Problem Statement #01: (30 marks)

Write a **multithreaded** program to compute the **sum of an integer array** by splitting it into **two halves**. Use two threads to calculate the sum of each half independently. The main thread will combine the results and print the partial sums and the total. Ensure dynamic memory allocation for thread results to avoid undefined behavior.

Standard Inputs:

The program reads any array of size **N** (even or odd). For example:

Case 1: 1 2 3 4 5 Case 2: 10 20 30 40

Output:

Print the sum of the first half, the sum of the second half, and the total sum in the following format:

Sum of the first half: Sum of the second half: Total sum:	3 12 15	30 70 100

Problem Statement #02: (30 marks)

Solve the following Critical Section problem with any synchronization technique.

```c
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <unistd.h>

int accountA = 500; // Initial balance of account A
int accountB = 0;   // Initial balance of account B

void *transfer(void *arg) {
    int amount = *((int *)arg);

    if (accountA >= amount) {
        sleep(1); // Some preemption delay by other thread
        accountA -= amount;
        accountB += amount;
    }

    printf("Transfer %d: A=%d, B=%d (Total: %d)\n",
            amount, accountA, accountB, accountA + accountB);
    return NULL;
}

int main() {
    pthread_t thread1, thread2;
    int amount1 = 100, amount2 = 500;

    pthread_create(&thread1, NULL, transfer, &amount1);
    pthread_create(&thread2, NULL, transfer, &amount2);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    printf("Final balances: A=%d, B=%d (Total: %d)\n",
           accountA, accountB, accountA + accountB);
    return 0;
}
```

Problem Statement #03: (30 marks)

Automate the following process with a shell script (auto.sh)

Step 1: User Input

- ➢ Prompt the user to enter a filename.
- ➢ Store the input in a variable.

Step 2: File Existence Check

- ➢ If the file does not exist:
  - ○ Create the file.
  - ○ Notify the user that the file was created.
- ➢ If the file already exists:
  - ○ Inform the user that the file exists and will be edited.

Step 3: Execute Permission Check

- ➢ If the file lacks execute permission:
  - ○ Add execute permission to the file.
  - ○ Notify the user that permission was added.
- ➢ If the file is already executable:
  - ○ Confirm that no changes were needed.

Step 4: Open the File in the Editor

- ➢ Open the file in the **nano or vim** text editor for the user to edit.

Bonus Step: File Metadata & Script Content

Then, **nano** opens, and the user adds the required script content.

1. Print File Metadata:

- ➢ After creating/editing the file, display:
  - File permissions.
  - File owner.
  - File size.
  - Last modification time.

Example Workflow

A user might interact with your script like this:

```
Give filename: my_script.sh
Created new file: my_script.sh
Added execute permission to my_script.sh
File Metadata:
-rwxr-xr-x 1 user user 35 Aug 25 10:00 my_script.sh
```

Hints:

File Test Operators in Bash

Test Expression	Checks If...	Description
`-e FILE`	file exists	True if file exists
`-f FILE`	regular file	True if it's a normal file (not directory/device)
`-d FILE`	directory	True if it's a directory
`-r FILE`	readable	True if file has read permission
`-w FILE`	writable	True if file has write permission
`-x FILE`	executable	True if file has execute permission
`-s FILE`	non-empty file	True if file exists and is not empty
`-L FILE`	symbolic link	True if file is a symbolic link
`-b FILE`	block device	True if file is a block special device
`-c FILE`	character device	True if file is a character special device
`-N FILE`	modified since last read	True if file was modified after last read
`-O FILE`	owned by current user	True if the user owns the file
`-G FILE`	owned by user’s group	True if file belongs to the user’s group