡 Question 5

Given an array of integers, print sums of all subsets in it. Output sums can be printed in any order.

Examples :

Input : arr[] = {2, 3}
Output: 0 2 3 5

Input : arr[] = {2, 4, 5}
Output : 0 2 4 5 6 7 9 11

Explanation :

We can recursively solve this problem. There are total 2n subsets. For every element, we consider two choices, we include it in a subset and we don't include it in a subset.

Time complexity : O(2^n)

Auxiliary Space : O(n)

Solution :

```
def subsetSums(arr, l, r, sum=0):

        # Print current subset
        if l > r:
                print(sum, end=" ")
                return

        # Subset including arr[l]
        subsetSums(arr, l + 1, r, sum + arr[l])

        # Subset excluding arr[l]
        subsetSums(arr, l + 1, r, sum)

# Driver code
arr = [5, 4, 3]
n = len(arr)
subsetSums(arr, 0, n - 1)
```

-

JavaScript

💡 **Question 1**

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)`.

Explanation :

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

Fn = Fn-1 + Fn-2
with seed values

F0 = 0 and F1 = 1.
Time Complexity: Exponential,

as every function calls two other functions.

Auxiliary space complexity:

O(n), as the maximum depth of the recursion tree is n.

Solution :

```
function fib(n) {
  if (n == 0) return 0;
  if (n == 2 || n == 1) return 1;
  return fib(n-1) + fib(n-2);
}
```

💡 Question 2

Given an integer `n`, return `*true` if it is a power of four. Otherwise, return `false`*.

An integer `n` is a power of four, if there exists an integer `x` such that `n == 4x`.

Example 1:

Input: n = 16
Output: true

Example 2:

Input: n = 5
Output: false

Example 3:

Input: n = 1
Output: true

Solution :

```
var isPowerOfFour = function(n) {
    if (n === 0) return false;
    if (n === 1) return true;

    if (!(n%4)) return isPowerOfFour(n/4);

    return false;
};
```

💡 Question 3

****Given a number, we need to find sum of its digits using recursion.

Examples:

Input : 12345
Output : 15

Input : 45632
Output :20

Explanation :

The step-by-step process for a better understanding of how the algorithm works.

Let the number be 12345.

Step 1-> 12345 % 10 which is equal-too 5 + (send 12345/10 to next step)

Step 2-> 1234 % 10 which is equal-too 4 + (send 1234/10 to next step)

Step 3-> 123 % 10 which is equal-too 3 + (send 123/10 to next step)

Step 4-> 12 % 10 which is equal-too 2 + (send 12/10 to next step)

Step 5-> 1 % 10 which is equal-too 1 + (send 1/10 to next step)

Step 6-> 0 algorithm stops

Time complexity : O(logn) where n is the given number.

Auxiliary space : O(logn) due to recursive stack space.

Solution :

```
<script>
function sum_of_digit(n)
{
        if (n == 0)
        return 0;
        return (n % 10 + sum_of_digit(parseInt(n / 10)));
}

// Driven code
var num = 12345;
var result = sum_of_digit(num);
document.write( "Sum of digits in "+ num
+" is "+result );

</script>
```

💡 Question 4

Given two numbers **a** and **b**, the task is to find the GCD of the two numbers.

Note: GCD (Greatest Common Divisor) or HCF (Highest Common Factor) of two numbers is the largest number that divides both of them.

Input: *a = 20, b = 28*

Output: 4

Explanation: The factors of 20 are 1, 2, 4, 5, 10 and 20. The factors of 28 are 1, 2, 4, 7, 14 and 28. Among these factors, 1, 2 and 4 are the common factors of both 20 and 28. The greatest among the common factors is 4.

Input: *a = 60, b = 36*

Output: 12

Time Complexity: O(log(min(a,b))

Auxiliary Space: O(log(min(a,b))

Solution :

```
<script>
function gcd(a, b){

// Everything divides 0
if(b == 0){
        return a;
}

return gcd(b, a % b);
}

// Driver code
let a = 98;
let b = 56;

document.write(`GCD of ${a} and ${b} is ${gcd(a, b)}`);

</script>
```

💡 Question 5

Given an array of integers, print sums of all subsets in it. Output sums can be printed in any order.

Examples :

Input : arr[] = {2, 3}

Output: 0 2 3 5

Input : arr[] = {2, 4, 5}
Output : 0 2 4 5 6 7 9 11
Explanation :

We can recursively solve this problem. There are total 2n subsets. For every element, we consider two choices, we include it in a subset and we don't include it in a subset.

