

02

- 2.1. You are developing a feature for a **financial fraud detection system** that monitors unusual transactions. Given a list of n real-valued transactions and a suspicious target amount s , you need to determine whether any **two distinct transactions** in the list sum up exactly to s . Design an algorithm that solves this problem with **better than quadratic time complexity**.
- 2.2. A warehouse robot operates on an $n \times m$ grid, where some cells contain coins (at most one per cell). The robot starts at the **top-left corner** of the grid (cell (1,1)) and must reach the **bottom-right corner** (cell (n,m)) by moving only **right or down** at each step. Whenever the robot visits a cell containing a coin, it automatically collects it. Your task is to help the robot **collect the maximum number of coins** on its way to the destination. Design an algorithm using **dynamic programming** to compute:
- The **maximum number of coins** the robot can collect.
 - The **path** the robot should take to collect them.
- 2.3. A cloud computing company is assigning a set of N **independent tasks**, each with an associated **resource cost** (given as a non-negative integer). These tasks must be divided between **two servers**:
- Server A** handles exactly K tasks
 - Server B** handles the remaining $N - K$ tasks
- Your goal is to design a **greedy algorithm** that partitions the tasks between the two servers such that the **absolute difference between the total resource cost assigned to Server A and Server B is maximized**.

(CLO-3: Prove correctness of an algorithm using loop invariant and induction.)

Marks = [3+4]

03

- 3.1. State the **Loop Invariant**, $I(n)$ of the following algorithm.

ALGORITHM *Palindromes*($A[0 \dots n - 1]$)

Pre-Condition: $n \geq 1$ and $A[0 \dots n - 1]$ is a character array

```
1.  $i \leftarrow 0$ 
2. while( $i < \lfloor n/2 \rfloor$ ) do
3.   if( $A[i] \neq A[n - i - 1]$ ) then
4.     return F
5.   end if
6.    $i \leftarrow i + 1$ 
7. end while
8. return T
```

- 3.2. Establish the **correctness** of the above algorithm using loop invariant that you define in part (3.1). Show all steps i.e., **initialization, maintenance and termination**.

(CLO-4: Analyze best, average, and worst-case behaviors of an algorithm)

Marks = [3+5+2]

04

- 4.1. Suppose you are sorting a list of n elements using **Quick sort**. Imagine a situation where we always get 90% of the items on one side of the split, and 10% on the other. Set up a recurrence relation for this scenario and solve it using recursion tree method.
- 4.2. A plagiarism detection tool uses a brute-force string matching algorithm to locate a specific **pattern** within a larger text document. Suppose the tool is searching for the pattern of length 100 in the text of length 5000 using the brute-force approach. How many **character comparisons** are made by the brute-force algorithm while searching for the pattern in the text.

the **time complexity** of the following code. Use the most suitable asymptotic notation among O , Ω , and Θ to justify the time efficiency class of the given code.

function()

```
i = n;
while (i > 1) {
    j = i;
    while (j <= n) {
        <body of the inner while loop>    // Executes in  $\Theta(1)$  time
        j = 2 * j;
    }
    i = floor(i / 2);
}
```

(CLO-5: Explain the concept of various complexity classes with examples.)

Marks = [4+3]

Consider the following real-world computing problems. For each problem, identify whether it is mostly to be **tractable** or **intractable**. Justify your answer briefly.

Finding a name in a sorted list of 10 million customers.

Trying every possible route for 30 cities to find the shortest path (like in a delivery app).

Sorting a list of 1000 numbers.

Generating all possible passwords of 12 characters using letters, numbers, and symbols.

Suppose we are dealing with three computational problems in the context of computer science:

P1 is a known NP-Complete problem (for example, the **Travelling Salesman Problem**).

P2 is another problem (say, a custom scheduling problem) that is not known to be in NP, but we know that P2 can be reduced to P1 in polynomial time.

P3 is a third problem (perhaps a resource allocation problem in a cloud system), and P1 can be reduced to P3 in polynomial time.

Explain in which complexity class the problem P3 belongs to?

Best of Luck