Time complexity : O(2^n)

Auxiliary Space : O(n)

Solution :

```
<script>
function subsetSums(arr, l, r, sum)
{

	// Print current subset
	if (l > r)
	{
		document.write(sum + " ");
		return;
	}

	// Subset including arr[l]
	subsetSums(arr, l + 1, r,
			sum + arr[l]);

	// Subset excluding arr[l]
	subsetSums(arr, l + 1, r, sum);
}

// Driver code
let arr = [5, 4, 3];
let n = arr.length;

subsetSums(arr, 0, n - 1, 0);

</script>
```

-

Java

💡 Question 1

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)`.

Explanation :

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

Fn = Fn-1 + Fn-2
with seed values

F0 = 0 and F1 = 1.
Time Complexity: Exponential,

as every function calls two other functions.

Auxiliary space complexity:

O(n), as the maximum depth of the recursion tree is n.

Solution :

```
class Solution {
   public int fib(int n) {
       if(n==0 || n==1){
           return n;
       }
```

```
        else
            return fib(n-1)+fib(n-2);
    }
}
```

💡 Question 2

Given an integer `n`, return `*true` if it is a power of four. Otherwise, return `false`*.

An integer `n` is a power of four, if there exists an integer `x` such that `n == 4x`.

Example 1:

Input: n = 16
Output: true

Example 2:

Input: n = 5
Output: false

Example 3:

Input: n = 1
Output: true

Solution :

```
class Solution {
    public boolean isPowerOfFour(int n) {
        if(n<=0) return false;
        if(n==1) return true;
        if(n%4==0) return isPowerOfFour(n/4);
        else return false;
    }
}
```

💡 Question 3

Given a number, we need to find sum of its digits using recursion.

Examples:

Input : 12345
Output : 15

Input : 45632
Output :20

Explanation :

The step-by-step process for a better understanding of how the algorithm works.

Let the number be 12345.

Step 1-> 12345 % 10 which is equal-too 5 + (send 12345/10 to next step)

Step 2-> 1234 % 10 which is equal-too 4 + (send 1234/10 to next step)

Step 3-> 123 % 10 which is equal-too 3 + (send 123/10 to next step)

Step 4-> 12 % 10 which is equal-too 2 + (send 12/10 to next step)

Step 5-> 1 % 10 which is equal-too 1 + (send 1/10 to next step)

Step 6-> 0 algorithm stops

Time complexity : O(logn) where n is the given number.

Auxiliary space : O(logn) due to recursive stack space.

Solution :

```
import java.io.*;

class sum_of_digits
{
	static int sum_of_digit(int n)
	{
		if (n == 0)
			return 0;
		return (n % 10 + sum_of_digit(n / 10));
	}

	// Driven Program to check above
	public static void main(String args[])
	{
```

```
                int num = 12345;
                int result = sum_of_digit(num);
                System.out.println("Sum of digits in " +
                                                num + " is " + result);
        }
}
```

💡 Question 4

Given two numbers **a** and **b**, the task is to find the GCD of the two numbers.

Note: GCD (Greatest Common Divisor) or HCF (Highest Common Factor) of two numbers is the largest number that divides both of them.

Input: *a = 20, b = 28*

Output: 4

Explanation: The factors of 20 are 1, 2, 4, 5, 10 and 20. The factors of 28 are 1, 2, 4, 7, 14 and 28. Among these factors, 1, 2 and 4 are the common factors of both 20 and 28. The greatest among the common factors is 4.

Input: *a = 60, b = 36*

Output: 12

Time Complexity: O(log(min(a,b))

Auxiliary Space: O(log(min(a,b))

Solution :

```
import java.io.*;

class Test
{
        // Recursive function to return gcd of a and b
        static int gcd(int a, int b)
        {
        if (b == 0)
                return a;
        return gcd(b, a % b);
        }
```

```
	// Driver method
	public static void main(String[] args)
	{
		int a = 98, b = 56;
		System.out.println("GCD of " + a +" and " + b + " is " + gcd(a, b));
	}
}
```

💡 Question 5

Given an array of integers, print sums of all subsets in it. Output sums can be printed in any order.

Examples :

Input : arr[] = {2, 3}
Output: 0 2 3 5

Input : arr[] = {2, 4, 5}
Output : 0 2 4 5 6 7 9 11

Explanation :

We can recursively solve this problem. There are total 2n subsets. For every element, we consider two choices, we include it in a subset and we don't include it in a subset.

Time complexity : O(2^n)

Auxiliary Space : O(n)

Solution :

```
import java.io.*;

class Main {

	static void subsetSums(int[] arr, int l, int r, int sum)
	{

		// Print current subset
		if (l > r) {
			System.out.print(sum + " ");
```

```
                return;
            }

            // Subset including arr[l]
            subsetSums(arr, l + 1, r, sum + arr[l]);

            // Subset excluding arr[l]
            subsetSums(arr, l + 1, r, sum);
        }

        // Driver code
        public static void main(String[] args)
        {
            int[] arr = { 5, 4, 3 };
            int n = arr.length;

            subsetSums(arr, 0, n - 1, 0);
        }
}
```

-

C++

💡 **Question 1**

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)`.

Explanation :

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

Fn = Fn-1 + Fn-2
with seed values

F0 = 0 and F1 = 1.
Time Complexity: Exponential,

as every function calls two other functions.

Auxiliary space complexity:

O(n), as the maximum depth of the recursion tree is n.

Solution :

```
class Solution {
public:
    int fib(int n) {
        if (n == 0 || n == 1) {
            return n;
        } else {
            return fib(n - 1) + fib(n - 2);
        }
    }
};
```

💡 Question 2

Given an integer `n`, return `*true` if it is a power of four. Otherwise, return `false`*.

An integer `n` is a power of four, if there exists an integer `x` such that `n == 4x`.

Example 1:

Input: n = 16
Output: true

Example 2:

Input: n = 5
Output: false

Example 3:

Input: n = 1
Output: true
Solution :

```
class Solution {
public:
    bool isPowerOfFour(int n) {
        if (n <= 0) {
            return false;
        }
        if (n == 1) {
            return true;
        }
        if (n % 4 == 0) {
            return isPowerOfFour(n / 4);
        } else {
            return false;
        }
    }
};
```

💡 **Question 3**

Given a number, we need to find sum of its digits using recursion.

Examples:

Input : 12345
Output : 15

Input : 45632
Output :20
Explanation :

The step-by-step process for a better understanding of how the algorithm works.

Let the number be 12345.

Step 1-> 12345 % 10 which is equal-too 5 + (send 12345/10 to next step)

Step 2-> 1234 % 10 which is equal-too 4 + (send 1234/10 to next step)

Step 3-> 123 % 10 which is equal-too 3 + (send 123/10 to next step)

Step 4-> 12 % 10 which is equal-too 2 + (send 12/10 to next step)

Step 5-> 1 % 10 which is equal-too 1 + (send 1/10 to next step)

Step 6-> 0 algorithm stops

Time complexity : O(logn) where n is the given number.

Auxiliary space : O(logn) due to recursive stack space.

Solution :

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

int sum_of_digit(int n) {
    if (n == 0)
        return 0;
    return (n % 10 + sum_of_digit(n / 10));
}

int main() {
    int num = 12345;
    int result = sum_of_digit(num);
    cout << "Sum of digits in " << num << " is " << result << endl;
    return 0;
}
```

💡 Question 4

Given two numbers **a** and **b**, the task is to find the GCD of the two numbers.

Note: GCD (Greatest Common Divisor) or HCF (Highest Common Factor) of two numbers is the largest number that divides both of them.

Input: *a = 20, b = 28*

Output: 4

Explanation: The factors of 20 are 1, 2, 4, 5, 10 and 20. The factors of 28 are 1, 2, 4, 7, 14 and

28. Among these factors, 1, 2 and 4 are the common factors of both 20 and 28. The greatest among the common factors is 4.

Input: *a = 60, b = 36*

Output: 12

Time Complexity: O(log(min(a,b))

Auxiliary Space: O(log(min(a,b))

Solution :

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

// Recursive function to return gcd of a and b
int gcd(int a, int b) {
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

// Driver method
int main() {
    int a = 98, b = 56;
    cout << "GCD of " << a << " and " << b << " is " << gcd(a, b) << endl;
    return 0;
}
```

💡 Question 5

Given an array of integers, print sums of all subsets in it. Output sums can be printed in any order.

Examples :

```
Input : arr[] = {2, 3}
Output: 0 2 3 5

Input : arr[] = {2, 4, 5}
Output : 0 2 4 5 6 7 9 11
```

Explanation :

We can recursively solve this problem. There are total 2n subsets. For every element, we consider two choices, we include it in a subset and we don't include it in a subset.

Time complexity : O(2^n)

Auxiliary Space : O(n)

Solution :

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

void subsetSums(int arr[], int l, int r, int sum) {
    // Print current subset
    if (l > r) {
        cout << sum << " ";
        return;
    }

    // Subset including arr[l]
    subsetSums(arr, l + 1, r, sum + arr[l]);

    // Subset excluding arr[l]
    subsetSums(arr, l + 1, r, sum);
}

// Driver code
int main() {
    int arr[] = {5, 4, 3};
    int n = sizeof(arr) / sizeof(arr[0]);

    subsetSums(arr, 0, n - 1, 0);

    return 0;
}
```

-